DEGREE PROJECT IN COMMUNICATION SYSTEMS, SECOND LEVEL
STOCKHOLM, SWEDEN 2015

Sanitization of embedded network devices

Investigation of vendor’s factory reset procedures

MAGNUS LARSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Sanitization of embedded network devices

Investigation of vendor’s factory reset procedures

Magnus Larsson
magnus@stril.com

2015-05-07

Master’s Thesis

Examiner and Academic Adviser
Gerald Q. Maguire Jr.

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

Embedded devices such as routers, switches, and firewalls commonly have sensitive information stored on them such as passwords, cryptographic keys, and information about the network around them and services that these device(s) provide. When disposing of or reselling this equipment in the secondary market it is crucial to erase this sensitive information. However, there is an important question that must be asked: *Do the erase commands and routines offered by the device manufacturers actually erase the sensitive data?*

This thesis investigates methods and tools to determine the completeness of this erasure in some common network devices. These methods are used on a sample of networking equipment found to still contain sensitive information after being erased according to vendor recommendations. A computer program was developed to show how this information can be removed.

The information in this document is useful for equipment owners, brokers and others looking to remarket their current equipment; all of whom want to minimize the risk of leaking sensitive data to other parties.

Keywords

Network device, router, switch, sanitization, forensics, flash, EEPROM, configuration erase, rommon. NVRAM, JTAG, programmer, RS-232, terminal, marker probability in data

Sammanfattning

Nätverksutrustning såsom routrar, switchar och brandväggar har ofta känslig information lagrad internt, som lösenord, kryptografiska nycklar, information om nätverket runt dem samt tjänster de tillhandahåller. Om denna utrustning ska säljas på andrahandsmarkanden eller på annat sätt byta ägare är det viktigt att all känslig information raderas. *Men kan man lita på att raderings rutiner och metoder som tillhandahålls av tillverkaren verkligen raderar känslig data?*

Denna avhandling undersöker lämpliga verktyg och metoder för att granska vilken information som minnen i inbyggda system innehåller. Dessa metoder testas praktiskt på några system som visar sig ha kvar känslig information efter att de raderats enligt tillverkarens rekommendationer. Ett datorprogram som demonstrerar hur denna information kan undersökas och raderas finns med som en del av avhandlingen.

Informationen i detta dokument är användbar för ägare av datakomutrustning, mäklare av sådana samt andra som vill minimera risken för att läcka känslig information vid återförsäljning av sin begagnade utrustning.

Nyckelord

Nätverksutrustning, router, switch informations sanering, flash, EEPROM, radera konfigurationer, rommon, NVRAM, JTAG, programmerare, RS-232 terminal, markör sannolikhet i data

Acknowledgments

Thanks to:

Professor Gerald Q. Maguire Jr,

for all the valuable feedback and research help. You have the work capacity exceeding a 10 man around-the-clock research department!

My wife Rubi,

who hasn't seen much of me lately. Thanks for your support.

My friend Fahad,

for providing feedback and listening to my boring talks about marker search probabilities.

Ganesh and Dave,

for keeping me company during all the hours in the Stril Networks lab.

My father Tommy,

for giving valuable feedback on how to better explain the math section.

My grandmother Ingrid,

who pushed me to eventually finish my degree. I will take you to the diploma ceremony in December!

Thanks also to Tommaso De Vivo at xjtag.com for letting me use your JTAG figures and lending me your JTAG tool.

Stockholm, May 2015
Magnus Larsson

Table of contents

List of Figures

List of Tables

List of Output Listings

List of Algorithms

List of Erase Procedures

List of acronyms and abbreviations

AT	Advance Technology
ATA	AT Attachment
BDM	Background Debug Mode
BGA	Ball Grid Array
BSDL	Boundary Scan Description Language
CFI	Common Flash Interface
CLI	Command Line Interface
CPU	Central Processing Unit
DIP	dual in-line package
ECC	Error-Correcting Codes
EPROM	Erasable programmable read only memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
FTL	Flash Translation Layer
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IC	Integrated Circuit
I/O	Input/Output
IOS	Internet Operating System (for a Cisco router)
JTAG	Joint Test Action Group
LAN	Local Area Network
LSB	Least Significant Bit
MAC	Media Access Control (address)
Mb	Megabit
MLC	Multi Level Cell (Flash)
MMU	Memory Management Unit
MSB	Most Significant Bit
NIST	National Institute of Standards and Technology
NVRAM	Nonvolatile Random Access Memory
ONIF	Open NAND Flash Interface Specification
PCB	Printed Circuit Board
PRNG	PseudoRandom Number Generator
RAM	Random Access Memory
RNG	Random Number Generator
SATA	Serial ATA
SDRAM	Synchronous Dynamic Random-Access Memory
SLC	Single Level Cell (flash memory)
SNMP	Simple Network Management Protocol
TAP	(JTAG) Test Access Port
TCK	Test Clock
TCL	Tool Command Language
TDI	Test Data In
TDO	Test Data Out
TFTP	Trivial File Transfer Protocol
TMS	Test Mode Select

TMSC	Test Serial Data
TLC	Tri Level Cell (Flash)
TRST	Test Reset
TTL	Transistor-transistor logic
TSOP	Standard Thin Small Outline Package
UART	Universal asynchronous receiver/transmitter
UNEDA	United Network Equipment Dealer Association
Vcc	positive power supply voltage
VLAN	Virtual LAN
VTP	VLAN Trunking Protocol
WEEE	Waste Electrical and Electronic Equipment

Conventions

Hexadecimal numbers are prepended by 0x in text, such as 0xFF

The number representation in console logs and input/output from various devices will of course have the representation utilized by that particular device. Thus if a given device requires hex values to be entered as *(hex)FF* that notation will be used in the console logs.

"*<snip>*" inside a console log indicates some parts of the log have been removed.

Prefixes in front of a bit (b) or byte (B) are binary prefix. Table 1-1 lists the first examples and also the IEC prefix equivalent [1].

Table 1-1: Binary prefix convention

Prefix	**Value in front of bit or byte**	**IEC binary prefix notation**
(K) Kilo	1024	Ki, kibi
(M) Mega	1024^2	Mi, mebi
(G) Giga	1024^3	Gi, gibi
(T) Tera	1024^4	Ti, tibi

Overlining an electrical signal means the signal is inverted. For example: $\overline{\text{WE}}$ is an inverted Write Enable signal.

1 Introduction

Today there are both economic and environmental sustainability advantages of giving a piece of equipment a new home once it is unneeded in its current deployment. However, a transfer of ownership requires sensitive configuration data to be removed, otherwise this (business or societally) sensitive data could be improperly disclosed. Improper disclosure of configuration data could provide information that might lead to harm to the business or society. For example, knowing the system administrator's password for a previously used piece of equipment might provide an attacker with either the current password used by the original owner for other equipment or insight into their choice of passwords. The later could facilitate a brute force attack on the password (by reducing the search space) of the original owner's existing equipment.

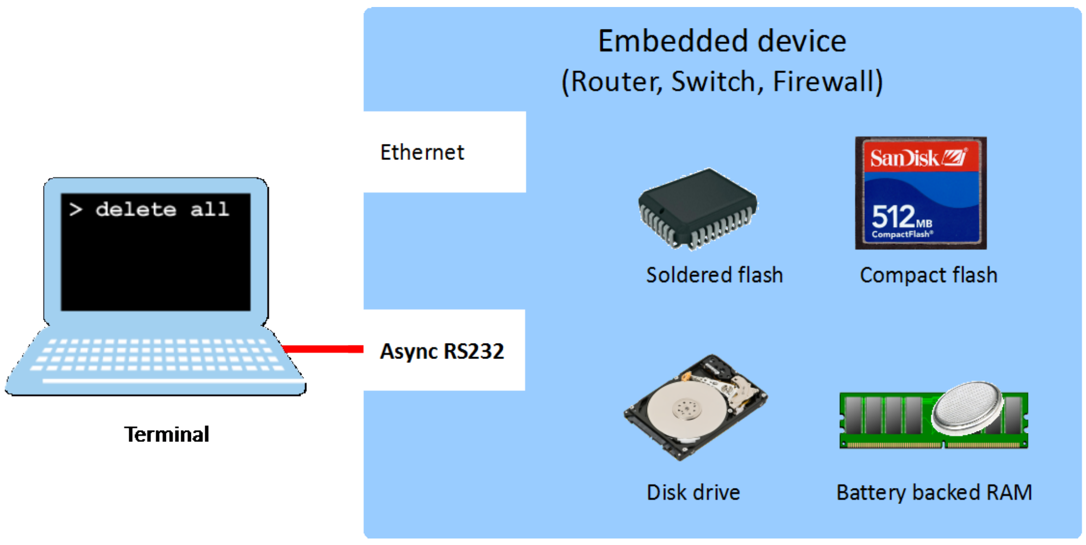
Producing advanced electronic equipment consumes environment resources such as water, energy, and raw materials. The European Union's directive on waste electrical and electronic equipment (WEEE) prioritizes reuse over recycling of equipment to prevent (or at least delay) this equipment from becoming waste [2 Para. 6]. As a result, there will be extensive re-use of equipment by new owners.

This thesis investigates one aspect of the change in ownership for networking equipment by focusing on erasing sensitive data stored as part of the device's configuration information.

1.1 Background

Embedded networks systems such as routers, switches, firewalls, and wireless access points have configuration data and software stored in them. The type of memory used for storing this information includes flash memory chips soldered on a circuit board, removable flash cards, and hard disk drives. The systems are commonly controlled either via a Command Line Interface (CLI) (accessible using an asynchronous RS-232 connection or a ssh/telnet virtual terminal connection via a network interface). In some cases, the device may also be configured and controlled via a web interface provided by a built-in webserver. Unfortunately, the storage used for the configuration data is often inaccessible by means other than the manufacturer's supplied methods, hence when attempting to erase the device's configuration it is important to know if the vendor provided methods actually erase the configuration data.

Figur 1-1: Example of storage devices in an embedded system



1.2 Problem definition

Enterprise networks consist of many devices that communicate with each other. When a network attached device is decommissioned and transferred to another (untrusted) party, it is important to erase sensitive data present in the device to avoid leaking this information to another party.

Examples of sensitive information include:

- Passwords for user or administrator accounts to gain access to the device
- Wi-Fi[*] keys
- Firewall rules
- Information about the internal network, such as VLAN structure and routing
- Protocol authentication keys, such as SNMP community strings, VLAN Trunking Protocol (VTP) password, etc.
- The version of software previously present in the device. This information could be sensitive since it suggests which version of software an enterprise might currently be using in their network and could facilitate attacks. This is especially true if this specific version has known security vulnerabilities.
- The passwords and keys in the decommissioned device could still be used somewhere in the remaining active network (of the former owner), hence if they can be recovered, these values could be used in an attack on the former owner's network.

Instead of selling the equipment, the original owner of the decommissioned equipment may send the device to a scrap yard for recycling/destruction. However, under the EU's WEEE Directive, which promotes reuse before recycling, this is **not** the preferred way to dispose of equipment in an environmentally friendly way. Additionally, this method still presents a risk of leaking sensitive information, as when a device is sent to a scrap yard for destruction there remains a danger that the storage media or the entire device is stolen or resold, thus leading to the risk that the sensitive elements of the configuration could be improperly disclosed to other parties.

As we can see from the above, regardless of the method used to deal with decommissioned networking devices, a method for securely erasing sensitive data is necessary.

1.2.1 Semantics of the word "erase"

To avoid misunderstandings between the meanings of different words describing information removal, I will discuss and define these terms as they will be used in this thesis. These definitions are by no mean universal, but rather the authors' view of their semantics.

According to the Merriam-Webster Dictionary "Erase" means "remove" [3]. In the context of this thesis project, we seek to remove information from a network device. To erase data from a memory means that we remove the information used to represent (i.e., encode) this data.

Does erasing data mean that it can never be recovered? In everyday semantics I would argue that yes, erasing data would mean that it can never be recovered; but it is impossible to ensure that data can *never* be recovered. Because we do not know yet what techniques might be invented in the future that could be used to recover the erased data. For this reason, the definition of the verb *erase* used in this thesis will be:

Erase := "The process of significantly reducing information content."

[*] In this thesis we will use the term Wi-Fi to refer to all wireless local area network equipment that is compatible with one or more of the IEEE 802.11 standards, even if this equipment is not certified by one of the testing laboratories approved by the Wi-Fi Alliance (http://www.wi-fi.org/).

The difference in the ease of reading information before and after the erase operation should be very different. The exact difference will not be explicitly defined, but it should be **much** harder to recover information *after* it has been erased. An example is the written shopping note shown in Figure 1-1. The information, in this case the word “MILK” has been erased in two ways: (1) processed by an eraser gum and (2) overwritten by doodling. Although the results are completely different, they are both examples of methods of erasing information because it harder to recover the original word after the erase operation. For analog information, the “erase” operation can be seen as reducing the signal to noise ratio.

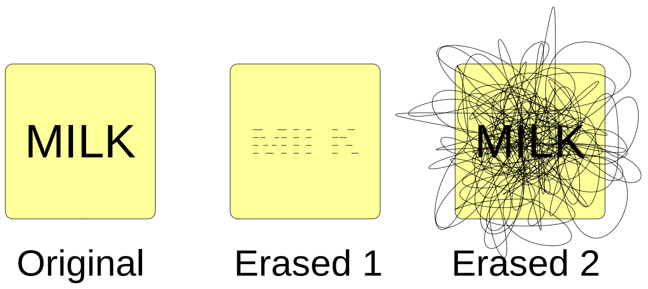


Figure 1-1: Shopping note erase example

1.2.2 Semantics of the word “sensitive information”

In the case of embedded devices completely erasing a memory is impractical and in most cases unnecessary. For instance, the memory may store meta data about the device such as its serial number, Media Access Control (MAC) addresses for the Ethernet interfaces, and other factory set parameters and firmware which are necessary for the device to function properly. In this thesis we focus on a subset of the information in the device’s memory, namely the data which has been stored in the memory of the device during its normal use. Some of this data is private to the current user and could cause harm to this user or other entities if this data were to be disclosed to unauthorized parties. We will refer to this information as *sensitive* information and define it as:

Sensitive information := “Confidential information which could potentially cause harm if disclosed to unauthorized parties”

An example of sensitive information is passwords. However, the actual risk of harm depends on the context. For instance, a password found in a decommissioned router might *not* be sensitive if the previous owner used unique and uncorrelated passwords for each device. In most contexts, the version of the operating system might not be considered sensitive information. However, if a decommissioned network switch has an asset tag glued to it indicating the previous owners’ name, then the combined knowledge of the previous owner and operating system version might facilitate an attack on current devices in the previous owner’s network, which may be running the same version of the operating system or firmware. While techniques exist to profile a device’s OS over the network [4], knowing the likely version of the OS makes it easier to exploit known bugs of the specific version of the operating system and to exploit related security holes.

1.2.3 Semantics of the word “sanitization”

In this paper we define the word sanitization as

Sanitization := “A process that erases sensitive information.”

Sanitizing a memory inside a router does not require that all of the information stored in the memory be fully erased, but rather only the sensitive information must be erased. Sanitization is a weaker form of erasure. Therefore, a memory that is *erased* is also *sanitized*.

1.3 Purpose

The initial purpose of this thesis project is to test if some common embedded network devices typically used in enterprise networks have flaws in their erasure routines. If the manufacturer's erasure routine is unsafe or its safety is unknown, then this project should propose alternative methods to safely erase the sensitive data, i.e., to sanitize the device.

The method used to erase configuration data must be sufficient to ensure that transfer of ownership does *not* risk leaking *any* of the sensitive data. Furthermore, the method should be cost effective and avoid rendering the device unusable.

The existence of an appropriate erasure mechanism would allow reuse of network equipment, benefiting both the environment and all of the parties potentially involved in the transaction, e.g. seller, broker, and buyer. However, the original manufacturer of the equipment might have an economic incentive to sell new devices, rather than facilitating old devices remaining in circulation. However, environmental legislation, such as the EU's WEEE, requires manufacturers **not** to design products in such a way that would prevent their re-use [2Sec. Article 4, Product Design]. Therefore, in the long run, both environmental and commercial customers' demands may place pressure on manufacturers to provide appropriate erasure routines in their software. Note that it is clearly in the interest of the customer who purchases the equipment from the original manufacturer to expect that this vendor will provide appropriate erasure routines, as it is this customer's configuration data that would be exposed!

1.4 Goals

The goals of this project are:

1. Investigate whether common networking devices correctly and completely sanitize sensitive data.
2. Consider various alternatives methods to sanitize this data
3. Select a suitable method from those considered in item (2) to erase sensitive configuration data from a device in satisfactory, easy, and cost effective manner. This erasure mechanism should be suitable to facilitate transfer of ownership of the device.
4. Develop the specifics of the erasure method and implement a "proof of concept" for the devices considered in item (1).
5. Propose a method vendors could use in their new software implementations that would assure complete erasure of sensitive data, i.e., that would guarantee that the device is santized.

1.5 Delimitations

This thesis will consider binary information stored in a networking device. We will assume that storing a new value in a storage cell completely overwrites any earlier data. The process of extracting information erased from storage media below the "below binary" level, e. g., data remanence in erased magnetic media is *outside* the scope of this thesis. Imperfections in current erasure routines investigated will be limited to those related to programming and logical design, rather than physics. Section 2.3.2 gives a brief overview of data remanence. For further details of how to prevent data recovery from magnetic media the reader is referred to the paper "Secure Deletion of Data from Magnetic and Solid-State Memory" by Peter Gutmann [5].

The devices that will be considered will primarily be older devices commonly used in enterprise networks, such as routers, switches, firewalls, and access points from vendors such as Cisco, HP, and Juniper for the following reasons:

- Since these are commonly used device in an enterprise network infrastructure, information recovered from a decommissioned device is likely to be a security threat to the enterprise's network.
- The value in the secondary market of enterprise class equipment is high enough to motivate spending time securely erasing configuration data from the device in order to

prepare it for resale. While equipment that has been previously deployed in a small office / home device, such as that from Netgear or DLink, is more likely to simply be scrapped – rather than resold, as its residual value is low (to very low).

- Enterprise equipment is expected to be configured, deployed, and managed by professionals. One would expect greater security awareness and maturity in the software of a US$3,000 router than a US$50 router.

Older devices were utilized in this thesis project because:

- The risk of destroying a new US$10,000 router in the laboratory while doing this research is high, while the expense versus risk ratio is acceptable for an older device.
- Decommissioned equipment is generally older, hence we investigate them first.

1.6 Structure of the thesis

Chapter 2 presents relevant information about previous work done related to this project. Chapter 3 presents methods for reading and writing memory storage of embedded devices and introduce a method to test existing erasure procedures. Chapter 4 presents the results of the testing a proposed new method for secure erasure. Chapter 5 presents some ideas for improving the completeness of the erasure. Chapter 6 summarizes the results of this thesis project, suggests future work, and discusses some reflections on social, environmental, and ethical issues not addressed elsewhere in the thesis.

2 Related work and useful technologies

This chapter presents the technology behind some storage media and tools useful to access their contents and a summary of references to work already done by others in this field.

2.1 Storage media in embedded systems

Embedded devices commonly store their firmware and configuration in media which preserves the stored information even while power is not applied to the system. Memory media that can be written to more than once and retains data without being powered is called non-volatile memory. Examples of non-volatile memory are hard disk drives, EEPROM, and flash.

2.1.1 Electrically Erasable Programmable Read-Only Memory (EEPROM)

The EEPROM was invented in 1978 at Intel by George Perlegos. EEPROM as an improvement over erasable programmable read only memory (EPROM) which had to be erased by exposure to ultraviolet light [6]. Memory in an EEPROM can be read, erased, and written a single byte at a time (as opposed to flash memory, which has to be erased in blocks) [7]. The random access memory (RAM) like interface of EEPROM makes it easy to add EEPROMs directly to the CPU address/data bus without requiring any glue circuitry in between them [8]. EEPROMs are more expensive to produce than flash, so the typical size of an EEPROM is less than 1 Mb [7].

In the original EEPROM devices, the address and data interfaces are accessed in parallel, e.g. 1 byte of data was passed over 8 pins. There are also serial EEPROMs with a bit serial interface where the address and data are read and written 1 bit at a time [9]. The advantage of the serial interface is that less pins are needed, thus the IC package can be smaller for a given capacity device [10].

Protocols used to access a serial EEPROM includes SPI, I²C, Microwire, UNI/O, and 1-Wire*. The user manual for the Xeltek IS01 programmer contains tips on how to design a printed circuit board (PCB) to permit easy connection in order to permit in-system programming of a soldered chip [11 pp. 8–13]. One particularly useful bit of advice is to have the board power the EEPROM (i.e., to provide a positive power supply voltage, Vcc), but to ensure that no other components try to access the EEPROM while it is being read or programmed by an in-circuit programming device.

2.1.2 Non-volatile Random Access Memory (NVRAM)

NVRAM devices have two methods to maintain their information while power is removed [9]:

- A dedicated battery supplies power.
- Data is saved in an EEPROM at power off and restored after power is returned.

2.1.3 Flash memory

Flash memory is a non-volatile semiconductor storage media developed and was introduced in the 1980s by Toshiba and Intel. There are two major types (NOR and NAND) with different transistor structures used to create each data cell. The table below summarizes some characteristic high level differences of these two types of flash memories. Further details of these memories (how they can be accessed, how they can be read, etc.) is given in the following paragraphs.

* See http://en.wikipedia.org/wiki/EEPROM#Serial_bus_devices

Table 2-1: NOR and NAND flash comparison [12]

	NOR	NAND
Production cost per MB	High	Low
Write performance	Slow	Fast
Erase performance	Slow	Fast
Erase cycles limit	10,000	100,000
Random bit flips in data	Less common	More common
Production imperfections	Less common	More common
Interface	Standard memory interface Can be mapped into memory space as normal memory. CPU can address individual random words and execute code directly from the device.	Differs between vendors Data is read and written in blocks. Code must be copied to RAM before execution. Needs “bad block” management.
Typical application	Small (~4 MB) boot loader mapped into address space	Camera memory card

2.1.3.1 NAND Flash

Originally NAND flash memory could differentiate between two different cell charge levels and thus store one bit per cell. To increase the data density per unit of silicon area and to reduce production costs, flash producers developed ways to store more information per cell. The original 1 bit per cell NAND is now referred to as Single Level Cell (SLC). NAND flash that stores more than one bit per call is called Multi Level Cell (MLC). An example of MLC is a Tri Level Cell (TLC) NAND which stores 3 bits per cell. The drawback of encoding more information into each cell is reliability. Bit errors in data are more likely to happen and the number of write/erase cycles before a cell is worn out decreases by a factor of 10-20 between a SLC and a TLC. [13]

Cells are grouped into “pages” which are the smallest addressable unit. Page sizes differ, but are commonly a multiple of 512 bytes + some extra bytes to store error correction information or flags. For example, 512 bytes +16 additional bytes = 528 bytes. However, it is completely up to the processor that is connected to this memory to decide how to use the page and where and how to store data, Error-Correcting Code (ECC) bits, and flags within a page. To read a page the processor sends a read command and an address to the chip, then the page is placed in an internal register which can then be shifted out.[14]

Two operations are used to modify the contents of a NAND flash: “program” and “erase”. Erase resets all cells to binary true (“1”). The erase operation can only be performed on a group of pages (called a “block”) at the same time. Programing is done page by page and can only invert a binary 1 to a binary 0. Therefore, if we need to write a 1 to a page cell which is currently a 0, then the whole block has to be erased and rewritten.[14]

2.1.3.2 Flash standard interfaces

A group of flash producers joined forces and created a common NAND flash interface standard called “Open NAND Flash Interface Specification” (ONIF) [15]. The specification defines issues such as pinouts, electrical interface, commands, and how producers flag factory defects. There are also commands for probing a compatible chip for it specifications, such as memory organization and capabilities.

The JDEC Common Flash Interface (CFI) [16] specifies a standard way to put supported flash chips into a query mode and to read out parameters, such as manufacturer, memory organization, and timing specifications.

2.1.3.3 *Managed NAND flash*

NAND flash memory in its raw form is unreliable. However, methods can be implemented to make it appear to be more reliable. The main problems and their solutions are:

Bad blocks NAND flash chips are produced on silicon wafers. Manufacturing imperfections cause some storage cells to be defective. Factory defects are typically ~1% of the available storage blocks [17], but even flash chips with 80% factory verified defects have been integrated into consumer grade products [18]. During use additional blocks may become defective. For these reasons bad block handling is crucial. To discover and recover from blocks failing during normal use, error-correcting codes (ECC) must be stored together with the data.

Cell wear Each time a cell is written or erased it loses some of its ability to store data. When data is repeatedly written to the same cells, these cells eventually become unreliable. Flash chips commonly have a defined (on the device's datasheet) maximum number of times a block can be rewritten. For instance the Intel K9F5608X0D 32 MB NAND specifies this maximum number of writes as 100,000 [19p. 3]. A procedure, called "wear leveling", to distribute the writes over as many cells as possible is desirable.

Data retention Over time, charge loss in cells cause the voltage levels in the cell to reach levels where the state can no longer be determined. However, the cell itself is undamaged. For this reason stored data must be read and rewritten periodically to refresh the charge levels in the cells [20] [19].

Duplicate on write and garbage collect A single page cannot be completely rewritten without first erasing the whole block it is located in. This requires the rest of the block to be temporarily read into memory, and then after the block is erased all pages are rewritten, including the new page. It is faster to write the new page to another previously erased block and to copy the unchanged pages from the old block into this previously erased block. The old block is flagged as "invalid" and in the background a garbage collector process will later erase this block. [21p. 3]

To make NAND flash appear to be reliable we need bad block management, ECC correction, and wear leveling. This functionality can be performed by the operating system using a file system specifically designed for flash, such as JFFS2 or YAFFS [22p. 12]. However, new releases and types of NAND flash may be less reliable (but cheaper) and these new flash memories require different algorithms, such as stronger ECC. It is impractical to adjust file system drivers to accommodate all of these changes, thus flash management functionality is increasingly handled by an embedded hardware controller on the flash chip itself (resulting in what is called managed flash) or in a separate customized controller chip [18]. Figure 2-1 shows where this flash management can be placed.

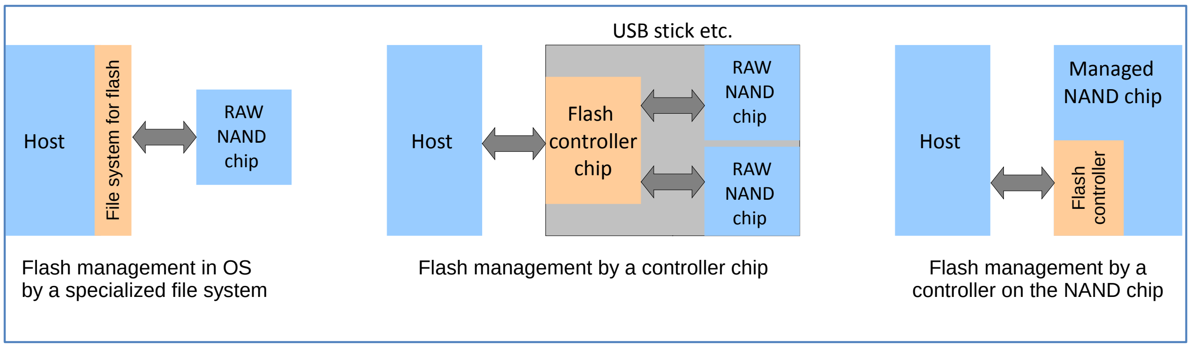


Figure 2-1: Example location of flash management functions

The logic that makes a flash memory system look like a hard disk drive is called the Flash Translation Layer (FTL). Wear leveling is achieved by presenting virtual addresses to the host that are subsequently mapped into physical pages within the actual flash memory device by a translation table. This translation table gives the flash controller the freedom to move data around inside the flash memory and to administer the wear leveling, bad block management, and garbage collection. [21p. 7]

Flash memory chips are commonly packaged on a pluggable memory card for easy handling[*]. Most of these flash memory chips have a controller inside to perform flash management and to provide one of the following host interfaces:

- CompactFlash,
- Solid-state drive (SSD),
- Secure Digital (SD) card,
- USB memory stick, or
- MultiMediaCard (MMC).

An exception to this approach are xD and SmartMedia which are controller-less and thus provide direct access to the NAND chip inside without any wear leveling or address translations.

2.1.3.4 Erasing managed flash

Erasing managed flash from the host interface faces several challenges due to the FTL. Overwriting the same address twice *may not* actually overwrite the old data if the wear leveling and FTL logic are implemented correctly and efficiently – as this data will be written to new physical blocks and not to the previous block(s). For this reason an SSD drive, which is a managed flash device with a IDE/SATA host interface, has special interface commands to erase the storage media: ATA sanitize commands "ERASE UNIT" and sometimes the more potent "ERASE UNIT ENHANCED". The later should erase the entire flash. However, can we trust the implementation to actually do this erasure?

The paper "Reliably Erasing Data From Flash-Based Solid State Drives" [23] investigates these problems and arrives at some conclusions after testing 12 SSD drives:

- The SSD ATA "ERASE UNIT" command reported success in one drive, but left all data intact.
- Degaussing (a procedure used to clear the magnetic fields of Hard Disk Drives) did **not** erase any of the data in the flash chips.
- It is very difficult to ensure that a single file is erased from a managed flash because the FTL usually does not keep track of which parts of the flash could have data related to a logical/virtual address. Therefore, they propose additional logic for the FTL layer so that pages in the flash chip related to a given logical/virtual address can be properly erased.

2.1.3.5 Forensics of managed flash

Reading out data from a raw unmanaged flash is easy: For example, one can de-solder the chips and place them in an external flash chip reader. However, because of the scrambling done by FTL in the controller, it can be very tricky to assemble that data into something useful [21]. Some of the problems are:

- If the controller spreads data to several flash chips, then this spreading needs to be known in order to assemble the data correctly.

[*] Wikipedia has an overview of and photos of each type of card, see: http://en.wikipedia.org/wiki/Comparison_of_memory_cards

- ECC data is interleaved in the raw data. At a minimum, the ECC data would be stripped across the actual storage, but to accurately recover the stored data the ECC algorithm and storage method must be known and understood. The vendor is free to choose the ECC implementation, but there are two standard ways to store the ECC bits [21p. 6].
- Finding out which blocks are bad and unused.
- The FTL translation table and logic must be known and understood in order to fully assemble a correct sequential data-stream as seen from the virtual/logical side.

Each manufacturer can implement their FTL logic and other transformations in a proprietary and undocumented way (due to competitive trade secrets). Joshua White describes the many steps of the reverse engineering process he used when trying to rescue his photos from an SD card [24]. Some sites offer flash chips reader and "recipes" on how to puzzle together the data from various flash controllers and devices [25] [26].

2.1.3.6 Example of managed flash storage: CompactFlash

Here we look under the hood of a common router flash storage card. This flash storage card's primary use is to hold the executable Cisco Internet Operating System (IOS) file, but other files can be stored on it as well (such as configuration files). Figure 2-2 shows such a Compact Flash card taken from a CISCO1812 router. There are three main components: a Samsung Electronics K9F5608U0D flash memory chip, a Hyperstone F2-L16XT flash controller, and a Unigen PCB. The back of the PCB has solder pads for 3 additional flash chips. The exterior case is labeled as a Cisco 32 MB flash card. In the following paragraphs, we examine this flash chip and controller in greater detail.

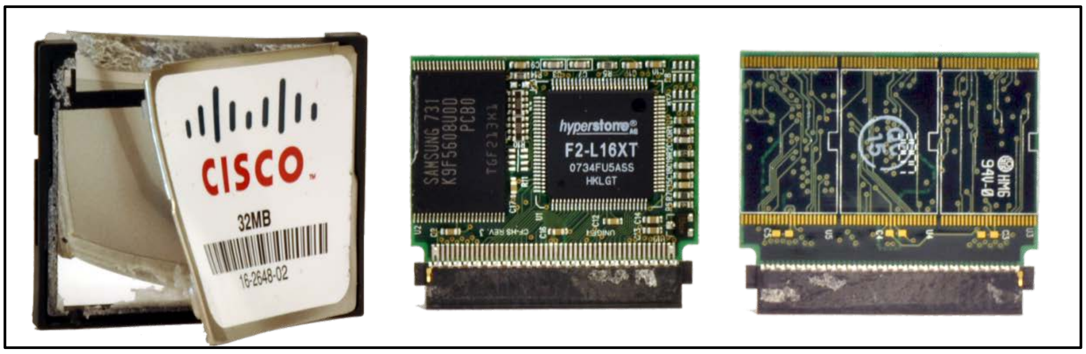


Figure 2-2: 32MB Compact Flash Card

2.1.3.6.1 NAND flash chip

According to the datasheet for the Samsung Electronics K9F5608U0D flash memory chip [19] the flash ship is a 3.3 volt 32 MB NAND memory organized in pages each of 528 bytes (512 + 16 bytes). There are 32 pages per block and 2048 blocks in total. It can endure 100,000 program/erase cycles and data can be stored for 10 years. The chip is delivered from the factory in the all erased (0xFF) state. A flag (a non 0xFF) byte set in the first or second pages of each block indicates that the block is deemed bad by Samsung during their factory testing. Since the flags are written in the area used to store normal data, it is up to the user (in this case a controller) to recognize these bad blocks and save this bad block information in its own table *prior* to using the flash chip to store data. If these bits are cleared (by erasing the block) there is no way to recover the original bad block list from the chip itself.

The flash memory is erased at the block level by sending an "erase" command, the block address, and a "confirm erase command", then wait for about 2 ms and check the data Input/Output (I/O) pins for an indication of success or failure.

2.1.3.6.2 Flash controller chip

The Hyperstone F2-L16XT is a controller with a PCMCIA/Compact Flash and NAND flash driver. It can control up to 10 NAND flash chips. According to its datasheet [27] it is an embedded system containing:

- a 32-bit RISC Hyperstone E1-32X CPU, running at 20 or 40 MHz,
- a 8 Kbyte boot ROM (containing flash access helper routines), and
- 16 Kbyte RAM.

The firmware executed by this controller is stored in a section of the external flash chip it controls. At power on, execution starts with ROM code which tries to locate the firmware in the first attached NAND flash chip, then copies this firmware to the controller's RAM and executes this firmware.

The firmware can be written into the NAND chip prior to soldering it to the board. If the firmware is not found in the first flash chip at boot, the controller will ask for new firmware via the PCMCIA/CompactFlash. When the host sends this firmware it will be copied to the controller's RAM and executed. We will refer to this later process as PCMCIA boot. Note that this firmware could program the first flash chip, i.e., install new firmware into the CompactFlash card.

If the "WE/service mode" and "WAIT" pins are held low during power on, then the controller will perform a PCMCIA boot. The firmware provided by the host could do block erases of the flash chip(s). However, if we erase the entire flash, then the firmware and data structures needed by the controller are lost and would have to be restored to make the card useable again. Moreover, the bad block list of the NAND chip would be lost, so the card would have to be retested. Note that the firmware loaded via the PCMCIA boot need not erase any of the flash contents; hence, all of the previous firmware and bad block list are retained. *If* the firmware loaded via the PCMCIA boot knows where to find the bad block list and the allocated/free blocks list (depending upon the organization of the block allocation), then this firmware could erase all of the blocks, clear the allocated block list, place all of the non-bad blocks in the free list, and re-create the bad block list; thus returning the device back to the state it would have been in after testing (modulo the fact that more blocks may be in the bad block list due to failures detected while the flash memory has been in use).

The controller datasheet does not mention any security measures that would prevent loading new firmware. As a result, we could boot our own firmware supplied from the PCMCIA/CompactFlash interface and once executing in the controller CPU we can permanently overwrite the existing firmware in the NAND flash so it survives a reboot. This could be prevented if the NAND area containing the original firmware was write-protected. Some NAND devices offer block protection functionality [22p. 25], but there is no such feature in the K9F5608U0D-PCB0 used in this compact flash [19]. A similar firmware rewrite of a SD card was shown at the Chaos Computer Congress in 2013 [18].

It is important to note that the host does not have direct access to the NAND chips itself; hence all access is mediated by the controller. The controller starts to execute as soon as it gets power and the controller's CPU can do a lot of operations – even while the host interface is idle (for example, the CPU can perform garbage collection and other management tasks). Here are some examples of what **malicious firmware in the card** could potentially do:

- Scan the files to determine which host the CompactFlash card is in.
- If the host asks for a configuration file, the controller could change it on the fly to bypass the security of the embedded system, i.e. it could change passwords, crypto keys, firewall rules, etc.
- It could compress data stored in the flash in order to make room for hidden storage, while decompressing this data on the fly when the data is requested by the host.
- Scan the stored data for interesting information, such as password and crypto keys, and copy them to hidden storage.

- Get occasional indirect access to host RAM contents - if the host uses the flash card for a virtual memory page file, debug core file, or hibernation file.
- If the host reboots, the controller could deliver an alternative maliciously crafted boot file or boot script which could in turn initialize the network ports and send out data (for example, the hidden data stored by the card).
- If the host tries to erase the flash by sending a predictable erase pattern (such as a series of zeros or ones) it can try to interpret this stream, but maintain the data stored in the NAND chip unchanged. Subsequently when the host wants to verify the erase the controller can regenerate the stream in order to satisfy the host. As a result, from the host side it would appear as everything was erased, but in fact not a single bit was modified.

The controller could be commanded by the host to perform a PCMCIA-boot in order to load custom test firmware capable of checking for potentially malicious firmware stored in the NAND flash. Alternatively, at PCMCIA boot firmware could be installed in RAM which sends back to the host the firmware stored in NAND for verification. Both methods are safe since they only execute the ROM boot code and the test code, but do not execute any potentially unsafe firmware stored in the flash.

We could also de solder the NAND chip, put it in an external flash reader and verify the portion of this storage that contains the firmware, but this could potentially destroy the card or cause it to look like the card shown in Figure 2-2.

2.1.3.7 *Example of unmanaged flash storage: linear PCMCIA Flash card*

The card shown in Figure 2-3 and Figure 2-4 is a 4 MB PCMCIA linear flash card from a CISCO 1601 router.

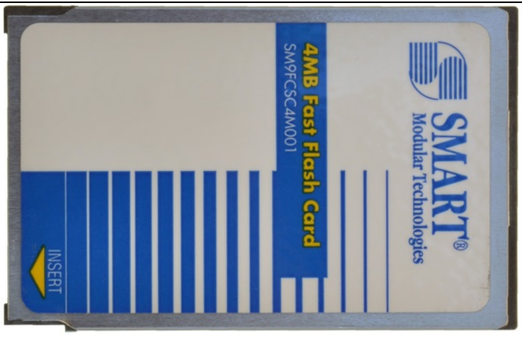


Figure 2-3: PCMCIA Linear Flash (external view)

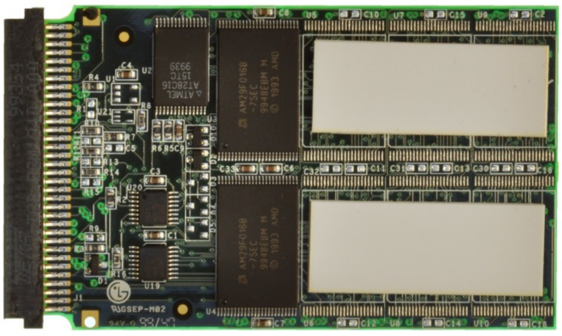


Figure 2-4: PCMCIA Linear Flash (internal view)

The chips inside this card are:

- Two AMD/Spansion **AM29F016B** 2 MB NOR flash chips. The data access to these chips is "memory"-like, i.e., with address and 8 bit parallel data lines.
- One **ATMEL AT28C16** a 2 KB EEPROM [28]. This EEPROM chip probably stores information about the card type and size using a so-called PCMCIA "Card information Structure" [29p. 15].
- Two Texas Instruments **SN74AHCT138DBR** 3-line to 8-line decoder/demultiplexers. These chips probably do some address translation between the PCMCIA and flash chips.

PCMCIA linear flash access is similar to accessing a RAM with an address and data bus [30p. 15]. A host such as the CISCO 1601 can map the flash into its memory space and directly execute software from it. Since there is no controller and thus no FTL, this type of flash card should be easier to erase from the host-side than one with an embedded memory controller.

2.2 Methods to inspect and erase nonvolatile memory

This section will present methods and tools for erasing and inspecting nonvolatile memory useful to the investigate section of the paper.

2.2.1 Vendor's erase procedure

Each vendor typically recommends methods for erasing sensitive data. These could potentially involve one or more of the following procedures:

- Executing commands via a command line interface (using a serial interface or ssh/telnet),
- Pushing buttons during the device's boot sequence,
- Removing a NVRAM's battery,
- Shorting a jumper on the logic board, and/or
- Loading a default configuration from a Trivial File Transfer Protocol (TFTP) server.

This procedure is generally documented in vendor provided user guides or technical notes. Appendix A has a list of erase procedures taken from various vendors' documentation.

2.2.2 Configuration overwrite

One means to erase the device's existing configuration information is to create a new configuration that will overwrite every sensitive configuration parameter by using a vendor supplied management interface. For example, if a (sensitive) SNMP password string is currently set, then the web interface could be used to set this password string to another new (non-sensitive) string. The length of the new string should be sufficiently long to completely overwrite the previous password - if this password is written to the same place in memory. For example, if the configuration parameters are stored at fixed offsets in direct access memory (i.e., without an FTL layer) in an EEPROM. Note that after overwriting the password string, one might also overwrite this with the vendor's default value for this password.

2.2.3 Delete and overwrite free space

If we do not have direct access to the storage and suspect, or know, that the vendor supplied erase methods leave data in unallocated parts of a file system we can attempt to ensure that as much of the accessible storage is overwritten by filling the unallocated space with new data. This data can be provided by any interface offered by the device, including (but not limited to the following): TFTP, Xmodem, Text transfer via the console interface, Duplicating an existing file, or File upload via a web interface.

This data should be sufficiently random that a compressed file system cannot reduce its size. Additionally, the data should be read back and verified (if possible) to ensure that the replacement data was actually stored. When a device cannot be queried in advance to determine how much free space is available and the device simply reports if the write was successful or not, we could use the following algorithm to efficiently fill up the remaining empty space by transferring multiple files:

Algorithm 2-1: Free space overwrite with multiple files

```
Start with a file size F [bytes] approximated to half of the flash memory raw
size (which can be determined form the datasheet)

While F > 0
  Transfer a new file of size F to the device
  If not successful set F = F / 2
End while
```

If new file transfers delete the last transfer (e.g., the file system can only hold one more file) we can do a binary interval search to find the file size F which fills the free space. In the general case we have to assume the file size can be anywhere from zero to the storage capacity of the memory. Any prior knowledge about the file system and block size will reduce the number of sizes to be tried.

Algorithm 2-2: Free space overwrite with one file

```
F = about half the size of the flash device chip size

While ( Transfer a new file of size F is successful )
  F = F*2
End while

HIGH = F   ;this now is the upper boundary for the search interval
LOW = 0

While (HIGH – LOW > 1 )
  F = ( HIGH + LOW )/2
  If (Transfer a new file of size F to the device)
    LOW = F
  Else
    HIGH = F
End while
```

Both algorithms could be improved so the final size which matched the free spaced is remembered and used as start value for F the next time a similar device is to be erased.

2.2.4 JTAG

IEEE standard 1149.1 was developed by the Joint Test Action Group (commonly called JTAG). This standard defines a protocol for controlling integrated circuits (ICs) on a circuit board[31]. Each JTAG compatible IC has extra logic built-in to handle the JTAG protocol. The initial purpose was to provide an interface to set and read the state of IC pins for debugging and troubleshooting systems. The continuity of the paths on the circuit board itself can be tested by driving an output signal on one pin of a chip and reading it at the corresponding pin at the other end of the path.

Today, IC vendors support JTAG as a method to control internal functions, read and write data to internal memories, and to control debugging / execution of a CPU.

A JTAG enabled device has 4 mandatory electrical signals. This set of signals are referred to as the Test Access Port (TAP) and consists of:

- Test Data In (TDI),
- Test Data Out (TDO),
- Test Clock (TCK), and
- Test Mode Select (TMS).

There is often an optional 5th signal called Test Reset (TRST). The IEEE 1149.7 standard defines a two wire interface consisting of TCK and Test Serial Data (TMSC) [32].

The TAP interface can be connected to an IC via physical pins or an internal part of or a sub-blocks/function inside of an IC. The data I/O is serial and clocked in and out of the system by TCK. TAPs can be daisy chained so the TDO of one TAP is connected to the TDI of the next. Figure 2-5 shows how to connect multiple JTAG capable devices together into a JTAG chain. The last TDO is connected to the JTAG interface of the host. Here the host is the device connected to the TAP, it can be a separate host that is used for programming & debugging or it might even be the device itself (so that it can dynamically change its own functionality).

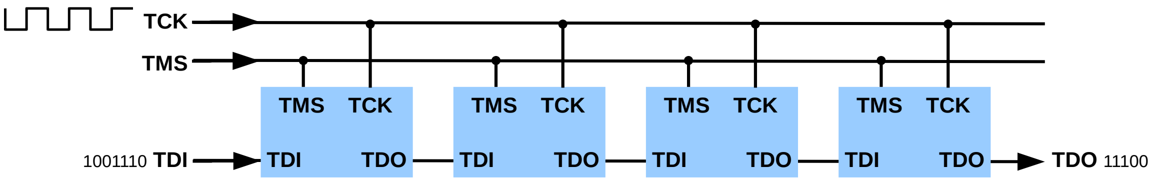


Figure 2-5: JTAG daisy chain, inspired by Figure 4.1 in the JTAG specification [32]

Each JTAG enabled device typically has an instruction register and several data registers. By manipulating the TMS (and the clock) the device state can be configured to receive an instruction. For each instruction, there is a corresponding data register that can be read or written to. A JTAG device must implement a set of mandatory instructions to allow control of its pins, referred to as the boundary scan. Figure 2-6 shows the TAP interface and the internal registers [33].

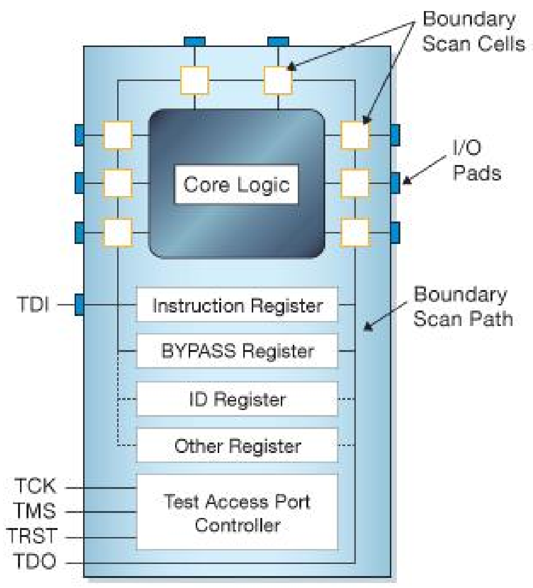


Figure 2-6: IC with JTAG TAP interface. From http://www.xjtag.com

The JTAG specification allows an unknown device chain to be probed by reading the IDCODE data register of each device. The JTAG capabilities of an IC are documented in a Boundary Scan

Description Language (BSDL) file. Unfortunately, not all manufacturers make a BSDL file generally available for each of their devices.

The JTAG interface is a powerful way to read and write to memories of embedded devices. Unfortunately, few commercial embedded devices make any documentation for their JTAG interface or board schematics available, therefore using the JTAG interface together with these devices require some reverse engineering or the cooperation of the manufacturer/vendor.

2.2.4.1.1 Locating the JTAG pins

JTAG is commonly used during the development phase of a product or for troubleshooting and repairs. However, it is rarely intended to be used by anyone other than the manufacturer (or vendor). The final release of a board might not even have a header to connect to the JTAG lines, thus making it necessary to add a header by soldering one on. Even more troubling is that the JTAG lines may be placed in an awkward position, i.e., it may be hard to access without removing daughter boards, expansion cards, etc. In some cases, it is possible to use a sticky adhesive to attach conductors to the pads, thus avoiding the need to solder on a new connector to temporarily connect wires to the surface soldering pads. When boards are being manufactured a “bed of nails” or special fixtures are often used to connect to the board for testing and loading code and data into the board [34].

Unfortunately, there is no standard JTAG connector or pinout. However, there are at least three common alternatives:

- Look for pads arranged as would be used for a 2.54 mm spacing: 2x10, 2x7, 2x5, or 8x1 header or a MIPI Alliance, Inc. MIPI10/20/34 debug connector. Sometimes there will even be a JTAG label on the silkscreen layer of the circuit board.
- CPU manufacturers normally offer JTAG debugging hardware to developers and thus there are some established standard CPU family pinouts. A collection of these can be found at http://www.jtagtest.com/pinouts/ [35].
- Locate a JTAG enabled IC on the board (such as a CPU). Find its JTAG pins via its datasheet and try to trace the connections to these pins to a connector[*]. Today most CPUs have their connectors underneath (i.e., the CPU is connected to the PCB via a ball grid array, BGA) and the chip has to be de-soldered for access. This de-soldering can be easily done using a US$50 hot air gun, but putting the chip back is time consuming and requires an expensive BGA soldering station. As a result this approach is only realistic if a unit can be sacrificed to explore the JTAG pinout.
- Use a specialized tool which can probe a large number of pins (15-30) believed to include the JTAG pins. An example of such a tool is the JTAGulator [36].

2.2.4.1.2 Accessing nonvolatile memory with JTAG

If the memory of interest is in the JTAG chain, we can read or write to it using the supported instructions in its Boundary Scan Description Language (BSDL) file or documentation. Memory devices that are not part of a JTAG chain can still be manipulated indirectly by controlling the CPU via JTAG. The CPU certainly has access to the memory containing the configuration information (otherwise, it could not load a configuration).

In the article “Forensic imaging of embedded systems using JTAG (boundary-scan)” [37], M. F. Breeuwsma proposes two methods to interact with memory devices controlled by the CPU: “Extest mode” and “Debug mode”.

- In *Extest mode* the CPU is put into Boundary Scan mode to control its pins interfacing with the memory of interest. Data is read by driving the signals on the address bus, then

[*] This is typically done by using a continuity check – i. e., looking for a low resistance path from the pins to a potential connector.

waiting for the memory to present the data on the data bus, and then reading out the value presented on the data pins of the CPU. This method controls the physical memory directly, thus any Memory Management Unit (MMU) inside the CPU is bypassed.

The commercial tool XJTAG (see section 2.2.4.1.3) comes with scripts to program flash memory through driving the pins of a JTAG compatible CPU [38].

- In *Debug mode*, the CPU execution is stopped and instructions to read or write the memory are fed into the instruction pipeline. As we are executing instructions in the CPU, the addresses might be translated by an MMU if present and enabled.

The article by M. F. Breeuwsma also address some problems such as watch dog timeouts, refresh of Synchronous Dynamic Random-Access Memory SDRAM, and how to prevent interfering with other hardware during Extest mode.

2.2.4.1.3 JTAG hardware and software tools

There are a number of JTAG hardware and software tools. Some examples of these tools are:

- **JTAGulator** [36] is a hardware tool aiding in locating the 5 JTAG pins from a set of unknown pins. It is an embedded system controlled via an RS-232 over USB interface. The controller has 24 general purpose I/O pins which are connected to the group of pins that may include the JTAG pins.
- **USBJTAG NT** (www.usbjtag.com) is a low cost hardware JTAG interface bundled with graphical and command line software for Windows, Linux, and Apple's MAC OS. It is used to interface to MIPS based CPUs According to the manual [39] it can read and write flash and scan memory ranges. It uses the MIPS EJTAG protocol which is electrically equivalent to JTAG, but has extensions to control the CPU [40].
- **XJTAG** (www.xjtag.com) offers commercial JTAG controllers, software, and scripts to access memories by boundary scanning CPUs.
- **OpenOCD** (openocd.sourceforge.net) is an open source software package for debugging, in-system programming, and boundary scan testing. OpenOCD can be used with many different hardware JTAG adapters [41]. It has support for reading and writing to NOR flash and some NAND flash controllers, as well as debugging some ARM CPUs. [42p. 1]. The user interface is command line based. Scripting is done using Tool Command Language (TCL). A function called "autoprobe" reads out the IDCODE entries for every device in an unknown scan chain. Support for boundary scans seems limited, but there is a user TCL script add on that can help [43].

2.2.5 Other debug interfaces

There are a number of other debugging interfaces. For example, Freescale Semiconductor's CPUs have a Background debug mode (BDM) with an in-circuit debugging interface.

Additionally, the target system may have a monitor with debugging capabilities built in into its bootstrap program. An example is Cisco's Rommon that has commands to inspect and modify memory. Some versions of Rommon have a hidden command set with additional debugging functionality which can be activated by calculating a challenge password. The hidden Rommon commands can show how the storage devices are mapped into physical memory [44].

2.2.6 Custom software method

Another approach to access non-volatile storage is to write a custom program to execute in the CPU of the device of interest to inspect and erase storage media. Software executing in the CPU will have complete control of the device's memory space and can access the non-violate storage in the same way as the vendor's software can. Although proprietary embedded systems utilize standard components, it can be difficult to write software which will run on these device. Several problems exist:

- The device could require software to be signed by specific keys before executing it.
- Documentation for developing software for the device is typically not public; hence, developing code for a given device may require a lot of reverse engineering.
- Legal restrictions may prevent reverse engineering the vendor's code and require that any new custom code be approved by the vendor.[45]
- Accessing memory mapped into the address/data bus such as an EEPROM would be quite easy, but other devices such as managed NAND flash might need special methods to control it, i.e. a device driver.

In order to apply this approach it is necessary to know at least the following about the target device in order to write custom software:

- CPU architecture,
- how and where storage devices are mapped into memory,
- binary format accepted by a boot-loader (compression, checksums),
- a method of transferring the executable file to the device, and
- driver logic to interface to a control port (such as an RS-232, USB, or network port) in order to see what is going on.

Some prior work has already done to make custom software for commercial embedded network systems. Some examples of this are:

- a version of Linux (ucLinux) has been ported to a CISCO 2500 series router [46],
- a lot of custom firmware has been developed for the Linksys WRT-54GL wireless router (after the source code had to be released to the public to comply with GNU licensing)[47], and
- the site http://www.linux-mips.org/wiki/Cisco has information about CPUs used in various Cisco devices, the binary format, as well as a link to a "Hello World" program source code for the CISCO 3600 router series[48].

2.2.7 Hidden debugging console ports

Some embedded network devices, such as the Linksys E-1000, do not present a serial console to the user. For this device all configuration is performed via an Ethernet port (using either a web interface or telnet/ssh console). If the password or IP address is lost a push button restores a default factory configuration with a known IP address and passwords settings [49]. IP telephones frequently work in the same way, as they can be configured via the keypad or via network at boot time (typically using a combination of DHCP with vendor extensions and TFTP access to boot & configuration files).

Many of these devices have an internal serial console port for debugging, troubleshooting, and system setup by the manufacturer. However, as this interface was not intended to be accessed by the user no serial connector is available outside chassis and in many cases there is no connector on the PCB. Finding this port could provide an alternative means to investigate and erase the device's sensitive configuration information. Jonathan Claudius describes a procedure for finding and connecting to such a port on the PCB in his article "Getting Terminal Access to a Cisco Linksys E-1000" [50].

It is possible to automate the search for the serial console transmit and receive lines and the bit rate used, among an unknown set of pins by using specialized hardware. The JTAGulator can identify changing serial pins at transistor-transistor logic (TTL) levels [36]. Another Arduino based tool, RS232Enum, does a similar job [51]. A RS-232 line driver chip, such as the MAX232 [52], can be used to convert between TTL and RS-232 signal levels.

2.2.8 External memory reader / programmer

Flash and EEPROM chips can be read and programmed in a device called a “programmer”. The unsoldered IC is placed into a custom socket that connects the programmer to the IC’s pins. The programmer provides power to the chip and comes with custom computer software to read, write, and erase many different vendors’ chips.

Table 2-2 shows a sampling of the flash and EEPROM devices used in some common embedded systems.

Table 2-2: Flash and EEPROM ICs in some selected embeded systems

Device	**Manufacturer**	**Vendor part#**	**Form Factor** [53]
Netscreen (Juniper) NS-5XP-105 firewall	AMD	am29dl323dt	TSOP48
CISCO 1812 router	AMD	am29lv160DB	TSOP48
CISCO 32MB compact flash#1 from 1812	Samsung	K9F5608U0D	TSOP48
CISCO 32MB compact flash#2 from 1812	Samsung	K9F1G08U0B	TSOP48
CISCO CP7911 IP phone	Spansion	S29GL128N10TF102	TSOP56
CISCO WS-C2924-XL-EN switch	Intel	DA28F320J5	SSOP56
CISCO WS-C2960-24-TT-L switch	Spansion	S29GL256P11TA101	TSOP56
HP J9085A, 2610-24 switch	STI	M29W128GH	TSOP56
CISCO2610 router	Sharp	LH28F008SAT-85	TSOP40
CISCO2611XM router	Intel	E28F640	TSOP56
CISCO2610 NVRAM	CSI	CAT28C256	PLCC32
CISCO AIR-LAP1131AG-2-K9 Flash	Intel	128J3D	FBGA64
CISCO AIR-LAP1131AG-2-K9 EEPROM	CSI	24C08W1	SOIC8

A low end programmer such as the True-USB PRO GQ-4X Willem Programmer [54] is inexpensive (around US$100 without adapters], but does not support TSOP56. A high end programmer that also supports TSOP56 packages is the Xeltek SuperPro 6100 (around US$2000 without adapters) [55].

Some test clips exist that allow a chip to be read and programmed while soldered on a board, see for example [56]. Jeong Wook made his own external NAND flash reader and content decoder [57].

2.3 Previous work and useful information

This section summarizes some useful related work performed by others.

2.3.1 U.S. National Institute of Standards and Technology (NIST)

The draft NIST Special Publication 800-88 Revision 1 “Guidelines for Media Sanitization” defines three actions/levels to perform sanitization on media in general ([58] page 8):

CLEAR	Rewrite or if rewriting is not possible, then use the vendor’s factory reset routines.
PURGE	Use a sanitization technique that makes data recovery infeasible.
DESTROY	Similar to PURGE, but with the addition that the media cannot be reused.

Table A-2 on page 26 of [58] gives advice on Networking Device Sanitization, by stating that PURGE on networking devices should be used with caution, hence they recommend using

DESTROY instead. However, as noted in Chapter 1 of this thesis, this is incompatible with the EU's WEE Directive.

2.3.2 Analog data remenance of Hard Disk Drives

Peter Gutmann describes data remenance on magnetic media (such as disk drives) in [59]. This problem occurs because on a hard disk binary data is encoded by using different analog levels of magnetization following a circular path on the disk(s). He proposes two ways of recovering data:

- When writing to a position on the disk the new analog value will be a mix of the new value and the old. By reading back the analog data with a sensitive read head the old data can be reconstructed.
- The second recovery method uses the fact that the write head is not always perfectly aligned to the center of the track, thus information might still be present along the sides of the track. Thus, it may be possible to read this data back by purposely shifting the head to the side of the track.

Gutmann further proposes overwrite patterns optimized for various types of magnetic encodings to minimize data remenance.

2.3.3 Embedded system analysis

The paper “Blackbox JTAG Reverse Engineering” by Felix Domke [60] proposes methods to explore the JTAG interface for an undocumented device. The process he describes is:

1. Locate the JTAG pins.
2. Measure the Instruction Register (IR) length (by a proposed method)
3. Iterate over the IR and for each of the data registers (DR) determine their characteristics (such as their length and if data are consistent during several reads). Those characteristics could be a clue to the DR's purpose. For example, a very long DR could be a boundary scan register, a zero length register is probably a command issued by just addressing it, etc.

Nathan Fain and Vadik's presentation “JTAG/Serial/FLASH/PCB Embedded Reverse Engineering Tools and Techniques”[*] at the 27th Chaos Communications Congress discusses tools to interact with an undocumented embeded system [61] [62]. They present an Arduino based JTAG Pin scanner and instruction probe.

2.3.4 Cisco flash file systems

Cisco uses use, at least, three different flash file systems, referred to a Class A, B, and C [63]. Linear (unmanaged) flash storage is mounted with a “Slot[x]:” device designator, while ATA disks are mounted as “disk[x]:”.Table 2-3 summarizes the different properties of these three file systems.

[*] http://events.ccc.de/congress/2010/Fahrplan/events/4011.en.html

Table 2-3: Cisco's Class A, B, and C flash file systems [63]

Class A	*Delete* command simply marks the file as deleted. *Undelete* recovers files marked for deletion. *Squeeze* command permanently erases files marked for deletion by rewriting the whole flash. *Format* command erases all files.
Class B	*Delete* command simply marks the file as deleted. *Erase* command erases all flash *Partition* command splits the flash into several file systems
Class C	*Delete* command simply marks the file as deleted. *Squeeze* command permanently erases files marked for deletion by rewriting the whole flash. *Format* command erases all files.

2.3.5 Cisco boot sequence and configuration

There is large variety of Cisco network devices, but a very common boot sequence is as follows. A small bootstrap software, called Rommon, starts. Its primary job is to look in the flash for a file to boot. The boot process can be interrupted via the device's terminal interface by sending an RS-232 break signal (for routers) or pushing the *mode* button on the front of switches, after which the user ends up in a *preboot* state. In this state system environmental variables can be set to control how and where to boot the IOS file and whether the startup configuration should be loaded or not during boot. The startup configuration is stored in nonvolatile memory and normally loaded into RAM during boot and called the "running configuration". Configurations stored in RAM are called the running-config and have to be explicitly copied to nonvolatile storage (often referred to as NVRAM in Cisco's documentation) in order to survive a reboot.

Once the system has booted a user can login. Now the user is presented with a Command Line Interface (CLI) with limited functionality called the "User EXEC mode". After entering the command "enable" and possibly a password (if an enable password is set) the user enters the "Privileged EXEC Mode" which gives access to all system commands, including making configuration changes.[64]

2.3.6 Cryptographic Erase

If all data on a storage device is encrypted and the encryption key is erased, then the plain text data is unrecoverable and thus can be considered to be erased. Therefore, as long as the encryption method is strong and the key cannot be recovered from the device, then simply erasing the key effectively and efficiently makes the data on the storage device inaccessible.

The encryption/decryption mechanism can be implemented in the host OS or in the device's own controller. Today's SATA drives often implement the Trusted Computing Group's OPAL and Enterprise standards. This enables the host to issue a Cryptographic Disk Erasure command that erases the key from the drive by generating a new key and overwriting the previous key and setting the drive into the "new drive" state [65].

A benefit of cryptographic erase is its high speed. Overwriting a hard disk drive (HDD) containing many terabytes of data with several different patterns can take hours. In comparison, erasing the crypto key is very quick (i.e., of the order of a few milliseconds). However, a problem remains: Can we be certain that the cryptographic key is permanently erased and that it not possible to retrieve this key from the disk controller's RAM or CPU registers? This problem is addressed in the paper "SAFE: Fast, Verifiable Sanitization for SSDs" [66], where Swanson and Wei propose a solution where the crypto key erasure is followed by a new unencrypted state in which the storage media itself can be overwritten and verified.

3 Research methods

How could we determine if a device still has sensitive data in its nonvolatile memories after a configuration is erased? Data in each memory can be inspected using the tools described in Chapter 2. By inspection, we can see if the nonvolatile memory contains some specific strings or patterns that might occur in a configuration. If we find a string “pass” in 256 MB of storage can we conclude that this string is from the configuration or could it appear randomly? This chapter:

1. Proposes a method to generate good markers to be used as parameters for such an inspection.
2. Shows how to calculate the probability of an accidental match in the data being searched. The chapter defines the concept of “marker strength” as the complement of this probability. Appendix I further discusses the strength of a marker.

The procedure described in Table 3-1 will be used to investigate if sensitive data can be recovered after an erase procedure is performed on a device. The research methodology used is *qualitative* as each test will deliver an outcome which is either “*unsafe*” or “*unknown*”. There is no numerical data to process for a *quantitative* analysis.

Table 3-1: Procedure for testing erase procedure

```
FOR EACH {DEVICE, PROPOSED_ERASE_PROCEDURE}
    1. Generate a set of random data to be used as markers.
    2. Configure some sensitive data on the DEVICE using each marker (only
       once). This data includes: SNMP passwords, Wi-Fi keys, passwords, etc.
    3. Save the configuration.
    4. Erase the configuration using the specified PROPOSED_ERASE_PROCEDURE
    5. Power off.
    6. Perform memory recovery and marker search
    7. If any of the markers can be found somewhere in the retrieved data, this
       particular PROPOSED_ERASE_PROCEDURE on the DEVICE can be declared unsafe.
       At least within the certainty of the marker strength.
```

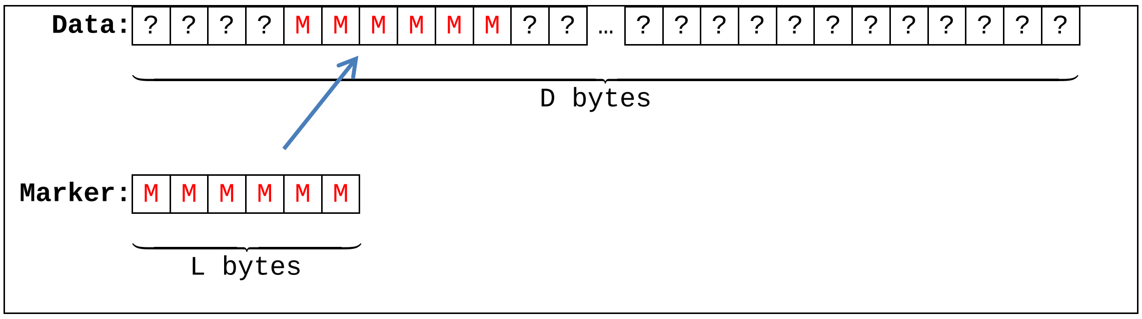
3.1 Device platform and erase procedure to be tested

In Table 3-1 a DEVICE is a particular combination of hardware and software, while an ERASE_PROCEDURE is the procedure by which the configuration is to be erased. This procedure may include CLI command(s) according to manufacturer’s instructions, toggling a dual in-line package (DIP) switch, pushing a reset button, removing the NVRAM battery, etc.

3.2 Marker generation and the risk for a false positive

Each marker must be constructed so that it complies with the configuration parameter’s format. For example, an alphanumeric string would be suitable for passwords, an IPV4 address would be 4 bytes, an Ethernet MAC address 6 bytes, etc. It is important to construct sufficiently long and unique markers that the chance of finding them by accident in a large string of data is negligible. That is we want to minimize the risk of false positives.

Figure 3-1: Data and marker sequences



In this analysis of marker strength I will assume that random marker of a single byte has the probability 2^{-8} to match any other byte (for further detailed discussion of this simplifying assumption see Appendix I).

A marker with length of L bytes can overlay data of D bytes in D-L+1 ways. Or differently said there are D-L+1 substrings of length L in a string of length D. We can generalize this to compute the probability of finding a random marker of length L bytes by accident in a data string of length D as the complementary probability of not finding it at any of the possible overlays/substrings within the file:

$$p = 1 - \left(1 - \frac{1}{256^L}\right)^{D-L+1}$$

In our case D is *much* larger than L, hence the above probability can be approximated as:

$$p = 1 - \left(1 - \frac{1}{256^L}\right)^{D}$$

The value $\left(1 - \frac{1}{256^L}\right)$ will be very close to 1 for large L, and the probability p above will be difficult to calculate with computer software such as Excel because of the limited precision in the floating point number representations [67]. As an example, Excel calculates this probability as exactly 1 for L≥7. Fortunately, the value of this expression can be approximated using the definition for the mathematical constant e [68p. 131] as:

$$p = 1 - \left(1 - \frac{1}{n^L}\right)^{D} = 1 - \left(1 - \frac{1}{n^L}\right)^{n^L \frac{D}{L}} \approx \left\{ For\ big\ n^L\ we\ can\ approximate \left(1 - \frac{1}{n^L}\right)^{n^L} \approx e^{-n^L} \right\} \approx$$
$$= 1 - e^{-\frac{D}{n^L}}$$

Table 3-2: Marker probability examples

Example	Marker length [bytes]	Data to search [bytes]	Probability of accidental match
IPv4 address in one megabyte	4	1MB = 2^{20}	0.024%
IPv4 address in one gigabyte	4	1GB =2^{30}	22%
MAC address in one gigabyte	6	1GB =2^{30}	0.00038%
MAC address in one terabyte	6	1GB =2^{40}	0.39%

From the above we can see that the probability of finding any given IPv4 address in a gigabyte of random data is over 20%. The devices this thesis aim to investigate could potentially have 1 GB of data to search. As such, IPv4 markers should be avoided if possible. Or, if used, then the result should at least have a note explaining the probability of an accidental match. A random MAC address marker on the other hand is very unlikely to be found in 1 GB of data, hence it would be a good strength marker.

To avoid problems with unsupported characters in passwords, only alphabetic characters in the range [a-z] and [A-Z] will be used. If these strings are stored as one character per byte, there are 52 combinations per every byte position. If the data searched contains an "equivalent alphabet" of character as the marker characters, then the probability of an accidental match is:

Equation 3-1

$$p = 1 - \left(1 - \frac{1}{52^L}\right)^{D-L+1} \approx 1 - e^{-\frac{D}{52^L}}$$

With "equivalent alphabet" is meant the marker and the data share the exact same characters. E.g. if the marker is the 52 characters in the set [a-z] and [A-Z] then the data must not contain any other symbol for Equation 3-1 to hold.

But what if the marker and data do not share the same alphabet? E.g. the marker may be composed of the 52 character set but the data could be any byte value 0x00 to 0xff? In Appendix I it will be shown that Equation 3-1 still holds as an *upper limit* of an accidental match between a random marker and *any* data.

Table 3-3: Probability of finding random character strings in a random text

Example	Marker length [chars]	Text to search [bytes]	Maximum probabilityof accidental match
4 chars in one kilobyte	4	10^3	1.4×10^{-6}
8 char string in one megabyte	8	10^6	1.9×10^{-6}
8 char string in one gigabyte	8	10^9	1.8×10^{-5}
8 char string in one terabyte	8	10^{12}	1.9×10^{-2}
10 char string in one terabyte	10	10^{12}	6.9×10^{-6}

Conclusion: 10 character markers will be sufficiently long enough to rule out an accidental collision in the amount data we will be looking at, when looking for passwords, SNMP community strings, etc.

We define the quantity *marker strength* to be the complementary of the maximum probability to find the marker by accident. For example, the marker strength of the marker "8 char random string in one terabyte" would be 1-0.019 = 0.981 = 98.1% with reference to Table 3-3.

Based upon my study of white noise versus contrasting markers (See Appendix I, Appendix J, and Appendix K), I conclude that the benefits of using contrasting markers are outweighed by the simplicity of the white noise marker. Hence, I will use white noise markers for my investigations and in the computations of marker strength. To make searching for markers easier for humans (and to facilitate the examples in this thesis) each string marker will be prefixed by the string "MARK". The marker strength increases due to the addition of these 4 extra characters, but since they are non-random, we cannot calculate the extra strength it adds without knowledge of the character probability distribution of the data the marker is placed in. Therefore, the marker prefix will not be included when calculating and evaluating the strength. Appendix B contains the Excel function used to generate the string markers .

3.3 Configuration and marker injection

When testing a device some selected sensitive parameters of the device will be set using the method that it is normally used for configuring the device, e.g., through a console terminal or web interface.

To prevent marker contamination between tests, each marker will only be used once for a given combination of device, erase-test, and parameter. The marked configuration will be saved to nonvolatile storage following the vendor's standard platform specific procedure. If configuration backup or archiving is available as a device feature or common technician's practice, then different sets of markers will be used for the current and backup/archive configuration so that we can distinguish which of the configuration sets a marker originated from when a marker is found.

3.4 Configuration erasure

The erasure part of the test will try each of the different methods of erasure offered by the device. Each erase attempt will be followed by the recovery step to see if the erasure was successful or not. The primarily erase procedures that will be tested are those described in the device vendor's recommendations. If all of these procedures prove to be unreliable, then an improved erasure method will be suggested and tested using the same procedure.

A single erasure method may involve several steps and sub methods. Typically, there is a power-off step as part of the erase procedure. There may be a difference between whether the device is powered off by physically removing all sources of power and when the device is simply powered off with a command. A "power-off" command gives the device an opportunity to store some configuration data from volatile RAM into non-volatile storage, where it can be retrieved later. For this reason, it will be important to ensure that both methods of powering off the device are tested, thus a computer controlled power strip would be a useful tool to use during this testing.

3.5 Memory recovery and marker search

This step aims to recover as much non-volatile memory data as possible, and then the recovered data is searched for the previously set markers. Several methods of recovering data will be used, especially those discussed in Chapter 2.

The markers will be searched for in the data and when a marker is found we can conclude that the erase procedure used on this device does not erase this item of sensitive information and thus the erase procedure is unsafe. Note that it is relatively easy to determine if a method is unsafe. We simply need to prove a single exception. However, proving the opposite is very hard; hence, if the markers are not found we can only be certain of one thing: the erase procedure of this device is safe for the specific parameters tested while using the tested recovery procedures. There will always be some probability there exists or will exist a method able to recover sensitive data.

Unfortunately (for our testing), many devices perform a hash before storing passwords –so that the plain text of the password *cannot* be recovered, but the password can be verified by matching the hash with a stored hash. For this reason, we may need to use the CLI or other interface to learn the hashed form of passwords – so that we can search for these strings. Note that some equipment has defined procedures for recovering passwords, see for example the note "Note: Unlike other Cisco platforms, the Aironet hardware and software do not allow password recovery. You must instead return the equipment to its default state, from which it can be reconfigured." in Cisco's Aironet 1200 Series AP document "Password Recovery Procedure for Cisco Aironet Equipment" [69].

4 Investigation of sanitization completeness

This chapter investigates the sanitization of a Cisco router and 3 HP Procurve switches. Memory investigation methods discussed in Chapter 2 are used as well as the marker injection technique from Chapter 3.

4.1 Sanitization of the Cisco 1712 router

This section will investigate the CISCO 1712 router. The full part number of the specific device that has been examined is CISCO1712-VPN/K9. First, we will examine the hardware to locate its nonvolatile memories. Next we try to apply methods from Chapter 3 to read and write to each of them. Finally, we will check how well the vendor recommended methods erase the sensitive information.

4.1.1 Router overview and exterior interfaces

Cisco's describes the CISCO 1712 router as [70]:

"The Cisco 1712 Security Access Router offers an all-in-one security, routing, and switching solution for enterprise small branch offices and small and medium sized businesses; with built-in Fast Ethernet LAN switching, Fast Ethernet port for DSL or broadband modem connectivity, integrated IOS Security and backup WAN for link redundancy to help ensure high availability of critical business applications."

All of the router's ports are located on the back of the device as shown in Figure 4-1. These connectors are:

- 10/100 Ethernet port
- Console port for configuration (asynchronous RS232)
- AUX port
- ISDN Interface (right card)
- 4 port Ethernet switch card (left card)
- Power connector (12V, -12V, and 5V DC inputs)

Both the Ethernet and ISDN card have been riveted into the chassis. They are actually standard WIC-4ESW and WIC-1B-S/T modular cards, but Cisco has riveted them into place to make a fixed system model for the CISCO 1700 router series.

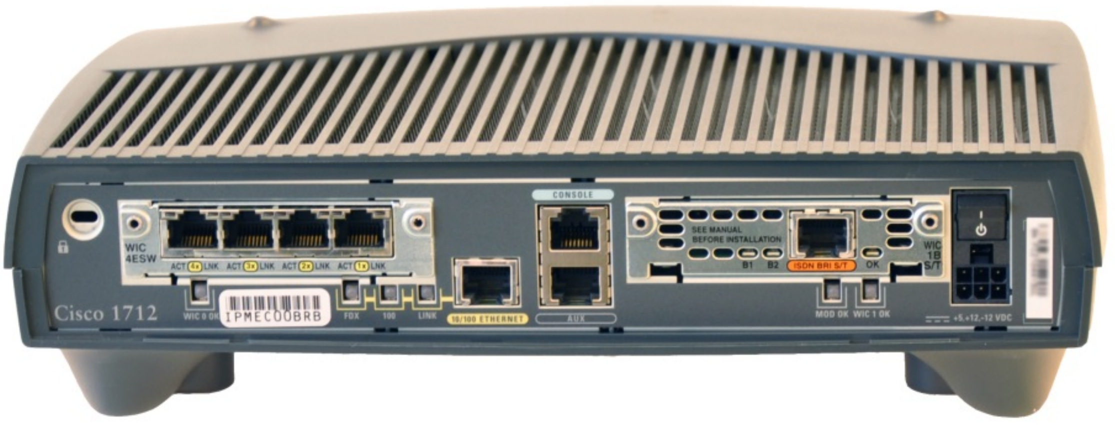


Figure 4-1: CISCO1712 router connector side

Inside there is another installed card: a MOD1700-VPN VPN encryption module used to offload the main CPU when handling encrypted traffic [71].

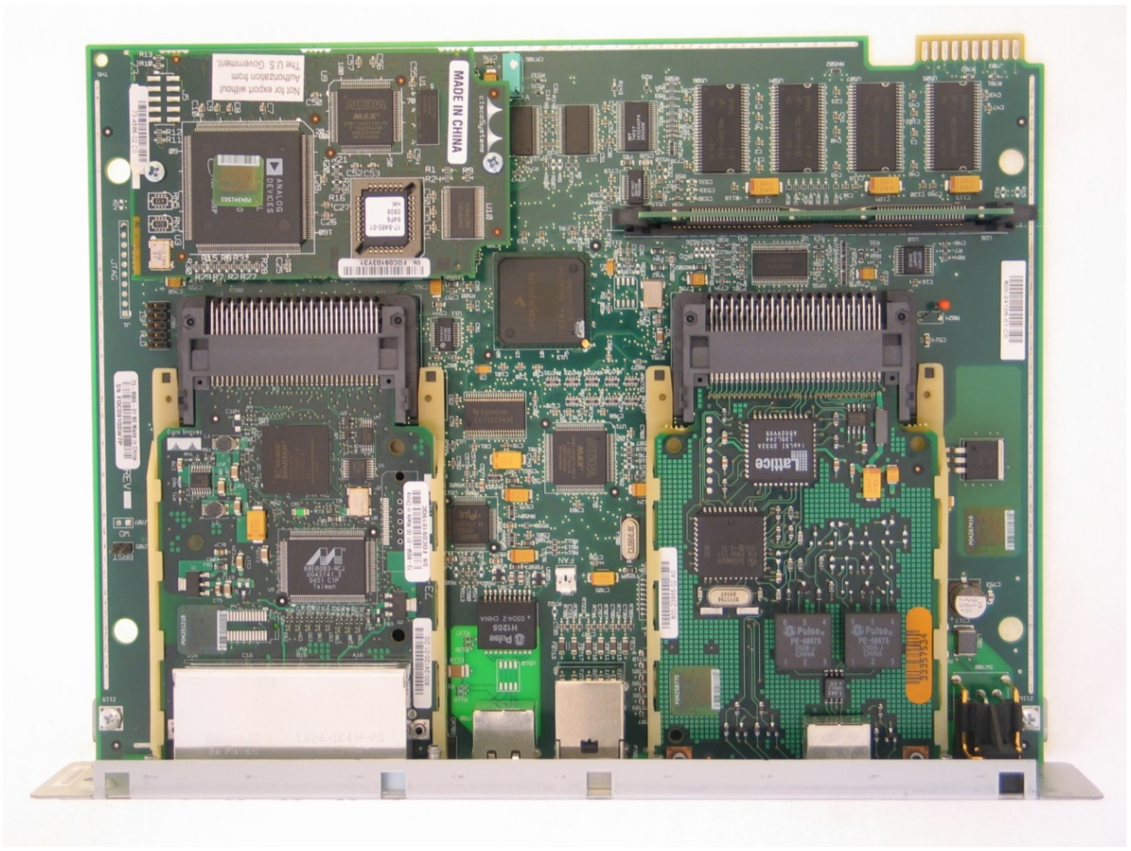


Figure 4-2: CISCO1712 inside view – with a MOD1700-VPN VPN encryption module in upper left-hand corner and with the 4 port Ethernet (lower left slot) ISDN (lower right slot) and modules in place.

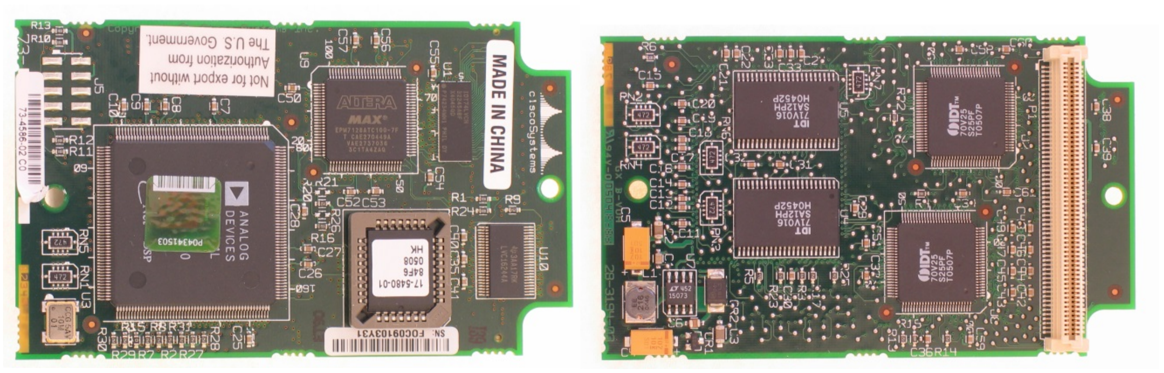


Figure 4-3: MOD1700-VPN VPN card removed. Top view (left) and bottom view (right)

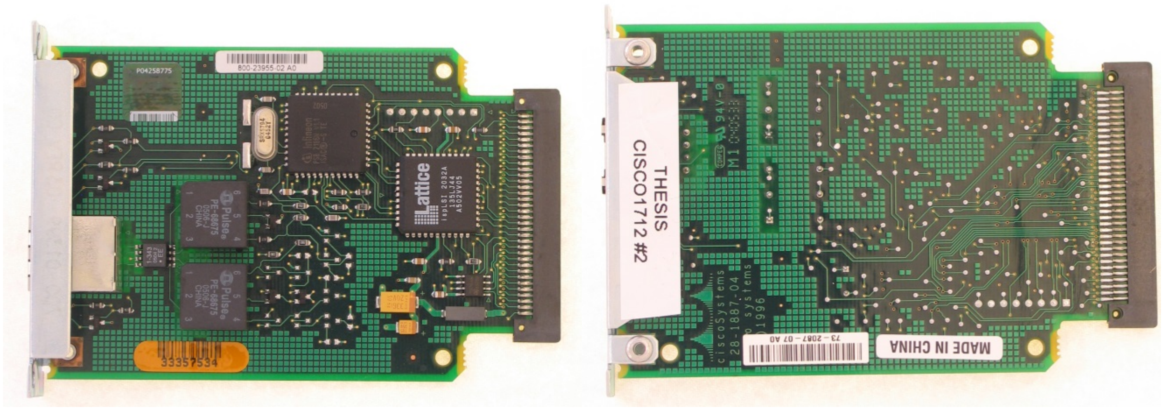


Figure 4-4: WIC-1B-S/T ISDN card removed. Top view (left) and bottom view (right)

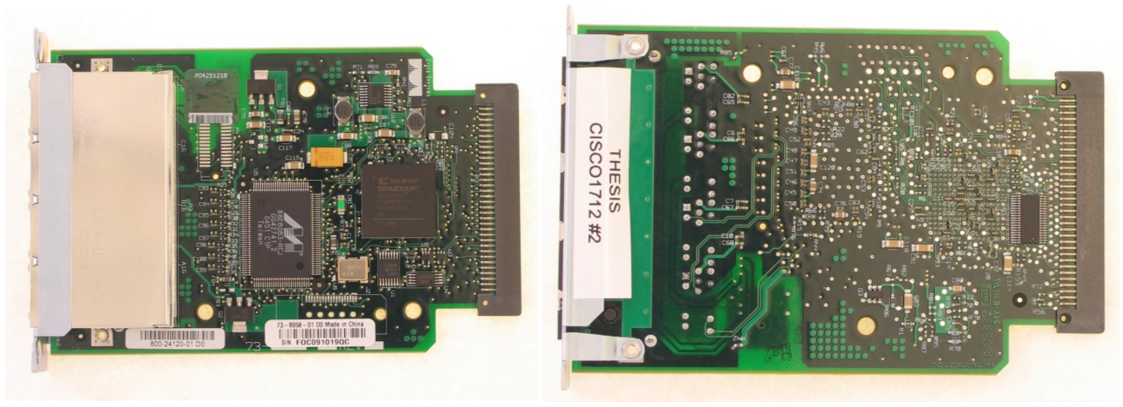


Figure 4-5: WIC-4ESW Ethernet switch card removed. Top view (left) and bottom view (right)

Figure 4-6 shows the logic board with the interface cards and the crypto card removed (the interface cards' rivets were drilled away). There are no components on the underside of this main logic board. The numbers in the figure refer to the components listed in Table 4-1.

Table 4-1: CISCO1712 Interesting objects on main logic board

Object Number	Description
1	2 x 4 header solder point marked J3, CODE TAP
2	Two Intel E28F128 J3A150 16 MB flash TSOP56 memories marked U502 and U503
3	1 x 10 header solder point marked J1, JTAG
4	2 x 5 header marked J701, ISP PLD
5	CAT 28C256-12 32KB PLCC32 EEPROM marked U16
6	2 point jumper solder point marked J707, WD
7	2 point jumper marked J705, BRST
8	PLCC32 socket labeled U501 holding a 512 KB ST M27C4001 EPROM labeled “17-23 , 58-02, CS=245E ROMMON, 0507 HK”
9	Freescale XPC862PZ100B
10	RAM, 64MB soldered and 32MB expansion module
11	20 pin (10 on each side) board edge connector.

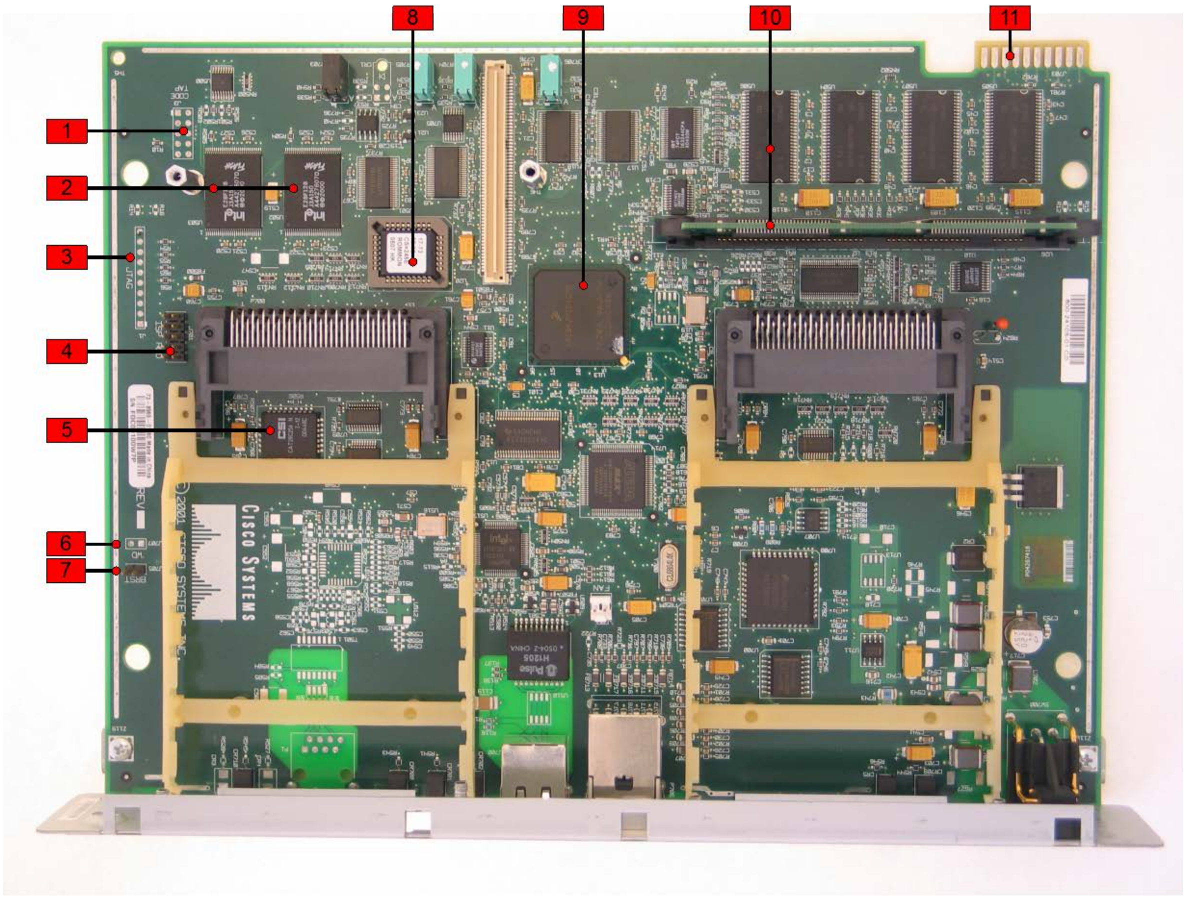


Figure 4-6: View of main PCB with modules removed

The “show version” command in Output listing 4-1 provides some useful information with reference to the objects of interests in Figure 4-6 (as numbered in Table 4-1):

- ROM Bootstrap corresponding to the 512 KB EPROM (object 8).
- CPU is a MPC862P (object 9)
- 32 KB of NVRAM. Corresponding to the 32 KB EEPROM (object 5)
- 32 MB of flash corresponding to the two 16 MB flash chips (object 2)
- 64 MB RAM + 32 MB RAM expansion (object 10). The "show version" command lists the RAM divided into two parts "**86598K/11706K bytes of memory**", where 86598KB is RAM used to execute the IOS and 11706KB is I/O Memory dedicated to packet traffic [72]. For the CISCO 1712 this is just a logical division performed upon boot and both regions share the same physical RAM chips and the total amount of RAM is the sum, 96MB.

Output listing 4-1: Output of show version command on CISCO 1712

```
Router#show ver
Cisco IOS Software, C1700 Software (C1700-K9O3SY7-M), Version 12.3(11)T9, RELEASE SOFTWARE
(fc3)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2005 by Cisco Systems, Inc.
Compiled Tue 13-Dec-05 05:20 by ccai

ROM: System Bootstrap, Version 12.2(7r)XM4, RELEASE SOFTWARE (fc1)

Router uptime is 1 minute
System returned to ROM by power-on
System image file is "flash:c1700-k9o3sy7-mz.123-11.T9.bin"

<snip>

Cisco 1712 (MPC862P) processor (revision 0x101) with 86598K/11706K bytes of memory.
Processor board ID FOC09100W7P (2205170844), with hardware revision 0000
MPC862P processor: part number 7, mask 0
1 Ethernet interface
5 FastEthernet interfaces
1 Virtual Private Network (VPN) Module
32K bytes of NVRAM.
32768K bytes of processor board System flash (Read/Write)

Configuration register is 0x2102
```

4.1.2 Expansion cards: VPN card, ISDN and Ethernet switch

The MOD1700-VPN daughter card main component is an ADSP-2131L encryption coprocessor from Analog Devices. An Atmel AT28LV010 128KBEEPROM is in the PLCC32 socket. One can notice the interesting combination of labels: "Not for export without Authorization from the U.S Government" and "MADE IN CHINA". The 2 x 5 solder pads were tested with the JTAGulator. It exposes a TAP interface to the Altera MAX programmable logic device. The VPN card clearly has EEPROM data such as the one seen the "show diagnostic command" in Output listing 4-1. However, I believe that the information stored is factory written data, such as firmware and serial numbers, and thus these memories will not be part of my research when investigating the completeness of the sanitization. The same assumption is made for the ISDN and 4 port Ethernet switch cards.

If the consistency of this data should be investigated, then we could connect a logic analyzer to the memory control signals (such as WE going to the EEPROM) to ensure it is never written to during normal operation. However, this is left for future work.

Output listing 4-1 "show diagnostic" command on CISCO1712

```
<snip>
Slot 3:
        Virtual Private Network (VPN) Module Port adapter, 1 port
        Port adapter is analyzed
        Port adapter insertion time unknown
        EEPROM contents at hardware discovery:
        Hardware Revision        : 2.1
        Part Number              : 73-4586-02
        Board Revision           : C0
        Deviation Number         : 0-0
        Fab Version              : 03
        PCB Serial Number        : FOC09103Y31
        RMA Test History         : 00
        RMA Number               : 0-0-0-0
        RMA History              : 00
        Product (FRU) Number     : MOD1700-VPN=
        EEPROM format version 4
        EEPROM contents (hex):
          0x00: 04 FF 40 01 79 41 02 01 82 49 11 EA 02 42 43 30
          0x10: 80 00 00 00 00 02 03 C1 8B 46 4F 43 30 39 31 30
          0x20: 33 59 33 31 03 00 81 00 00 00 00 04 00 FF FF FF
          0x30: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0x40: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0x50: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0x60: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
          0x70: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
<snip>
```

4.1.3 ROM Monitor (Rommon) memory inspection

This router has at least two non-volatile memories: an EEPROM (object 5) and flash (object 2). In the following paragraphs, we will describe how we write a marker to each of these two non-volatile memories from the IOS CLI, reboot and try to find the marker by inspecting the memories using Rommon.

4.1.3.1 Marker injection in NVRAM and flash

SNMP passwords are stored in the startup-configuration residing in NVRAM while the VTP password is inside the *vlan.dat* file in flash. Table 4-2 shows the two markers that were used. These markers were written to the indicated memories using the IOS CLI, then verified with CLI commands to be stored in their respective non-volatile memories. The console log in Output listing 6-1 shows the details of how these markers were written and verified.

Table 4-2: First two markers

Parameter to host marker	String marker (10 random chars)	Parameter storage	Storage size	Marker Strength
SNMP password	MARKWHKMcflpXC	NVRAM (startup-config)	32KB	1 - 2*10^{-13}
VTP Password	MARKlscAlvXimn	flash:/vlan.dat	32MB	1 - 2*10^{-10}

4.1.3.2 Rommon Priv mode

In order to enter the privileged (priv) mode of Rommon, the router is power cycled and Rommon mode is entered by sending a break signal prior to the IOS booting. The 5 first words (double bytes) from the "cookie" command are added together and the least significant word is used as the password to access the hidden priv mode.

Output listing 4-2: Acquiring PRIV mode access

```
System Bootstrap, Version 12.2(7r)XM4, RELEASE SOFTWARE (fc1)
TAC Support: http://www.cisco.com/tac
Copyright (c) 2003 by cisco Systems, Inc.
C1700 platform with 98304 Kbytes of main memory



monitor: command "boot" aborted due to user interrupt
rommon 1 > cookie

cookie:
01 01 00 13 7f 3b df 68 33 00 01 ff 04 48 00 11
00 00 00 00 00 00 00 00 46 4f 43 09 10 30 57 37
50 01 01 00 00 00 00 00 00 ff ff ff 58 04 49 23
09 01 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
rommon 2 > priv
Password:
You now have access to the full set of monitor commands.
Warning: some commands will allow you to destroy your
configuration and/or system images and could render
the machine unbootable.
```

The Rommon is commonly used to change the boot method and to recover IOS if the flash has been erased. Those commands are well documented. But the Priv mode in rommon is undocumented to end users. (as far as I know), so making use if it requires some research. The Rommon priv mode on this router offers interesting commands to inspect and change memory. Table 4-3 shows some examples of priv model commands for the CISCO 1712, while Output listing 6-2 in Appendix C lists all of these commands.

Table 4-3: Examples of priv mode commands for CISCO 1712

Command	Function
dump <address> <length>	Displays memory contents to the console (hex dump style)
cpu	Displays CPU type and version
flash info	Shows flash information, such as base memory address

4.1.3.3 The Reading from NVRAM and investigating its structure

The NVRAM is a CAT 28C256 32 KB parallel EEPROM. It has a 14 bit address input and 8 bit data I/O. Because it is directly addressable, it is likely that this chip will be mapped into some region of the CPU's address space. The flash info command shows the base address of the flash as 0x60000000, but the meminfo command does not show the NVRAM base memory address (as it does on the Supervisor720 card [44]).

Output listing: 4-3: CISCO1712 Priv mode flash info and meminfo commands

```
rommon 12 > flash info

System flash info

flash driver info structure # 0
flash base: 0x60000000    flash width: 2
flash set size: 0x2000000    num banks: 2    device: INTEL 28F128J3A

total flash is 0x2000000

rommon 13 > meminfo

Main memory size: 96 MB.
Available main memory starts at 0x10000, size 98240KB
IO (packet) memory size: 10 percent of main memory.
NVRAM size: 32KB
rommon 14 >
```

According to the "Internetworking Troubleshooting Handbook" [73Fig. Table B–31] the CISCO1720 has "NVRAM, WIC, internal, regs" at base address to 0x68000000. Since that is a similar platform we will check what we can find at the same address using the dump command. The first part of the dump is listed in Output listing 6-3 in Appendix C and it looks promising. Table 4-4 shows the first 0xBF0 of the contents of the NVRAM, with a description of their likely purpose.

Table 4-4: NVRAM contents - some highlights

Address offset in (hex) relative to base address 0x68000000	Length (bytes, decimal)	Description
0	1	Unknown (maybe some checksum or magic number)
1	128	Cookie, possibly followed by a two byte checksum
90	2	Config-register
91	6	Magic values “defd feed face”
148	275	Rommon environmental variables
658	216	Text with information about the software
75A	128	The cookie (again)
82C	909	Startup-config including the injected SNMP marker MARKWHKMcflpXC

The cookie data (highlighted in Output listing 4-4) starts at byte offset 1 and contains the values shown in

Table 4-5. A description of the cookie fields can be obtained by issuing the cookie command in Rommon priv mode, see Output listing 6-4 in Appendix C. There are also a references on the web, such as [74].

Output listing 4-4: Initial portion of the NVRAM (highlighting the cookie)

```
rommon 16 > dump 0x68000000 0xfff
68000000  8d74 0101 0013 7f3b df68 3300 01ff 0448
68000010  0011 0000 0000 0000 0000 464f 4309 1030
68000020  5737 5001 0100 0000 0000 00ff ffff 5804
68000030  4923 0901 ffff ffff ffff ffff ffff ffff
68000040  ffff ffff ffff ffff ffff ffff ffff ffff
68000050  ffff ffff ffff ffff ffff ffff ffff ffff
68000060  ffff ffff ffff ffff ffff ffff ffff ffff
68000070  ffff ffff ffff ffff ffff ffff ffff ffff
68000080  ffff 1342 ffff ffff 0000 0000 0000 0000
68000090  2102 defd feed face 0000 0000 0000 000a
…
```

Table 4-5: Cookie contents

Cookie offset (hex)	**Value**	**Description**
0	0x01	vendor
1	0x01	version
2-7	00:13:7f:3b:df:68	MAC address
8	0x33	processor
9	00	NVRAM size code (0x00 stands for 32K)
a	01	CPU speed code (0x01 stands for 50 MH)z
b	FF	unused
c-d	0448	Board PM ID
e-f	0011	number of allocated MAC addresses (0x0011 - this device was allocated 17 MAC addresses)
10-17	00 00 00 00 00 00 00 00	8 bytes of zeros
0x18-0x22	0x464f, 0x4309,0x1030, 0x5737, 0x5001, and 0x01	board serial number (0x464f, 0x4309,0x1030, 0x5737, 0x5001, and 0x01 which corresponds to “FOC09100W7P” followed by 0x0101) – this value was shown as the processor board ID in Output listing 4-1, and in the barcode label on the board (Figure 1-1Figure 4-6)
23-24	00 00	deviation.
0x25-0x2c	00 00 00 00 ff ff ff 58	
2d	0x	Board configuration
2e-37	49 23 09 01 ff ff ff ff ff ff	

After 32KB, at address 0x68008000 the same NVRAM data repeats .This appears to be an exact copy of the NVRAM contents. As the chip is only 32 KB it can only contain this amount of data, but it seems that the chip select logic inside the CPU enables the chip for two different base addresses 0x68000000 and 0x68008000 – thus reading from the second set of addresses reads the same memory cells again.

4.1.3.4 Writing to NVRAM from Rommon priv mode

The first part of NVRAM, as seen in Output listing 6-7 in Appendix C, contains vital data structures, such as the device's primary Ethernet's MAC address. We must not overwrite these values, but address 0x68000660 contains some text string coming from the IOS[*], so that seems safe to try and write to. According to the CAT28C256 datasheet [75] writing to the EEPROM NVRAM is simple. A single byte can be written and the old data is automatically erased. However, when I tried using the *alter* and *fill* commands each appeared to happily execute, but no data was changed. The datasheet mentions a feature called "Software Data Protection". The chip can be put in a special write protected mode so that all writes must be prepended by a 3 byte magic sequence. There is also another special magic sequence to turn off "Software Data Protection" completely. That special write protect deactivation sequence was written to the chip, but still the memory could not be written.

To understand why these writes fail, I listened to the CPU's signals to the chip. A Pomona PLCC32 test clip was clamped around the EEPROM (Figure 4-7) and the signals in Table 4-6 were connected to the digital inputs of a Cleverscope CS328A digital oscilloscope.

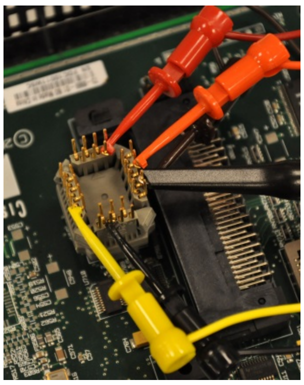


Figure 4-7: EEPROM PLCC32 Testclip on the NVRAM of CISCO1712

Table 4-6: Measured EEPROM pins

PLCC32 pin	Short name	Name
23	$\overline{CE}$	Chip Enable Inverse (This signal was also used as trigger.)
31	$\overline{WE}$	Write Enable Inverse
25	$\overline{OE}$	Output Enable Inverse
13	IO_0	I/O bit 0 (least significant bit)

According to the datasheet, data is written when $\overline{OE}$ is high and when both $\overline{WE}$ and $\overline{CE}$ go low.

Figure 4-8 shows a snapshot of the signals when NVRAM is written during execution of the built in Rommon priv command " menu, nvram test". $\overline{WE}$ and $\overline{CE}$ both correctly go low. On the other hand, Figure 4-9 shows what goes wrong when we try to write a byte 0xA1 with the command "fill -b 0x68000660 0x1 0xa1". We can see a bit coming in on I/O pin, but the CPU never pulls $\overline{WE}$ down, therefore nothing is written. I also tried to write to the address 0x68008660 without success.

[*] The start-config srcript actually starts at offset 0x82C, so I should have used this address rather than an offset of 0x660. However, I used an offset of 0x660 for my testing.

Figure 4-8: EEPROM Control signals during Rommon priv NVRAM test

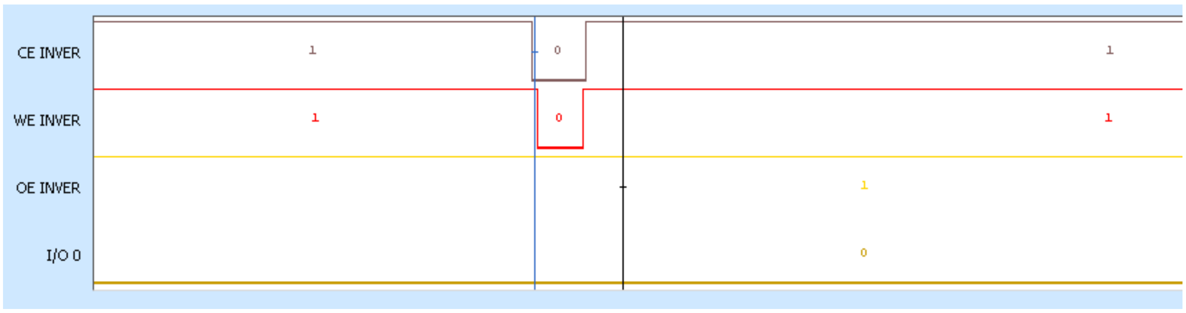
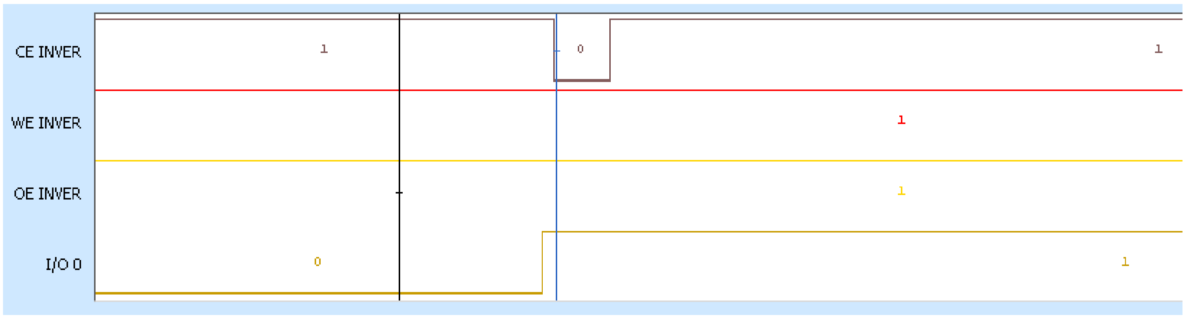


Figure 4-9: EEPROM Control signals during Rommon priv fill command



4.1.3.5 *The MPC862 internal memory controller logic*

To investigate why the $\overline{WE}$ signal is never pulled low, I did some research on the memory controller logic inside the MPC862.Its operation is explained in Chapter 15 of this processor's reference manual [76].

The CPU has 8 configurable "banks" of memory numbered 0 to 7. The memory region to control is defined for each bank by a 17 bit Base Address and a Mask. The Base Address is the most significant portion of the address on the memory bus. Each bank has configurable properties, such as data width, write protect, and whether parity is used or not. For each bank there is also a "machine" selection property which determines the control signal logic to interface to different types of memory. For example, reading and writing to RAM may need different timing and control signals than to an EEPROM.

The banks' configuration is stored in a special memory block inside the CPU holding various settings. The location of this memory block in memory space is determined by a register called "Internal Memory Map Register" (IMMR). This register is initialized at CPU startup. In the CISCO1712 IMMR is to 0xff000000. The IMMR register and memory controller settings can be found by the Rommon priv menu choice "show 862 registers". The output of this command is shown in Output listing 6-5 in Appendix C. An (almost) working Java program (see Output listing 6-13 in Appendix G) was written to decode the memory control registers and emulate the memory block selection mechanism for an address. The program's output is shown in Output listing 6-6 in Appendix C. I guess that bank#0 is for an 8 bit datapath BOOT ROM. Bank#6 is for flash and is write protected. Bank#7 fits the EEPROM address and the 8 bit width is just what would be expected for the EEPROM chip. We note that this bank is **not** write protected.

A continuity trace from the EEPROM's $\overline{WE}$ pin reveals it is not directly connected to the CPU. It terminates in at pin 55 on an Altera EPM7128 in the center of the logic board. This a programmable device [77] and pin 55 is a general I/O pin. The Altera chip has a security feature so the contents may not be possible to read out. At this point I think the easiest next step to understand how to activate the $\overline{\text{WE}}$ line properly is to disassemble the rommon or IOS code and search for parts where the EEPROM NVRAM is programmed. Such as during a *config-register 0Xxxxx* command. However, this work will be relegated to the future work section of the thesis, because I am not very experienced with reverse engineering machine code and there might also be legal complications in doing so. Later on in Section 4.1.8 we will see that the device actually sanitizes the contents of the EEPROM sufficiently using the built in routines, thus direct memory write access to this chip is not crucial.

4.1.3.6 Reading from flash and investigating its file system

The flash chips are two Intel E28F128 J3A150 16MB NOR flash chips. According to the datasheet they can be read using direct parallel memory access [78], so it is likely they are mapped into the address space. The flash info command shows the base address of the flash as 0x60000000 and the size to 32MB.

Output listing 4-5: CISCO1712 Priv mode flash info and meminfo commands

```
rommon 12 > flash info

System flash info

flash driver info structure # 0
flash base: 0x60000000    flash width: 2
flash set size: 0x2000000    num banks: 2    device: INTEL 28F128J3A

total flash is 0x2000000

rommon 13 >
```

We can begin to inspect the flash using the “dir” flash comment. The dir flash command lists two active files and 4 deleted files (as shown in Output listing 4-6)

Output listing 4-6: CISCO1712 Rommon PRIV mode dir flash

```

rommon 52 > dir flash:
         File size           Checksum   File name
  13296264 bytes (0xcae288)   0x2384    c1700-k9o3sy7-mz.123-11.T9.bin
       600 bytes (0x258)      0xba86    vlan.dat (deleted)
       600 bytes (0x258)      0x2c42    vlan.dat (deleted)
       600 bytes (0x258)      0x7cf8    vlan.dat (deleted)
       600 bytes (0x258)      0x94e9    vlan.dat (deleted)
       600 bytes (0x258)      0x759c    vlan.dat
rommon 53 >
```

The first 128 bytes memory following the flash base address are shown in Output listing 4-7. The first 16 bit word 0xbad0 0x0b1e seems crafted and is probably magic number. We can see the file size (0x00cae288), checksum (0x2384), and the name of the file “c1700-k9o3sy7-mz.123-11.T9.bin”. The contents of the executable file starts at 0x60000040 (set as red text in the table).

Output listing 4-7: CISCO1712 Priv mode flash memory dump

```
rommon 51 > dump  0x60000000 0x80
60000000  bad0 0b1e 00ca e288 2384 ffff 0000 0000 ........#.......
60000010  6331 3730 302d 6b39 6f33 7379 372d 6d7a c1700-k9o3sy7-mz
60000020  2e31 3233 2d31 312e 5439 2e62 696e 0000 .123-11.T9.bin..
60000030  0000 0000 0000 0000 0000 0000 0000 0000 ................
60000040  7f45 4c46 0102 0100 0000 0000 0000 0000 .ELF............
60000050  0002 0033 0000 0001 8000 8000 0000 0034 ...3...........4
60000060  0000 0054 0000 0000 0034 0020 0001 0028 ...T.....4. ...(
60000070  0006 0000 0000 0001 0000 0144 8000 8000 ...........D....
rommon 52 >
```

If we inspect the memory at the end of the end of the first file we see a similar structure (Output listing 4-8), 0xbad0 0b1e magic number (starts the highlighted text in red), followed by a file header, followed by the file itself. In this case the file is a previously deleted VLAN.dat file.

Output listing 4-8: CISCO1712 flash deleted file header

```
rommon 55 > dump  0x60cae288 0x80
60cae288  3a00 0000 169d ca00 0000 0000 0000 000d :...............
60cae298  0000 0018 feed bac0 4349 5343 4f20 5359 ........CISCO SY
60cae2a8  5354 454d 5300 0000 fade fad1 0000 0018 STEMS...........
60cae2b8  ead6 7067 bb1e d320 2e30 9799 0df4 4da3 ..pg... .0....M.
60cae2c8  bad0 0b1e 0000 0258 ba86 fffe 0000 0000 .......X........
60cae2d8  766c 616e 2e64 6174 00ff ffff ffff ffff vlan.dat........
60cae2e8  ffff ffff ffff ffff ffff ffff ffff ffff ................
60cae2f8  ffff ffff ffff ffff ffff ffff ffff ffff ................
rommon 56 >
```

Looking further into the flash memory, we find our VTP password marker in Output listing 4-9

Output listing 4-9: CISCO1712 flash active vlan.dat

```
60caed28  bad0 0b1e 0000 0258 759c ffff 0000 0000 .......Xu.......
60caed38  766c 616e 2e64 6174 0000 0000 0000 0000 vlan.dat........
60caed48  0000 0000 0000 0000 0000 0000 0000 0000 ................
60caed58  0000 0000 0000 0000 0000 0000 0000 0000 ................
60caed68  badb 100d 0000 0002 0200 0000 0000 0000 ................
60caed78  0000 0000 0000 0000 0000 0000 0000 0000 ................
60caed88  0000 0000 0000 0000 0000 0000 0000 0000 ................
60caed98  0000 0000 0000 0001 3030 3030 3030 3030 ........00000000
60caeda8  3030 3030 827f 244a ee8a 68bb e96e 048e 0000..$J..h..n..
60caedb8  4fa7 82d5 0e4d 4152 4b6c 7363 416c 7658 O....MARKlscAlvX
60caedc8  696d 6e00 0000 0000 0000 0000 0000 0000 imn.............
```

Comparing the file headers of active and deleted files a pattern emerges of the file meta data and contents – as shown in Table 4-7. There is no file system table. Files with their headers are laid out sequentially after each other. And file deletion is performed by changing a flag in the header.

Table 4-7: Probable CISCO1712 flash file format structure

Magic start of header 0xbad00b1e (4 bytes)	Length 4 bytes	Checksum 2 bytes	Flags 2 bytes 0xffff = active 0xfffe=deleted	Padding 2 bytes	File name. 0x00 terminated. Padded to 48 bytes	File contents

This corresponds well to the C source code class B file header in the Cisco Flash File System tool written by Simon Evans [79]. The date field in our case is not set but that makes sense as the CISCO1712 does not have a real-time clock (and I didn't use any Network Time Protocol time source during the tests).

Table 4-8: Class B file header from fileheader.h from http://si.org/cffs/

```
#define CISCO_CLASSB 0xBAD00B1E

/* Class B file header */

struct cb_hdr {
   uint32_t   magic;     /* CISCO_CLASSB */
   uint32_t   length;    /* file length in bytes */
   uint16_t   chksum;    /* Chksum */
   uint16_t   flags;
   uint32_t   date;      /* Unix date format */
   char    name[48];  /* filename */
};

/* Class B Flags */

#define FLAG_DELETED   1
#define FLAG_HASDATE   2
```

4.1.3.7 *Writing to flash*

How can we write to the flash memory from the tools offered in Rommon priv mode? Using the memory write commands alter and fill generates a “Machine Check Exception” while writing to the flash region. Probably because the CPUs memory controller is configured for the flash memory region to be write protected (see Register#6 in Output listing 6-6 in Appendix C). From the flash pre-study on flash memory in Section 2.1.3 we know that it should be possibly to reprogram a 1 to 0. The file system creators probably designed the delete flag with this flash property in mind, hence the bit is set at the time the file is first written and can easily be cleared later to perform a file “delete” – without having to re-write the block that this file header is in. After experimenting with various Rommon priv mode commands, I arrived at this conclusion for how to write to flash:

- The command “flash erase” starts erasing the whole flash, eventually leaving it in an all 0xFF state. The erase process takes longer than a minute, but can be stopped prematurely by sending a RS-232 break signal.
- The command “flash prog <source> <destination> <size>” programs (i.e. flipping 1 to 0) the flash memory starting at <destination> by copying <size> bytes from a <source>. Source address can point to flash, RAM, or NVRAM regions. <Size> must be at least 2.

Using our new flash write knowledge, we test the idea of how the deleted flag functions by modifying this flag for the first executable file which is currently visible. In our case, this is the IOS file listed in Output listing 4-6 and with a file header shown in Output listing 4-7. That is, we need to flip the least significant bit from 1 to 0 in the byte at address 0x6000000A. The “flash prog” command needs to read from RAM. According to the *meminfo* command output in Output listing: 4-3 RAM starts at address 0x10000 (64k). In this router we have 96MB to use. Address 0x2000000 is roughly in middle of that address range and looks empty so we can probably use it as a buffer for storing temporary data. According to the flash chip datasheet, this chip has an erase block size of 128 Kbyte = 0x2000000 [78]. We could program as little as two bytes but in this test we will work on a block size quantity of memory.

The console log in Output listing 4-10 shows the method being tested. The first block from the flash (containing the file header) is copied to RAM, the delete flag is changed (0xFFFF > 0xFFFE), and the block is copied back to flash. Executing a *dir flash* command shows that the file is now indicated as “deleted”.

Output listing 4-10: CISCO1712 modify file delete flag

```
rommon 60 > #Move first flash block to RAM
rommon 61 > move -b  0x60000000 0x2000000 0x20000
rommon 62 >
rommon 62 > #Modify delete flash
rommon 63 > alter -w 0x200000A
200000a = ffff > fffe
200000c = 0000 > quit
rommon 64 >
rommon 64 > #copy back
rommon 65 > flash prog 0x2000000 0x60000000 0x20000
Programming location 60010000
rommon 66 > dir flash:
         File size           Checksum   File name
  13296264 bytes (0xcae288)   0x2384    c1700-k9o3sy7-mz.123-11.T9.bin (deleted)
       600 bytes (0x258)      0xba86    vlan.dat (deleted)
       600 bytes (0x258)      0x2c42    vlan.dat (deleted)
       600 bytes (0x258)      0x7cf8    vlan.dat (deleted)
       600 bytes (0x258)      0x94e9    vlan.dat (deleted)
       600 bytes (0x258)      0x759c    vlan.dat
```

4.1.3.8 Conclusions on using the ROM monitor

We have successfully showed that the Rommon mode can be used to read from the two non-volatile memories of this router. Although we were not able to write to the NVRAM EEPROM, it is likely to be possible with some additional research. The main problem using this approach is speed. All data transfers take place over the terminal’s RS-232 asynchronous console and the console port’s maximum speed is limited to a maximum baud rate of 115,200 bps. Transferring the full 32 KB EEPROM contents is fast, but reading the entire 32MB flash at this maximum console speed with the overhead of the hexdump format would take about 2 hours. Although this is doable, it is not very efficient. This is especially true since vendors typically increase flash sizes on their devices over time as their software gets bigger. However, the console baud rate has not increased proportionally. Rommon has TFTP download transfer capabilities, but I have not found any way to upload data using TFTP. Perhaps it might be possible to change the first file size so it spans the entire flash, boot the IOS over TFTP and then transfer that whole file to a TFTP server over Ethernet. However, the forensic value of doing so is limited because once we boot IOS the running code can modify the content of the flash and EEPROM we want to examine. So this method would have to be combined with some hardware write protection of the non-volatile memories, such as pulling $\overline{\text{WE}}$ to VCC on the memory chips.

Erasing memories can be done without transferring all of the data via the terminal interface. A flash and RAM block could be written to with a pattern and the *compare* function can be used to verify that the RAM and flash contents match.

From a professional refurbisher's perspective, the rom monitor is a practical method to inspect and change non-volatile memories, but is too slow for transferring large amounts of data. An automated (scripted) tool could be made to erase a number of devices in parallel and this would effectively reduce the time needed per device. The benefit of this approach would be ease. Nothing has to be soldered and the device enclosure does not even have to be opened. However, the erasure and verification using this approach are dependent upon the correctness and trust of the Rommon program.

4.1.4 JTAG exploration of the CISCO1712 mainboard

Header pins were soldered to the row of 10 through holes labeled J1 JTAG. The JTAGulator was connected but did not find the JTAG signals. The CPU was removed to expose its BGA connector pads pins and the pinout checked with continuity check using a multimeter. The result is in table

Table 4-9: CISCO1712 J1 JTAG port pinout.

J1 Pin#	CPU Pin#	CPU pin name
1	GND	
2		
3		
4		
5	N4	$\overline{\text{HRESET}}$
6	G18	TMS
7	H17	TDI/SDI
8	G17	TDO/DSDO
9	H16	TCK/DSCK
10	GND	

All 4 required JTAG signals should be present on pins 6-9 but the JTAGulator was not able to communicate with the TAP. It seems the JTAG functionality of the CPU is disabled by configuration. The MPC862 CPU debug pin functionality is controlled by 4 bits in the Hard Reset Configuration Word sampled from the data lines during startup. The priv command register printout (see Output listing 6-5 in Appendix C) shows this so-called System Interface Unit Module Configuration Register (SIUMCR) : siu_mcr : 0x00230440. Bits 11 and 12, counting from bit 0=MSB is "10" which according to the manual means "reserved" and seems to disable JTAG. And the flag bit 15: "Debug register lock" is set too, preventing the JTAG port from being enabled by writing a new SIUMCR to the IMMR address of 0xff000000.

I will leave to future work to research how to enable the JTAG port on the CISCO1712.

4.1.5 BDM port access to the CISCO1712

Header pins were soldered to the 10 solder pads labeled “J3 CODE TAP”. The CPU was de soldered to expose its BGA connector pads and the pinout derived from a continuity check using a standard multimeter. The result is shown in table Table 4-10.

Table 4-10: CISCO1712 J3 BDM port pinout.

J3 Pin#	CPU Pin#	CPU pin name	Meaning according to CPU manual [76 Table 12-1]
1	G3	FRZ/IRQ6	Freeze
2	P2	$\overline{\text{SRESET}}$	Soft reset
3	GND		
4	H16	DSCK/TCK	Provides clock to scan chain logic or for the development port logic.
5	GND		
6	G3	FRZ/IRQ6	Freeze
7	N4	$\overline{\text{HRESET}}$	Hard Reset
8	H17	DSDI/TDI	Input serial data for either the scan chain logic or the development port and determines the operating mode of the development port at reset.
9	F16	VCC	
10	G17	DSDO/TDO	Output serial data for either the scan chain logic or for the development port.

This corresponds to the standard pinout of a Freescale BDM port [80p. 2]. I connected a specialized hardware tool called Cyclone MAX from PE Micro. The vendor says this device can debug Motorola/Freescale CPUs over the BDM as well as program flash memories behind the CPU.

I attached the Cyclone MAX to the CISCO1712 BDM port and connected it via USB to a computer running a version of Microsoft Windows and fired up the PROGPPC tool that PE Micro recommends to program flash memory. The tool would successfully connect to the CPU, so I thought it would be an easy task to read and write to the flash, since we know the physical address is (0x60000000) and the size from the Output listing 4-5 in the rommon investigations. However, this proved to be far from easy. It seems the problem is that the programmer begins it operation by resetting the CPU, and with that the whole CPU configuration including the memory manager unit. It then reconfigures the CPU and transfers a small program to the CPU to execute and to program the flash. I could not make it configure the CPU memory manager unit as I wanted. The preffered way would be to start the device normally and have it configure the CPU and then let the debugger intercept, and continue controlling the CPU with an already properly configured memory manager. The vendor said it might be possible, but after 2 months they still have not supplied a solution, therefore I gave up on this track. My perception is that this tool is not very helpful when trying to reverse engineer existing devices. It was indeed a misspent US$1000.

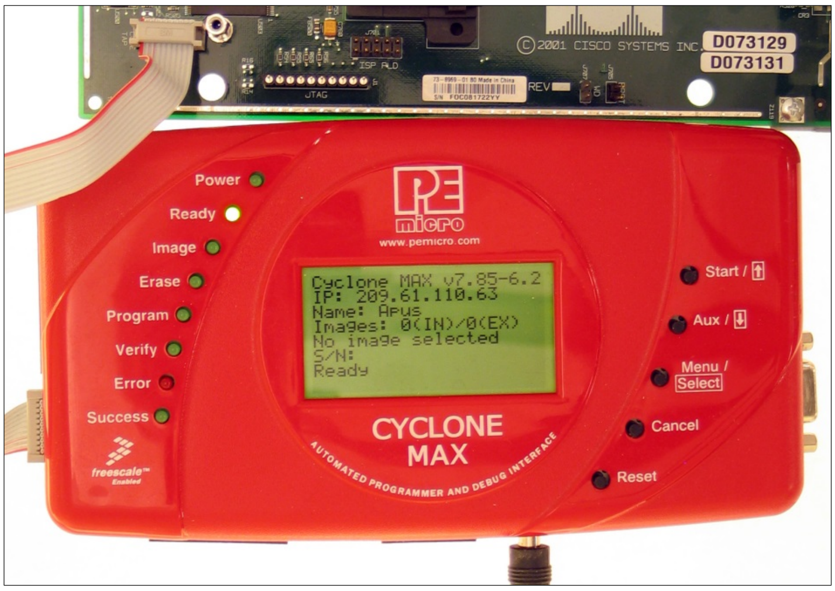


Figure 4-10: Cyclone MAX BDM debugger connected to a CISCO1712 BDM port

The benefit of using this device would be speed, as this interface seems to have a high transfer rate. However, from a refurbisher's point of view utilizing the BDM port on the CISCO1712 is still a time consuming method to sanitize the device, because:

1. The enclosure has to be opened.
2. The crypto card must be dismounted.
3. For BDM board connection either header pins have to be soldered or some spring loaded connector must be pushed against the solder pads. Preferably, the logic board should be placed in a customized jig to hold it in place and secure the spring loaded connector against the solder pads.

This work may not sound much, but if you have thousands of units to process every month each minute of manual work is important. Considering the work above, I believe the BDM method does not really fit into any category of use for this device. A professional refurbisher would prefer the RS232/rommon interface (explored in Section 4.1.3), since this is accessible from the outside. The transfer rate is not important because the devices can be hooked up to a scripted tool and several devices can be processed in parallel without any manual labor required while the script is running.

A forensic investigator, who may not trust the rommon code to provide accurate data, would simply desolder the chips and read out the contents from an external programmer.

However, there may be other devices where the BDM port is in fact the best option, so I will still relegate further investigation of this method to the future work Section in 6.3.

4.1.6 Using a programmer to access the NVRAM of the CISCO1712

The CAT 28C256-12 32KB PLCC32 EEPROM is soldered to the logic board. Would it be possible to read and write to it using a programmer while it is soldered to the motherboard? I made an adapter between the Pomona PLCC32 testclip and a DIP placeholder according to the datasheet [75]. The DIP adapter was inserted into a GQ-4X programmer as shown in Figure 4-11. To verify the setup, a loose de-soldered EEPROM was inserted into the jaws of the test clip and successfully read from.

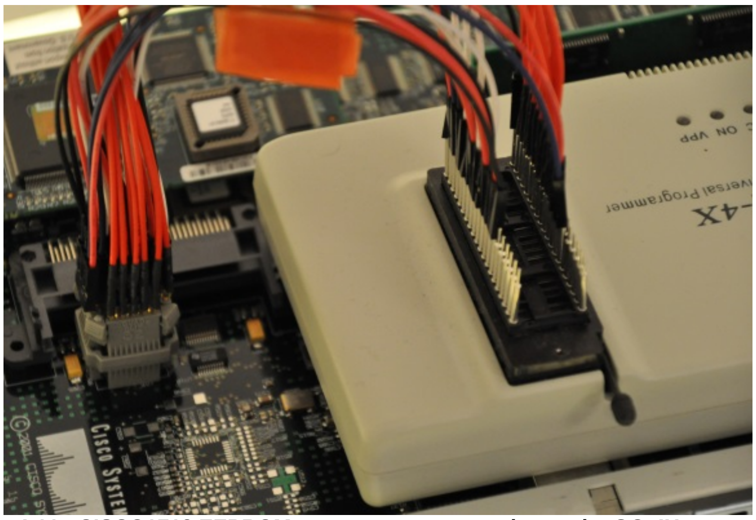


Figure 4-11: CISCO1712 EEPROM programmer connection to the GQ-4X programmer

However, reading from the EEPROM on the board was not successful. If the router was powered off, one could notice the front LEDs starting to light up. So it the programmer was trying to provide 5V power to the entire router via the EEPROM VCC pin. I disconnected the Vcc supply from the programmer and powered the router on using its normal power supply. The programmer would just read nonsense and random data.

The problem is that the CPU (or intermediate circuitry) is actively driving the pins of the EEPROM to TTL logic levels; hence our programmer cannot do its job. For instance, the $\overline{\text{OE}}$ pin is at 3.3V immediately after power on. The shortcut current to ground is 91 mA so there is low impedance to Vcc of 5V. For the programmer to work the surrounding circuitry interfacing the EEPROM's address, data I/O and control signals must be put in high impedance mode. The CPU manual suggest that HW reset can be used to put address and data lines in high impedance mode [76 Table 12-1. Signal Descriptions]. I tried combinations of the CPU's HW reset and CPU Freeze pins on the BDM port, but could not make it release its bus control of the EEPROM.

According to the CPU manual, there is a JTAG instruction called HI-Z which puts all output pins into high impedance mode [76Para. 46.4.5]. Since the JTAG TAP interface is disabled at CPU boot (see Section 4.1.4) testing the HI-Z JTAG instruction in combination with the programmer is left for future work.

The web article “Understanding In-Circuit EEPROM and Microcontroller Reading and Programming” [81] proposes some tricks to access serial EEPROMs in-circuit. One of them is to power the system with a reduced voltage. The idea is to find a voltage window where the CPU will not start (and take control of the bus), but still sufficient to power the EEPROM. In our case, we have a parallel EEPROM - but perhaps the same ideas could be applied.

According to the CPU hardware specification the processor can operate between 3.135V and 3.465V [82 chapter 6 table 5]. The EEPROM datasheet says it has a nominal supply voltage of 5V ±10%, but at the same time it mentions a “write inhibit function” that prevents writes when the supply voltage drops below 3.5V. Perhaps that means it is still possible to ***read*** from it below 3.5V.

To complicate things the EEPROM data, address, and control lines is not connected directly to the CPU. I have not investigated this entirely, but there is at least a transceiver chip and a programmable logic device involved. Therefore, I tried the simple approach: Lower the “5V” supply voltage in 0.5V steps and see if there is a level where the programmer successfully can read out the EEPROM contents.

The CISCO1712 has an external power supply providing 5V as well as +12V and -12V. It seems to start fine on 5V only, with the exception that RS232 does not work. As such, I conclude that the 3.3V to the CPU is derived from the 5V supply.

I opened the router power supply and connected a variable output power supply to the 5V lead. The "5V" was then gradually reduced in steps of 0.5V to see if there was any level where the GQ-4X programmer could read out the EEPROM contents. No success. I noted though that the router went into reset once the "5V" router supply voltage level fell below 4.7V.

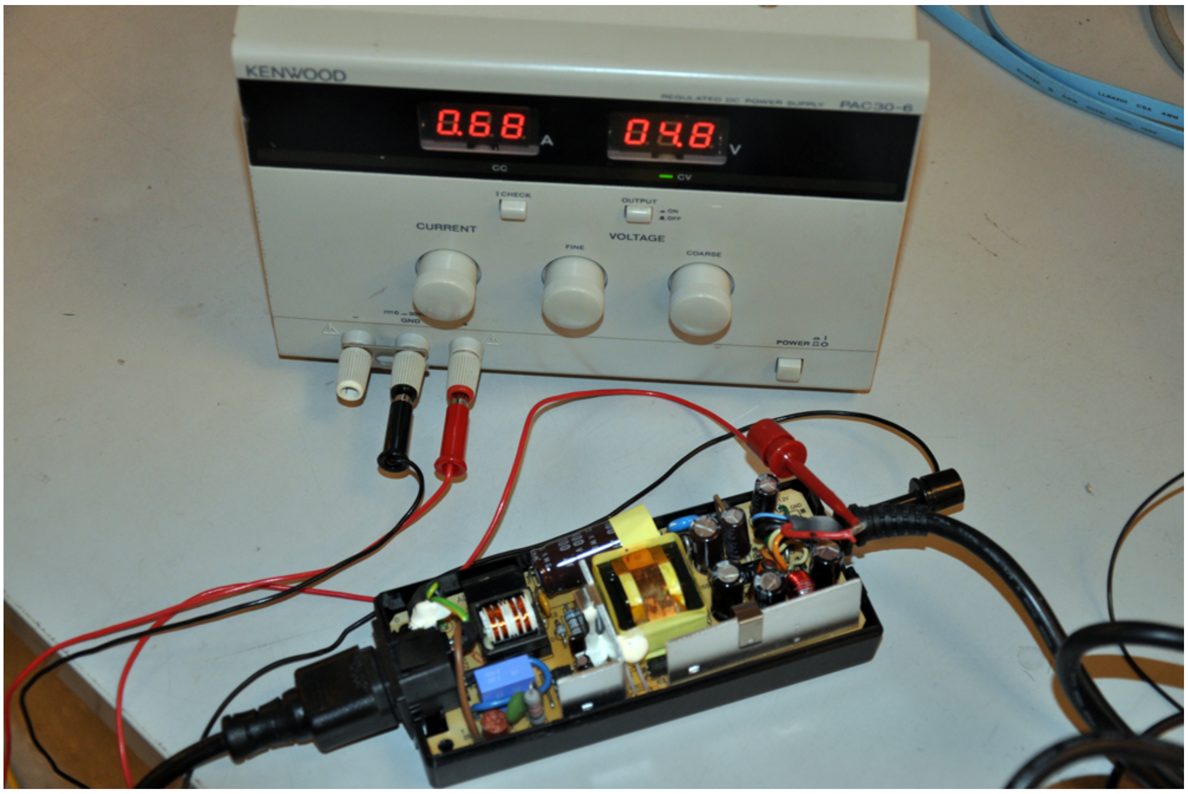


Figure 4-12: CISOC1712 power supply with external voltage control on its 5V lead.

Perhaps the low cost GQ-4X programmer cannot supply enough current to force the pins to TTL levels in competition with the other drivers on the bus. After Xeltek assured me their $2000 SuperPro programmer had current limiters on their interface pins I dared to use it for the same procedure (although they said that on-board programming of parallel chips is not recommended). Their Windows tool reported a variety of error messages and data was not read correctly at any voltage level tested. Although the programmer screen incorrectly reported "READ OK!" (see

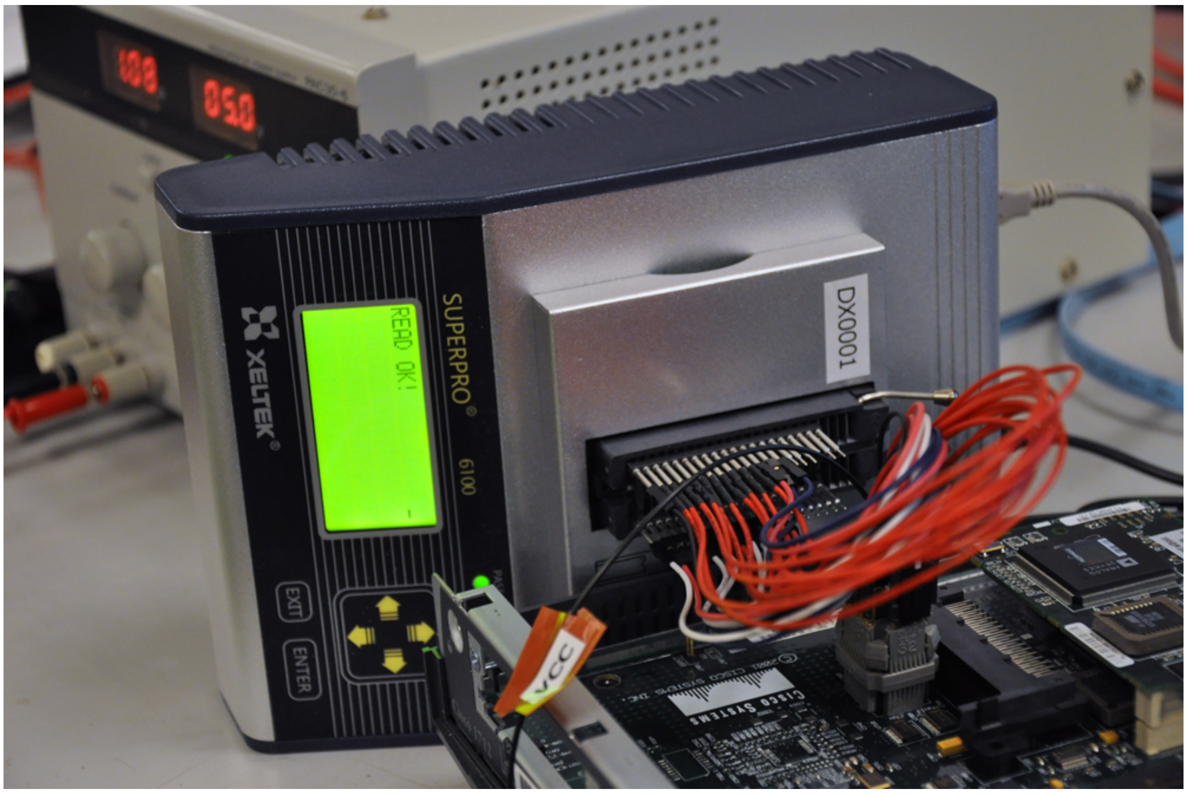
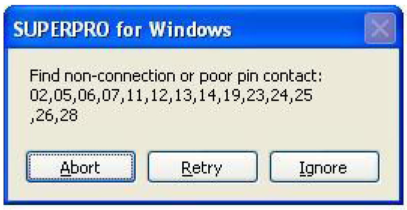
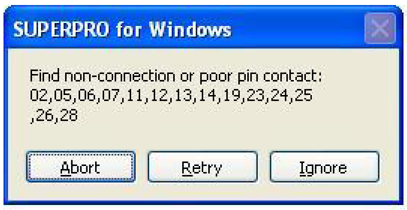
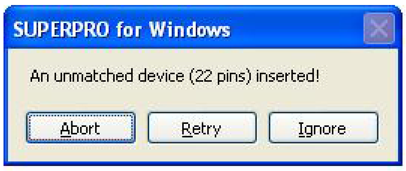
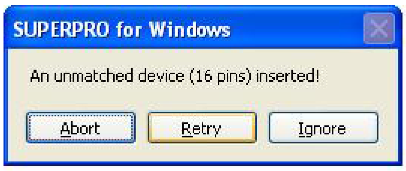
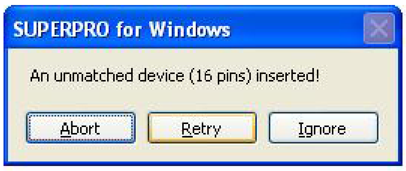
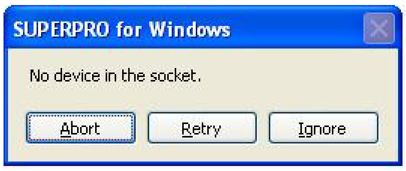


Figure 4-13) The results are shown in Table 4-11. (And yes, both the router and the programmer survived these tests).

Table 4-11: CISCO1712 onboard NVRAM read result with a Xeltek SP6100 programmer and varying supply voltage

"5V" Supply voltage to router	Data read	Xeltek Winpro error message
5.0V	rubbish	 SUPERPRO for Windows Find non-connection or poor pin contact: 02,05,06,07,11,12,13,14,19,23,24,25 ,26,28
4.7V	All 0xFF	 SUPERPRO for Windows Find non-connection or poor pin contact: 02,05,06,07,11,12,13,14,19,23,24,25 ,26,28
4.5V & 4.0	All 0xFF	 SUPERPRO for Windows An unmatched device (22 pins) inserted!
3.5V, 3.0V and 2,5V	All 0xFF	 SUPERPRO for Windows An unmatched device (16 pins) inserted!
2,0V	All 0x0c	 SUPERPRO for Windows An unmatched device (16 pins) inserted!
1.5V	All 0x00	 SUPERPRO for Windows No device in the socket.

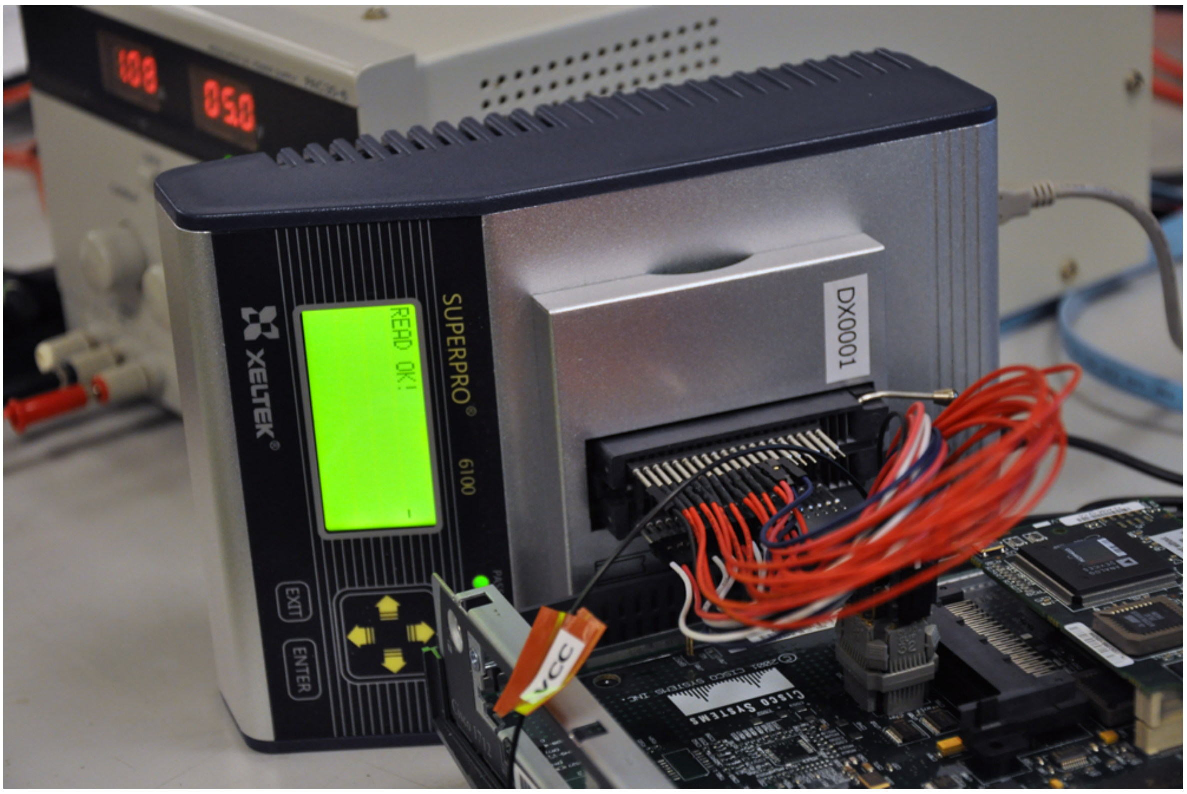


Figure 4-13: Programmer connected to CISCO1712 NVRAM chip.

I have not found any advice or evidence that a parallel EEPROM can be successfully programmed in circuit. However, I also have not found any evidence that it is impossible. In fact, it will be shown later (in Section 4.1.8.3) that we can force a low TTL level into high by connecting the EEPROM $\overline{\text{WE}}$ pin to VCC, without damaging the conflicting driver output stage.

Perhaps this method can work for all pins if the circumstances are favorable. For example, competing drivers have some current limiter, and our programmer can supply enough current to change the TTL level. Thus in "Mythbusters" terminology I will call in-circuit programming of a parallel EEPROM *plausible,* but relegate it to the future work section.

4.1.7 Other board debug ports on the CISOC1712

The 2 x 5 pre soldered header labeled J701 ISP PLD has a JTAG TAP interface to the Altera MAX EPM7128AETC100 in the center of the board. This was verified with the JTAGulator and the pinout is TDI: 9, TDO: 3, TCK: 1, TMS: 5

I do not know the exact purpose of the Altera MAX chip but according to the datasheet it is boundary scan / EXTEST instruction capable [83]. I/O pin 55 has a galvanic connection (continuity checked) to the $\overline{\text{WE}}$ pin on the NVRAM EEPROM chip. The Altera chip is both 5V and 3.3V capable according to the datasheet. Since the CPU is 3.3V and some components on the board such as the EEPROM is 5V I thought this could work as some level converter between the two sides. However, neither the MSB nor LSB of the CPU address or data bus pins are connected to it. Exploring the functionality of this chip is left for future work.

The 2 x 10 board edge connector labeled J703 has power input lines and also has pins that connect to the console port's RX and TX pins. Close inspection of the pins shows scratch marks so this connector is used during production. My guess is that the all the surface mount components are placed first and tested using the board edge connector, since the power and console ports may not be soldered on yet.

4.1.8 Investigation of the effect of vendor sanitization commands on CISOC1712

To reset a router running IOS 12.3 mainline software to factory defaults Cisco advices use either of the two procedures (that I have labeled CISCO_IOS_1 and CISCO_IOS_2) in Appendix A. The first procedure erases the non-volatile configuration, while the second loads a default configuration that is subsequently written to non-volatile storage. To test if these two procedures properly sanitize a router the 4 markers shown in Table 4-12 were configured. A 512-bit RSA key was also generated, as shown in Output listing 4-11.

Table 4-12: CISCO1712 markers

Parameter to host marker	**String marker (10 random chars)**	**Parameter storage**	**Storage size**	**Marker Strength**
SNMP password	MARKyeTBGfMWEF	NVRAM (startup-config)	32KB	1 - 2*10^{-13}
VTP Password	MARKpBcFsXZGVs	flash:/vlan.dat	32MB	1 - 2*10^{-10}
TFTP_FILE	MARKNuAjQAyJtv	NVRAM (environment variable)	32KB	1 - 2*10^{-13}
Hostname	MARKgJlBDKowbK	NVRAM (startup-config)	32KB	1 - 2*10^{-13}

Output listing 4-11: CISCO1712 RSA key generation

```
MARKgJlBDKowbK(config)#crypto key generate rsa
The name for the keys will be: MARKgJlBDKowbK.domain.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]:
% Generating 512 bit RSA keys ...[OK]

MARKgJlBDKowbK(config)#
Mar 28 13:35:47.999: %SSH-5-ENABLED: SSH 1.99 has been enabled
<SNIP>
MARKgJlBDKowbK#show crypto key mypubkey rsa
% Key pair was generated at: 13:35:47 UTC Mar 28 2006
Key name: MARKgJlBDKowbK.domain.com
 Usage: General Purpose Key
 Key is not exportable.
 Key Data:
  305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00CF96DE 6729EECF
  E3BB3230 7D760657 9F1AA209 A9DCAF31 E8DA7ECB 5102FD83 72793048 7A61A1BD
  40DA9F65 09AD11FA DF74BDEC 0F904580 B1D29E35 2D173739 31020301 0001
% Key pair was generated at: 13:35:49 UTC Mar 28 2006
Key name: MARKgJlBDKowbK.domain.com.server
 Usage: Encryption Key
 Key is exportable.
 Key Data:
  307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00BD1B4B 6B239E4B
  4BAB4835 236447C5 B2DAE27E 060E35B0 E326C069 79063CF2 03B1AA84 578FE4E7
  FA46C28D E4CF7BEA 381C1293 58B4C46A E7AC2409 D7D4E017 C2834AA9 2A47BE65
  D07496AA 405BD2A5 4B35EDCC E05B53BD 819FCD19 9F3CD80E 25020301 0001
```

The configuration was then copied to startup-config. Note that the cryptographic key is *not* stored in the startup-config file itself, but rather in a special file in NVRAM called private-config. That file does not have the read flag set (as shown in Output listing 4-12) and thus is inaccessible via the built-in IOS “more” command. However, we can see the contents by inspecting the NVRAM chip’s memory contents using the IOS command “show memory 0x68000000”. Output listing 6-7 in Appendix C shows the start of the private key at address 0x6800101F.

Output listing 4-12: CISCO1712 private-config in “dir NVRAM:”

```
MARKgJlBDKowbK#dir nvram:
Directory of nvram:/
    25  -rw-        1140                    <no date>  startup-config
    26  ----        1127                    <no date>  private-config
     1  -rw-           0                    <no date>  ifIndex-table
     2  ----          12                    <no date>  persistent-data

29688 bytes total (25321 bytes free)
```

The VTP password marker is verified as being stored in the in the flash:/vlan.dat file as shown in Output listing 4-13.

Output listing 4-13: Marker verification in vlan.dat file before sanitization

```
MARKgJlBDKowbK#more flash:/vlan.dat
:[^P
^@^@^@^B^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^A000000000000/o
   ^[^\/^^U.b^Y^Qp^R^NMARKpBcFsXZGVs^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^E^B^B^@^@^Bt0tdefault^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^A^E\^@^A^@^A^F!^S^@^@^@^C
j^Ck^@^@^@^@^@^@^Lfddi-
default^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^B^A^E\^Cj^@^A


^S^@^@^@^@^@^A^Ck^B^@^@^@^@^@^Rtoken-ring-
default^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^A^E\^Ck^@^A
^K^@^@^A^@^Cm^@^A^Cj^B^A^@^@^Ofddinet-
default^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^D^A^E\^Cl^@^A
^L^S^A^B^@^@^@^@^@^@^B^@^@^@^@^@
trnet-default^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^E^A^E\^Cm^@^A


^S^A^B^@^@^@^@^@^@^B^@^@^@^@^@^Bt0|^@^@^@^A^@^@^@^Bt08^E^B^@^@^Cj^Ck^Bt1^D^@^@^Cj
^@^@^@^P^Bt1@^A^A^@^@^D^A^@^@^E^B^@^@^@^A^Ck^Bt2^L^@^@^Ck^@^@^@^T^Bt1H^A^A^@^@^B^
A^@^A^D^A^Cm^E^B^@^@^@^A^Cj^Bt2^T^@^@^Cl^@^@^@^Bt2P^B^A^@^A^C^A^@^B^@^@^@^@^@^C
m^@^@^@^Bt2X^B^A^@^A^C^A^@^B
MARKgJlBDKowbK#
```

The vendor proposed procedure CISCO_IOS_1 was performed followed by CISCO_IOS_2 (Apendix A) and the router powered off. The router was then booted in Rommon and the NVRAM was inspected using the PRIV mode "dump" command. The SNMP password and hostname markers had been removed, but the TFTP_FILE marker was still present.

The router was then booted and the vlan.dat file was still present with the marker in the flash file system. Since both routines were performed in sequence, I conclude that *neither* of them has erased the TFTP_FILE or VTP password markers. While not proven formally via this test, I suspect that the factory default reset routines leave all of the environment variables and flash files intact. What information leakage could this cause? The environmental variables are used to control the boot process and to configure IP addresses and file names to load a new software file over TFTP. This information could identify the previous owner's TFTP server IP address; however, I consider this to be quite harmless. If the TFTP server has a public IP address it is exposed already. If it has a private IP address behind a firewall it is protected from outside access. However, the retention of the VTP password is more serious as will be addressed in the following paragraphs.

4.1.8.1 The vlan.dat file

The information in the vlan.dat file is potentially sensitive. This information includes the VLAN database, which gives insight into how the previous owner's LAN network was structured. VTP is a protocol used to distribute VLAN information between switches, with one switch acting as a server and distributing VLAN numbers and names to the other switches that act as clients. Given the VTP password and layer 2 trunk port access to a network it would be possible to pretend to be the VTP server and distribute a new VLAN list to the clients. Alternatively, one could distribute an empty list to remove all VLANs, causing all of these VLANs to be unavailable. This procedure is described in Section "4.6 VLAN Trunking Protocol (VTP) Attack" in the paper "Virtual LAN Security: weaknesses and countermeasures" [80 Ch. 4.6].

However, to make use of this attack we would need access to a layer 2 trunk port of the previous owner's network. A trunk port is used to carry VLAN traffic between switches and should **not** be possible to reach from a public space, such a lobby, via WIFI, or from an Internet Service Provider. However, according to a Cisco manual the default mode for all LAN ports is to be configured in a state called "switchport dynamic desirable":

"Makes the LAN port actively attempt to convert the link to a trunk link. The LAN port becomes a trunk port if the neighboring LAN port is set to ***trunk****,* ***desirable****, or* ***auto*** *mode." [85 Table 17-2 Layer 2 LAN Port Modes]*

Therefore, if the switch administrator forgot to actively disable this default setting, the port can become a trunk port and a VTP attack is possible.

Would it be possible to do a VTP attack over the internet into a target LAN? It is difficult, but maybe be possible. Andrea Barisani & Daniele Bianco showed at the BlackHat conference 2013 that an arbitrary Ethernet frame can be injected over the internet, via routers and firewalls, into a LAN [86]. They constructed a normal IP packet, such as an ICMP Echo reply ("ping") packet, but with a crafted payload that looked like an Ethernet frame. If the switch port happens to restart while this packet was in transit "on the wire" the port might start its frame synchronization algorithm in the payload, then find the maliciously embedded frame and treat it as a valid Ethernet frame. In the general case an outside attacker does not know the destination MAC addresses inside the target LAN which makes this attack difficult. However, the VTP protocol uses a *fixed* destination MAC address of "01-00-0C-CC-CC-CC" [87]. If the switchport would restart in trunk mode and accept a VTP frame injected as payload over the internet as the *first* packet on the restarted port, this exploit is theoretically possible to carryout, even remotely over the internet. Confirming this type of attack is left for future research.

4.1.8.2 Summary of sanitization using vendor factory reset commands.

The vendor sanitization commands of a CISCO 1712/K9 with IOS "C1700-K9O3SY7-M, Version 12.3(2)XF" erased the startup- configuration in NVRAM. The environmental variables in NVRAM were left and as well as flash files such as the vlan.dat. Table 4-13 shows the results.

Several blogs and end user recommendations advise removing the vlan.dat file as part of the sanitization process (for example [88] and [89]) and even the vendor underlines the important of this in their switch documentation [90]. However, they advise the use of the command "delete flash:/vlan.dat". From our previous examination of the CISCO 1712 file system (see Section 4.1.3.6) we learned that that command only flags the file as deleted, but does not actually remove any data, hence simply executing this command is insufficient.

I eventually found a note deep down in the vendor documentation about configuration management [91Para. Specifying the Startup Configuration File]: stating that the command "squeeze" has to be used to reclaim the flash space after erasing the startup-config from there. The advice is for reclaiming storage rather than a sanitization advice, but it gives a hint what we need to do to sanitize the router from the vlan.dat file.

I ran the "delete flash:/vlan.dat" command followed by a "squeeze flash:". After that, I dumped the full 32MB flash memory contents from Rommon/Priv mode. The VTP marker inside the VLAN.dat could not be found, so the delete and squeeze combination seems to be sufficient.

The overall assessment of the vendor erase routine is that it overwrites the startup-config in NVRAM correctly. It leaves the environmental variables intact, but that is a minor problem as they do not contain very sensitive information. Unfortunately, the flash memory is left intact; therefore, it is up to the user to delete any files that contain sensitive information (such as vlan.dat and config backups) **and** to actually overwrite these files using the "squeeze" command.

Table 4-13: Sanitization results of the CISCO1712

Parameter to host marker	Parameter storage	Storage size to search	Result	Evidence Strength	Security severity assessment
SNMP password	NVRAM (startup-config)	32KB	Erased by CISCO_IOS_1 and /or CISCO_IOS_2	Not calculated	
VTP Password	flash:/vlan.dat	32MB	Not erased by either CISCO_IOS_1 or CISCO_IOS_2	$1 - 2*10^{-10}$	Severe
TFTP_FILE	NVRAM (env variable)	32KB	Not erased by either CISCO_IOS_1 or CISCO_IOS_2	$1 - 2*10^{-13}$	Negligible
Hostname	NVRAM (startup-config)	32KB	Erased by CISCO_IOS_1 and /or CISCO_IOS_2	Not calculated	
RSA key (512 bit)	NVRAM (private-config)	32KB	Erased by CISCO_IOS_1 and /or CISCO_IOS_2	Not calculated	

4.1.8.3 *IOS verification for NVRAM chip erase*

Can we trust the NVRAM erase routine in the IOS to be foolproof? Does IOS actually verify that the data is actually overwritten? Consider the fact that the NVRAM chip used in the CISCO 1712, CAT 28C256-12, has a feature to disable all writes if the supply voltage falls below 3.5V.

I tested the IOS behavior by erasing the NVRAM chip with $\overline{\text{WE}}$ connected to Vcc[*], that is basically write protecting the chip. The normal behavior of a successful erase is shown in Output listing 4-14. The result of the failed erase of the write protected NVRAM is shown in Output listing 4-15. There are three errors reported (highlighted in red), but in the end the notification “[OK] Erase of nvram: complete” is printed, exactly as in the case of a successful erase. Thus if one has designed an automated script to erase the NVRAM one **must not** trust the success string as proof of NVRAM erasure.

[*] This could potentially damage any other driver output stage trying to drive $\overline{\text{WE}}$ to 0. However, in this case everything recovered fine and the NVRAM could still be written to after Vcc was removed from the $\overline{\text{WE}}$ pin.

Output listing 4-14: Successfully erasing an NVRAM

```
Router#write erase
Erasing the nvram filesystem will remove all configuration files! Continue?
[confirm]
[OK]
Erase of nvram: complete
Router#
*Mar  1 00:19:34.799: %SYS-7-NV_BLOCK_INIT: Initialized the geometry of nvram
```

Output listing 4-15: Failing to erase an electrically write protected NVRAM

```
Router#write erase
Erasing the nvram filesystem will remove all configuration files! Continue?
[confirm]
EEPROM byte write error - timeout
EEPROM byte write error - timeout
EEPROM write error - timeout[OK]
Erase of nvram: complete
Router#
*Mar  1 00:18:41.299: %SYS-7-NV_BLOCK_INIT: Initialized the geometry of nvram
```

4.2 Sanitization investigation of a HP ProCurve Switch 2626

This section will look into the sanitization of a HP ProCurve Switch 2626 (Part number J4900A).

4.2.1 Switch overview and interfaces

According to the vendors' Installation and getting started guide:

> *"The HP ProCurve Switch 2600 Series devices are multiport switches that can be used to build high-performance switched workgroup networks"[88 pp. 1–1]*

This switch has 24 10/100BaseT ports and two Gigabit Ethernet ports. A RS-232 asynchronous serial interface on the back of the switch can be used for configuration. The front panel has a "port LED View" button to control the meaning of the LEDs above the ports. Additionally, there are recessed "reset" and "clear" buttons.

4.2.2 Hardware investigation

Figure 4-14 below shows the front of the switch. The main logic board with interesting parts marked is shown in Figure 4-15 with explanations given in Table 4-14.

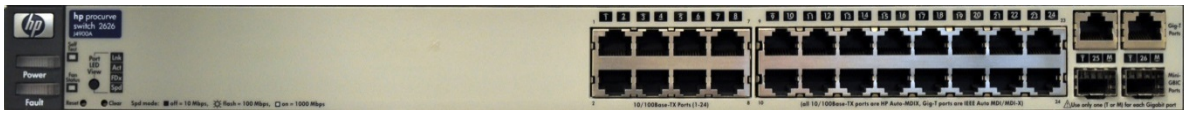


Figure 4-14: Front of HP ProCurve Switch 2626 (Part number J4900A)

Table 4-14: Interesting components of the Procurve 2626 mainboard

Object Number	Description
1	2 x 8 pin header labeled J3
2	A Motorola MPC8245 CPU
3	AMD Am29LV065D 8MByte flash
4	8 pin header labeled RP27(with pin 5 removed)
5	2 x 5 pin header labeled J4 with pairs of header pins labeled Debug, Bench and Dump
6	Solder pads labeled D3, D4, D5, D6
7	2 Lattice ispLSI5128VE programmable logic devices labeled“U21-18 V1.00” and “U22-18 V1-00”
8	U15 TI 16C752B Dual UART

Both the CPU and the Lattice chips are JTAG compatible according to their documentation [93][94]. I ran the JTAGulator on all the pins of object 1 (J3) and 4 (RP27) but it could not locate any JTAG TAP interface. One can note that the header object 5 is labeled RP27, just like the resistors close to the CPU, so perhaps it is intended to host resistors for in circuit programming of the Lattice chips.

All the right side pins of the J4 header (object 5) were ground so I concluded they were a jumper pin block to signal settings. Booting the device with the Debug jumper in place generated the terminal output shown in

Output listing 4-16, so the device seems to be waiting for a remote debugger to connect. The ribbon cables connect the serial port and a daughter card with the front push buttons.

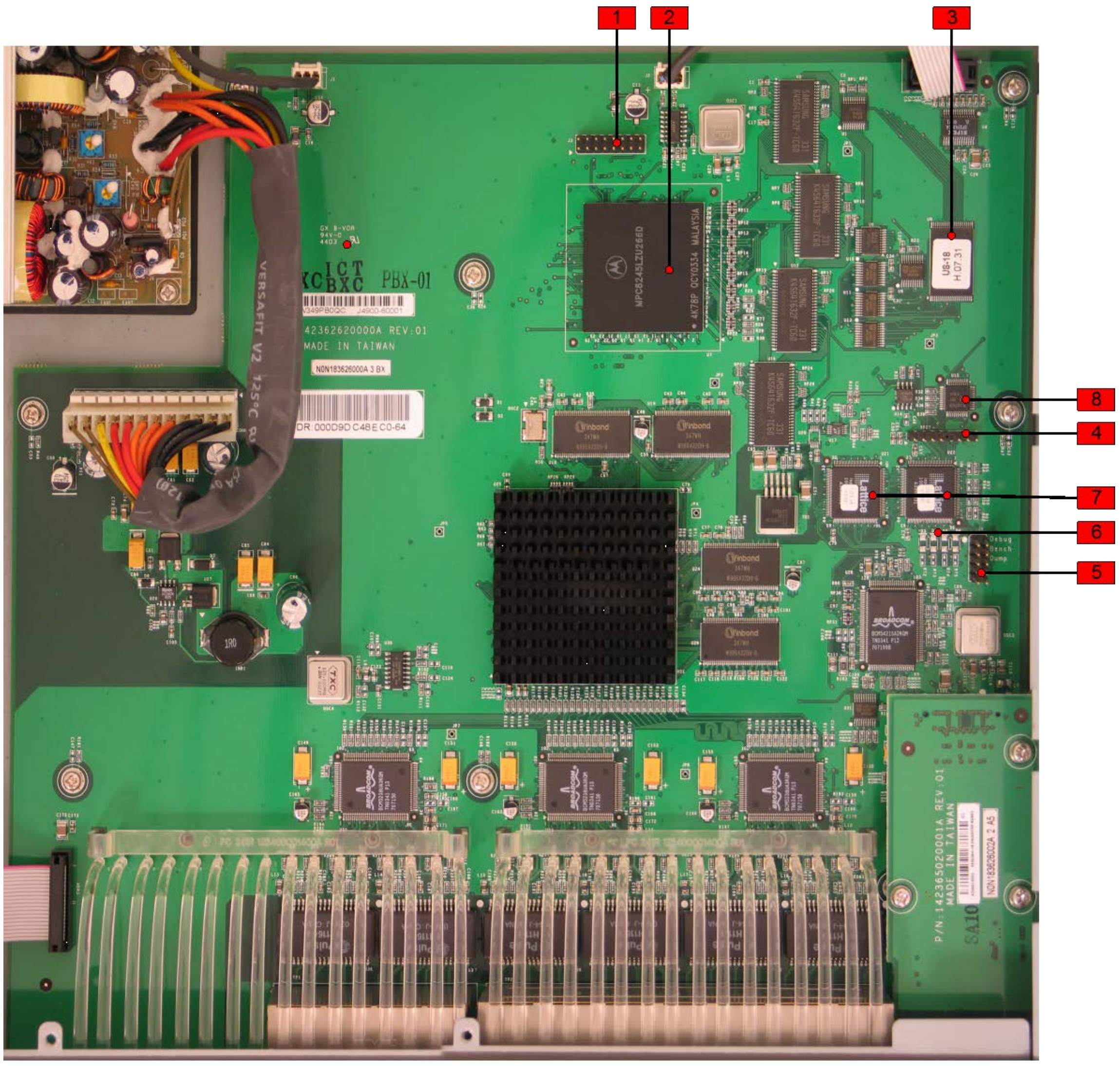


Figure 4-15: Main board ProCurve Switch 2626 (J4900A)

Output listing 4-16: Debug jumper

```
ROM information:
   Build directory: /sw/rom/build/fishrom(f04)
   Build date:      Jul 21 2004
   Build time:      10:45:52
   Build version:   H.08.02
   Build number:    137

OS identifier found at @ 0x7cb80000
Verifying Image validity ...
CRC on OS image header Passed
CRC on complete OS image file Passed
Valid OS image @ 0x7cb80000

Decompressing...done.
Answer the following questions:

Switch startup, use Debugger (y or n)? Y
System level debug session (y or n)? Y




System stopped
Bring up GDB and do a "attach system"
```

Booting with the “Bench” jumper activates a large number of extra commands. These commands can be seen in Output listing 6-8 in Appendix D. The commands starting with capital letters are the extra Bench mode commands. It is also possible to enable the same “bench” mode command set from the CLI (without the jumper) by executing the command “edomtset” twice*. [95]. And also to execute the “streboot” command. Benchmode is similar to the Cisco Rommon Priv mode with the exception that the command software here is part of the main executable file.

Booting with the “Dump” jumper looks quite similar to booting normally, except it is not possible to enter the main CLI by hitting enter twice. I do not know what this mode does. The UART used (object 8) is actually a dual port UART which seems unnecessary for controlling only a single user serial port [96]. Perhaps there is an internal UART serial interface which can be enabled to connect a debugger or to program the Lattice chips. However, the JTAGulator UART scanner function could not locate send and receive pins.

The device can store two executable images: primary and secondary. A ROM Monitor console allows selection of one of the two. There is also an xmodem download function (in case both images are somehow deleted). However, this monitor does not seem to have any memory manipulation routines, unlike the Cisco ROM Monitor. The Rommonitor is accessed by pressing the “reset” button on the front panel or by rebooting the switch with the “update” command.

* Testmode spelled backwards.

Output listing 4-17: ProCurve Switch 2626 ROM Monitor console

```
ProCurve Switch 2626# update
The device will be rebooted to Monitor ROM Console.
This will take the switch offline and allow only direct console access.
Do you want to continue [y/n]?


Enter h or ? for help.

=>?

LAN Monitor Commands

  do(wnload)                      - Download via Xmodem
  sp(eed) <baud>                  - Set a new baud rate
  h(elp)                          - Display help screen
  ?                               - Display help screen
  id(entify)                      - Print out identification string
  jp(jump) <1|2>   - Jump to product code, optional 1-primary, 2-secondary
  q(uit)                          - Exit the monitor
  boot                            - Reboot the system
  reset                           - Reset the system
  v(ersion)                       - Display version information
```

4.2.3 File system structure investigation

When booting the switch it reveals it finds a valid OS image at address 0x7cb80000 (Output listing 4-16). We assume this address is inside the 8 MB flash address range, thus we can expect to find a file header there indicating an executable file. The bench mode commands include read, write, and fill memory manipulation routines (Output listing 4-18). There is also a command to explore the file system called fs (Output listing 4-19). The result of the fs command with various options can be found in Output listing 6-9 in Appendix D.

Output listing 4-18: ProCurve Switch 2626 Bench mode memory manipulation commands

```
Read                    Read memory: r [MOPT] <ADDR>
WR                      Write memory: w [MOPT] <ADDR> <VALUE>
FILL                    Fill memory: fill [MOPT] <ADDR> <ADDR> <VALUE>
SMode                   Set Memory Mode: sm [-l<READ_LENGTH> -b -h -w -a<bhw>
                         -d<bhw> -n -i -c -s] Set default memory operation modes
                         (MOPT).
```

Output listing 4-19: ProCurve Switch 2626 Bench mode fs command

```
ProCurve Switch 2626$ fs
Usage: fs cat FILENAME          :: print file to stdout
          cp OLDFILE NEWFILE    :: copy file
          ll PATHNAME           :: file or dir listing
          ln -s OLDPATH NEWPATH :: create symlink
          ls PATHNAME           :: file or dir listing
          od FILENAME           :: od -x file to stdout
          nvfswalk              :: walk flash file system
          pnbfswalk             :: walk pnbfs file system
          ramfswalk             :: walk ramfs file system
          rm FILENAME           :: remove file
```

I believe that the filesystem works as follows. The area called pnbfs holds the logical structure of the file system, as seen by the OS. This is a directory structure of real and virtual files. For instance, it is possible to read the contents of the file “os/primary” using the “fs od” command. Output listing 4-20 shows what the executable file header looks like. The filetable seems to start at address 0x14f5178. This is far from the flash address at 0x7cb80000 so it might be a structure built dynamically at boot time and stored in RAM.

Output listing 4-20: ProCurve Switch 2626 Bench mode command "fs od os/primary"

```
roCurve Switch 2626$ fs od os/primary
00000000  40 28 23 29 20 48 50 20 4a 34 38 39 39 43 20 4a  @(#) HP J4899C J
00000010  34 38 39 39 41 20 4a 34 39 30 30 41 20 4a 38 31  4899A J4900A J81
00000020  36 35 41 20 4a 38 31 36 34 41 20 4a 34 38 39 39  65A J8164A J4899
00000030  42 20 4a 34 39 30 30 42 20 4a 38 37 36 32 41 20  B J4900B J8762A
00000040  4a 34 39 30 30 43 20 48 4d 44 4c 30 32 20 0d 0a  J4900C HMDL02 ..
00000050  40 28 23 29 20 44 6f 77 6e 6c 6f 61 64 20 46 69  @(#) Download Fi
00000060  6c 65 20 31 30 2f 30 39 2f 30 37 20 34 37 34 30  le 10/09/07 4740
00000070  20 48 2e 31 30 2e 35 30 20 20 42 75 69 6c 64 23   H.10.50  Build#
00000080  20 20 20 33 35 39 20 20 20 20 20 20 20 20 20 20     359
00000090  20 20 20 78 78 78 78 78 2d 78 78 78 78 78 20 30     xxxxx-xxxxx 0
000000a0  30 30 31 20 30 30 30 33 20 0d 0a 40 28 23 29 20  001 0003 ..@(#)
000000b0  43 6f 6d 70 61 74 69 62 6c 65 20 52 4f 4d 20 47  Compatible ROM G
000000c0  2e 30 36 2e 58 58 20 20 20 20 30 30 20 0d 0a 1a  .06.XX    00 ...
000000d0  00 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  .
000000e0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
000000f0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000100  5f 4f 53 77 00 34 60 0d 47 0b fa e5 00 00 00 05  _OSw.4`.G.......
00000110  ea 7a f5 66 00 00 00 02 00 00 01 3c 00 00 00 01  .z.f.......<....
00000120  00 01 33 88 10 00 00 00 00 32 1f 89 00 00 50 44  ..3......2....PD
00000130  00 32 23 dd 00 00 33 42 00 33 3d 7e 00 00 03 22  .2#...3B.3=~..."
00000140  0b ad ad d2 00 01 32 2c 00 01 32 2c 97 5e 3e f0  ......2,..2,.^>.
<snip>
```

The command "fs ls flash" shows the non-volatile files stored in a part of the flash memory dedicated to config files and similar files.

Output listing 4-21: ProCurve Switch 2626 Bench mode "fs ls flash" command

```
ProCurve Switch 2626$ fs ls flash

Name              Size    Date
----------------  ------  -----------------
      .bootblock    1248  02/06/26 06:28:15
     mgrinfo.txt      96  01/01/90 00:00:21
      config.txt    6555  01/01/90 00:00:05
          iflags      50  01/01/90 00:00:14
          rbtcnt       4  01/01/90 00:00:05
```

The command "fs nvfswalk" shows the underlying file system structure that is used to store the config files shown in Output listing 4-21. Output listing 4-22 shows the first (active) file in the list called *.bootblock* and inactive (i.e. deleted) versions of the files called *rbtcnt* and *iflags*. The full output of the file system commands can be found in Output listing 6-9 in Appendix D. That listing shows the large number of old deleted files that can be found in a real decommissioned switch. A file is represented by an address block containing a start address, the file name, size, date, and flag field. Two stars in the "A" column seem to indicate which blocks are active (Output listing 4-22).

Output listing 4-22: Procurve "fs nvfswalk" output examples

```
fs nvfswalk

ProCurve Switch 2626$ fs nvfswalk
   A        addr          filename      size      date  flgs
  --  ----------  ----------------  --------  --------  ----
  **  0x7cf20000        .bootblock  000004e0  ffffffff  ffff
      0x7cf20500            rbtcnt  00000004  00000002  ffff
      0x7cf20530            rbtcnt  00000004  00000002  ffff
      0x7cf20560            iflags  00000032  00000006  ffff
      0x7cf205c0            rbtcnt  00000004  00000002  ffff

<snip>
```

The memory blocks are organized in a linked list structure with a 32 byte header followed by the data. Each header is aligned to an even 16byte boundary. In our case the first memory block is the active block named ".bootblock" at address 0x7cf20000. An example of a file header can be seen starting at address 0x7cf54690 in Output listing 4-26. Table 4-15 is my estimate of the memory blocks of header information prepending a data portion.

Table 4-15: ProCurve Switch 2626 file system header field structure prepending data

Fieldname	**Length [bytes]**	**Comment**
File name (null terminated)	16 (including NULL)	
Size (in bytes)	4	Size of the data field
Date	4	Unknown format
Address of next block	4	FF FF FF FF if no next block (I.e. this block is last in list)
Active/inactive flag	2	0x0000 mean inactive 0x00FF means active
Flags	2	Unknown use. Seems to always be 0xFFFF.
Data contents	According to length field	

Since flash can be programmed "1" to "0" a block can be flagged erased and a new block linked in by rewriting the next block pointer. This is my guess of what is happening when a file is "overwritten" with new contents.

1) The linked list is searched for the block which is active and has the file name. The active/inactive flag is programmed from 0x00FF to 0x0000.
2) A new block is created after the last current block. The new block is set to active, and with a next block pointer of 0xFFFFFFFF. The previous last block next pointer (0xFFFFFFFF) is reprogrammed to point to the newly created block.

Similar to the Cisco flash file system investigated in Section 4.1.3.7, a file can be flagged deleted without erasing the flash. The interesting question from a sanitization perspective is whether the old block data content is properly sanitized. This will be investigated in Section 4.2.6.

4.2.4 Flash memory address region

For forensic purposes it would be useful to extract the whole contents of the flash chip. The flash chip in this particular switch model is an 8MB AMD Am29LV065D. We know that the address 0x7cb80000 is inside the flash memory region (Output listing 4-16). 0x7cb80000 is on an even 8MB boundary so perhaps that would be the flash base address. Playing around with the read and write commands verified that assumption (Output listing 4-23):

- There is some old boot "ROM" code at the base address.
- Reading the last byte of the assumed flash region, 0x7cFFFFFF, works. However, reading the next byte causes a switch reset.
- Reading 256 bytes before the base address (0x7c7FFF00) works, but returns only 0xFF.

With the knowledge of the flash range, we can construct a tool to read out the entire flash (see Chapter 5). This can also be done by desoldering the chip itself and placing it in an external flash programmer.

Output listing 4-23 Investigation of the Procurve 2626 flash memory region

```
ProCurve Switch 2626# edomtset
ProCurve Switch 2626# edomtset
ProCurve Switch 2626$ sm -b -i
  access:b, display:b, read_length:256, inc addr after read
ProCurve Switch 2626$
ProCurve Switch 2626$
ProCurve Switch 2626$
ProCurve Switch 2626$
ProCurve Switch 2626$ sm -b -i
  access:b, display:b, read_length:256, inc addr after read
ProCurve Switch 2626$ read 0x7c800000
7c800000  40 28 23 29 20 48 50 20 4a 34 38 39 39 41 20 4a   @(#) HP J4899A J
7c800010  34 39 30 32 41 20 4a 34 39 30 30 41 20 20 20 20   4902A J4900A
7c800020  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
7c800030  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
7c800040  20 20 20 20 20 20 20 48 4d 44 4c 30 32 20 0d 0a          HMDL02 ..
7c800050  40 28 23 29 20 44 6f 77 6e 6c 6f 61 64 20 46 69   @(#) Download Fi
7c800060  6c 65 20 30 35 2f 31 39 2f 30 33 20 34 33 32 30   le 05/19/03 4320
7c800070  20 48 2e 30 37 2e 33 31 20 42 75 69 6c 64 23 20    H.07.31 Build#
7c800080  20 20 20 37 37 20 20 20 20 20 20 20 20 20 20 20      77
7c800090  20 20 78 78 78 78 78 2d 78 78 78 78 78 20 30 30     xxxxx-xxxxx 00
7c8000a0  30 31 20 30 30 30 33 20 0d 0a 40 28 23 29 20 43   01 0003 ..@(#) C
7c8000b0  6f 6d 70 61 74 69 62 6c 65 20 52 4f 4d 20 47 2e   ompatible ROM G.
7c8000c0  30 36 2e 58 58 20 20 20 20 30 30 20 0d 0a 1a 00   06.XX    00 ....
7c8000d0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
7c8000e0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
7c8000f0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 00                  .
ProCurve Switch 2626$ sm -l1 -b -i
  access:b, display:b, read_length:1, inc addr after read
ProCurve Switch 2626$ read 0x7cFFFFFF
7cffffff  ff                                                .
ProCurve Switch 2626$ sm -l256 -b -i
  access:b, display:b, read_length:256, inc addr after read
ProCurve Switch 2626$ read 0x7c7FFF00
7c7fff00  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff10  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff20  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff30  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff40  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff50  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff60  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff70  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff80  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fff90  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fffa0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fffb0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fffc0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fffd0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7fffe0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
7c7ffff0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
```

```
ProCurve Switch 2626$ sm -l1 -b -i
  access:b, display:b, read_length:1, inc addr after read
ProCurve Switch 2626$ read 0x7d000000
<causes switch reboot>
```

4.2.5 ProCurve Switch 2626 configuration interface

At boot time the user is presented with a command line interface (CLI). As with the Cisco CLI there are different levels of user privileges. The initial “Operator” privilege level can see, but not change the configuration. Entering the command “enable” (and a password if set) the user enters “Manager” privilege level in which they can enter configuration mode with the “config” command. A terminal GUI utility can be started with the “menu” command. This GUI allows management of a subset of the switch’s functions and parameters without requiring knowledge of the CLI syntax. [97]

4.2.6 Investigating ProCurve Switch 2626 sanitization completeness

The markers in Table 4-16 were set using the CLI configuration mode and saved with the command “write memory”. A crypto key was generate with the command “crypto key generate ssh rsa”. Where did the switch store these settings? Output listing 4-24 is the output of the “show config” command which reveals the hostname and SNMP password markers, but not the crypto key or the manager password.

Table 4-16: ProCurve Switch 2626 markers

Parameter to host marker	String marker (10 random chars)
SNMP password	MARKZlUXIlNsfl
Manager password	MARKnyghQTEhTy
Hostname	MARKCGsXAGvSiH

Output listing 4-24: show config, before erase

```
MARKCGsXAGvSiH$ show config

Startup configuration:

; J4900A Configuration Editor; Created on release #H.10.50

hostname "MARKCGsXAGvSiH"
snmp-server community "public" Unrestricted
snmp-server community "MARKZlUXIlNsfl" Operator
vlan 1
   name "DEFAULT_VLAN"
   untagged 1-26
   ip address dhcp-bootp
   exit
password manager
```

Output listing 4-25 shows the active files. Inspecting the address of each of them, we can see that the SNMP password and hostname ended up in the file called “delta”. It seems the file config.txt contains the default configuration and the changes are in the delta file (see Output listing 4-26).

Output listing 4-25: fs nvfswalk, before erase

```
MARKCGsXAGvSiH$ fs nvfswalk
   A        addr           filename      size      date  flgs
  --  ----------  ----------------  --------  --------  ----
  **  0x7cf20000        .bootblock  000004e0  ffffffff  ffff
<snip>
  **  0x7cf4d6d0            iflags  00000032  0000000e  ffff
<snip>
  **  0x7cf50180            rbtcnt  00000004  00000005  ffff
<snip>
  **  0x7cf52a50        config.txt  0000199b  00000005  ffff
  **  0x7cf54410       mgrinfo.txt  00000060  000001e3  ffff
  **  0x7cf54490         host_ssh1  000001d4  000001e8  ffff
  **  0x7cf54690             delta  00002800  000001ee  ffff

MARKCGsXAGvSiH$
```

Output listing 4-26: delta file, before erase

```
MARKCGsXAGvSiH$ read 0x7cf54690
7cf54690  64 65 6c 74 61 00 ff ff ff ff ff ff ff ff ff ff   delta..........
7cf546a0  00 00 28 00 00 00 01 ee ff ff ff ff 00 ff ff ff   ..(............
7cf546b0  02 00 04 00 16 4e 41 4d 45 3d 7e 4d 41 52 4b 43   .....NAME=~MARKC
7cf546c0  47 73 58 41 47 76 53 69 48 7e 0a 00 01 ed 00 01   GsXAGvSiH~......
7cf546d0  0a 00 01 ed 00 0d 53 4e 4d 50 4e 4f 54 49 46 59   ......SNMPNOTIFY
7cf546e0  20 28 0a 00 01 ed 00 0d 52 4f 57 5f 53 54 41 54    (......ROW_STAT
7cf546f0  55 53 3d 31 0a 00 01 ed 00 12 4e 41 4d 45 3d 7e   US=1......NAME=~
7cf54700  73 74 61 63 6b 74 72 61 70 73 7e 0a 00 01 ed 00   stacktraps~.....
7cf54710  0e 54 41 47 3d 73 74 61 63 6b 74 72 61 70 0a 00   .TAG=stacktrap..
7cf54720  01 ed 00 0e 53 54 4f 52 41 47 45 54 59 50 45 3d   ....STORAGETYPE=
7cf54730  35 0a 00 01 ef 00 12 53 4e 4d 50 56 33 43 4f 4d   5......SNMPV3COM
7cf54740  4d 55 4e 49 54 59 20 28 0a 00 01 ef 00 0d 52 4f   MUNITY (......RO
7cf54750  57 5f 53 54 41 54 55 53 3d 31 0a 00 01 ef 00 09   W_STATUS=1......
7cf54760  4e 41 4d 45 3d 7e 31 7e 0a 00 01 ef 00 13 43 4f   NAME=~1~......CO
7cf54770  4d 4d 5f 4e 41 4d 45 3d 7e 70 75 62 6c 69 63 7e   MM_NAME=~public~
7cf54780  0a 00 01 ef 00 25 53 45 43 5f 4e 41 4d 45 3d 7e   .....%SEC_NAME=~
MARKCGsXAGvSiH$ read
7cf54790  43 6f 6d 6d 75 6e 69 74 79 4d 61 6e 61 67 65 72   CommunityManager
7cf547a0  52 65 61 64 57 72 69 74 65 7e 0a 00 01 ef 00 0e   ReadWrite~......
7cf547b0  53 54 4f 52 41 47 45 54 59 50 45 3d 32 0a 00 01   STORAGETYPE=2...
7cf547c0  ef 00 02 29 0a 00 01 ef 00 01 0a 00 01 ef 00 12   ...)...........
7cf547d0  53 4e 4d 50 56 33 43 4f 4d 4d 55 4e 49 54 59 20   SNMPV3COMMUNITY
7cf547e0  28 0a 00 01 ef 00 0d 52 4f 57 5f 53 54 41 54 55   (......ROW_STATU
7cf547f0  53 3d 31 0a 00 01 ef 00 09 4e 41 4d 45 3d 7e 32   S=1......NAME=~2
7cf54800  7e 0a 00 01 ef 00 1b 43 4f 4d 4d 5f 4e 41 4d 45   ~......COMM_NAME
7cf54810  3d 7e 4d 41 52 4b 5a 6c 55 58 49 6c 4e 73 66 6c   =~MARKZlUXIlNsfl
7cf54820  7e 0a 00 01 ef 00 25 53 45 43 5f 4e 41 4d 45 3d   ~.....%SEC_NAME=
7cf54830  7e 43 6f 6d 6d 75 6e 69 74 79 4f 70 65 72 61 74   ~CommunityOperat
7cf54840  6f 72 52 65 61 64 4f 6e 6c 79 7e 0a 00 01 ef 00   orReadOnly~.....
7cf54850  0e 53 54 4f 52 41 47 45 54 59 50 45 3d 32 0a 00   .STORAGETYPE=2..
7cf54860  01 ef 00 02 29 0a 00 01 ef 00 02 29 0a 00 01 ef   ....)......)....
7cf54870  00 01 0a 00 01 f5 00 08 53 4e 4d 50 53 20 28 0a   ........SNMPS (.
7cf54880  00 01 f5 00 0d 52 4f 57 5f 53 54 41 54 55 53 3d   .....ROW_STATUS=
MARKCGsXAGvSiH$ read
7cf54890  31 0a 00 01 f5 00 09 43 4f 4d 5f 49 44 3d 32 0a   1......COM_ID=2.
7cf548a0  00 01 f5 00 16 4e 41 4d 45 3d 7e 4d 41 52 4b 5a   .....NAME=~MARKZ
7cf548b0  6c 55 58 49 6c 4e 73 66 6c 7e 0a 00 01 f5 00 07   lUXIlNsfl~......
7cf548c0  56 49 45 57 3d 33 0a 00 01 f5 00 02 29 0a 00 01   VIEW=3......)...
7cf548d0  f5 00 01 0a 00 03 3e 00 0d 44 48 43 50 53 4e 4f   ......>..DHCPSNO
7cf548e0  4f 50 72 20 28 0a 00 03 3e 00 0d 52 4f 57 5f 53   OPr (...>..ROW_S
7cf548f0  54 41 54 55 53 3d 33 0a 00 03 3e 00 02 29 0a 00   TATUS=3...>..)..
7cf54900  03 3e 00 01 0a fe ff ff ff ff ff ff ff ff ff ff   .>..............
7cf54910  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
```

The manager password ended up in the file called mgrinfo.txt, as shown in Output listing 4-27.

Output listing 4-27: mgrinfo.txt, before erase

```
ARKCGsXAGvSiH$ read 0x7cf54410
7cf54410  6d 67 72 69 6e 66 6f 2e 74 78 74 00 ff ff ff ff   mgrinfo.txt.....
7cf54420  00 00 00 60 00 00 01 e3 7c f5 44 90 00 ff ff ff   ...`....|.D.....
7cf54430  00 00 00 01 46 4c 47 00 4d 41 52 4b 6e 79 67 68   ....FLG.MARKnygh
7cf54440  51 54 45 68 54 79 00 00 00 00 00 00 00 00 00 00   QTEhTy..........
7cf54450  00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff fb   ................
7cf54460  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
7cf54470  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
7cf54480  00 00 00 00 00 00 00 00 00 00 00 02 06 00 00 00   ................
```

The ssh key ended up in the file called host_ssh1, as shown in Output listing 4-28.

Output listing 4-28: host_ssh1 file, before erase

```
7cf54490  68 6f 73 74 5f 73 73 68 31 00 ff ff ff ff ff ff   host_ssh1.......
7cf544a0  00 00 01 d4 00 00 01 e8 7c f5 46 90 00 ff ff ff   ........|.F.....
7cf544b0  53 53 48 20 50 52 49 56 41 54 45 20 4b 45 59 20   SSH PRIVATE KEY 
7cf544c0  46 49 4c 45 20 46 4f 52 4d 41 54 20 31 2e 31 0a   FILE FORMAT 1.1.
7cf544d0  00 00 00 00 00 00 00 00 03 80 03 80 c2 e0 89 c1   ................
7cf544e0  dd f6 ec c1 af 5d 75 47 63 b2 29 f0 0f 94 7f 6c   .....]uGc.)...l
7cf544f0  b7 76 fe 7d c0 d3 62 2d 13 6b a1 33 b2 b1 a1 12   .v.}..b-.k.3....
7cf54500  0f 2e 9c b0 f0 a8 45 a7 00 2c 5f 18 7d 19 c0 a4   ......E..,_.}...
7cf54510  93 a3 17 76 d3 b9 d5 c4 df 67 93 1e a7 ce ab 86   ...v.....g......
7cf54520  27 36 d6 5b 54 08 b4 2e 0f 82 29 66 a2 1d 86 21   '6.[T.....)f...!
7cf54530  70 7e d1 58 68 90 f5 fd fa d5 f0 b3 42 b7 ba 81   p~.Xh.......B...
7cf54540  92 08 ab 2d 68 26 35 4a 77 3f 2a 1d 00 06 23 00   ...-h&5Jw?*...#.
7cf54550  00 00 09 68 6f 73 74 5f 73 73 68 31 ee 3a ee 3a   ...host_ssh1.:.:
7cf54560  03 7f 64 38 f6 63 b3 fb 55 30 68 d0 fa 7c 7c 6a   .d8.c..U0h..||j
7cf54570  41 74 25 45 0e 55 2b 27 41 0d 79 1c 41 1e 7f 04   At%E.U+'A.y.A..
7cf54580  27 04 a5 0a e5 1f 3b 02 07 70 f0 cb 91 89 16 08   '.....;..p......
```

The switch was powered down by removing power. After restoring power, the switch booted up again and asked for the manager password to enter the CLI enable mode and the hostname marker was shown in the prompt. At this point, the factory reset procedure referred to as HP_2626_BUTTON in Appendix A was performed (both front buttons push using two paper clips). The switch rebooted and did not present a password or the marker prompt, indicating that the factory reset seems to have done what it is intended to do. In addition, “show config” shows the absences of the markers.

Output listing 4-29: show config, after erase

```
ProCurve Switch 2626# show config

Startup configuration:

; J4900A Configuration Editor; Created on release #H.10.50

hostname "ProCurve Switch 2626"
snmp-server community "public" Unrestricted
vlan 1
   name "DEFAULT_VLAN"
   untagged 1-26
   ip address dhcp-bootp
   exit

ProCurve Switch 2626#
```

The command “fs nvfswalk” shows that the previous delta file memory block at address 0x7cf54690 has been marked inactive, indicated by the absence of the “**” in the A column. However, inspecting the actual memory contents shows that the hostname and SNMP markers are still present. This output of both commands is shown in Output listing 4-30.

Output listing 4-30: delta file data, after erase

```
ProCurve Switch 2626$ fs nvfswalk
   A         addr           filename       size      date   flgs
  --  ----------  ----------------  --------  --------  ----
<SNIP>
      0x7cf54690             delta  00002800  000001ee  ffff
      0x7cf56eb0            rbtcnt  00000004  00000005  ffff
      0x7cf56ee0       mgrinfo.txt  00000060  00000163  ffff
  **  0x7cf56f60            rbtcnt  00000004  00000005  ffff
  **  0x7cf56f90       mgrinfo.txt  00000060  00000005  ffff
      0x7cf57010             delta  00002800  00000005  ffff
  **  0x7cf59830        config.txt  0000199b  00000005  ffff

ProCurve Switch 2626$ read 0x7cf54690
7cf54690  64 65 6c 74 61 00 ff ff ff ff ff ff ff ff ff ff   delta...........
7cf546a0  00 00 28 00 00 00 01 ee 7c f5 6e b0 00 00 ff ff   ..(.....|.n.....
7cf546b0  02 00 04 00 16 4e 41 4d 45 3d 7e 4d 41 52 4b 43   .....NAME=~MARKC
7cf546c0  47 73 58 41 47 76 53 69 48 7e 0a 00 01 ed 00 01   GsXAGvSiH~......
7cf546d0  0a 00 01 ed 00 0d 53 4e 4d 50 4e 4f 54 49 46 59   ......SNMPNOTIFY
7cf546e0  20 28 0a 00 01 ed 00 0d 52 4f 57 5f 53 54 41 54    (......ROW_STAT
7cf546f0  55 53 3d 31 0a 00 01 ed 00 12 4e 41 4d 45 3d 7e   US=1......NAME=~
7cf54700  73 74 61 63 6b 74 72 61 70 73 7e 0a 00 01 ed 00   stacktraps~.....
7cf54710  0e 54 41 47 3d 73 74 61 63 6b 74 72 61 70 0a 00   .TAG=stacktrap..
7cf54720  01 ed 00 0e 53 54 4f 52 41 47 45 54 59 50 45 3d   ....STORAGETYPE=
7cf54730  35 0a 00 01 ef 00 12 53 4e 4d 50 56 33 43 4f 4d   5......SNMPV3COM
7cf54740  4d 55 4e 49 54 59 20 28 0a 00 01 ef 00 0d 52 4f   MUNITY (......RO
7cf54750  57 5f 53 54 41 54 55 53 3d 31 0a 00 01 ef 00 09   W_STATUS=1......
7cf54760  4e 41 4d 45 3d 7e 31 7e 0a 00 01 ef 00 13 43 4f   NAME=~1~......CO
7cf54770  4d 4d 5f 4e 41 4d 45 3d 7e 70 75 62 6c 69 63 7e   MM_NAME=~public~
7cf54780  0a 00 01 ef 00 25 53 45 43 5f 4e 41 4d 45 3d 7e   .....%SEC_NAME=~
ProCurve Switch 2626$ read
7cf54790  43 6f 6d 6d 75 6e 69 74 79 4d 61 6e 61 67 65 72   CommunityManager
7cf547a0  52 65 61 64 57 72 69 74 65 7e 0a 00 01 ef 00 0e   ReadWrite~......
7cf547b0  53 54 4f 52 41 47 45 54 59 50 45 3d 32 0a 00 01   STORAGETYPE=2...
7cf547c0  ef 00 02 29 0a 00 01 ef 00 01 0a 00 01 ef 00 12   ...)............
7cf547d0  53 4e 4d 50 56 33 43 4f 4d 4d 55 4e 49 54 59 20   SNMPV3COMMUNITY
7cf547e0  28 0a 00 01 ef 00 0d 52 4f 57 5f 53 54 41 54 55   (......ROW_STATU
7cf547f0  53 3d 31 0a 00 01 ef 00 09 4e 41 4d 45 3d 7e 32   S=1......NAME=~2
7cf54800  7e 0a 00 01 ef 00 1b 43 4f 4d 4d 5f 4e 41 4d 45   ~......COMM_NAME
7cf54810  3d 7e 4d 41 52 4b 5a 6c 55 58 49 6c 4e 73 66 6c   =~MARKZlUXIlNsfl
7cf54820  7e 0a 00 01 ef 00 25 53 45 43 5f 4e 41 4d 45 3d   ~.....%SEC_NAME=
7cf54830  7e 43 6f 6d 6d 75 6e 69 74 79 4f 70 65 72 61 74   ~CommunityOperat
7cf54840  6f 72 52 65 61 64 4f 6e 6c 79 7e 0a 00 01 ef 00   orReadOnly~.....
7cf54850  0e 53 54 4f 52 41 47 45 54 59 50 45 3d 32 0a 00   .STORAGETYPE=2..
7cf54860  01 ef 00 02 29 0a 00 01 ef 00 02 29 0a 00 01 ef   ....)......)....
7cf54870  00 01 0a 00 01 f5 00 08 53 4e 4d 50 53 20 28 0a   ........SNMPS (.
7cf54880  00 01 f5 00 0d 52 4f 57 5f 53 54 41 54 55 53 3d   .....ROW_STATUS=
ProCurve Switch 2626$
```

Similarly, the manager password marker can be found in the old mgrinfo.txt file block (which has also been flagged inactive). This is shown in Output listing 4-31.

Output listing 4-31: mgrinfo.txt data after erase

```
ProCurve Switch 2626$ fs nvfswalk
   A        addr           filename      size      date   flgs
  --  ----------  ----------------  --------  --------  ----
<SNIP>
      0x7cf54410       mgrinfo.txt  00000060  000001e3  ffff
  **  0x7cf54490         host_ssh1  000001d4  000001e8  ffff
      0x7cf54690             delta  00002800  000001ee  ffff
      0x7cf56eb0            rbtcnt  00000004  00000005  ffff
      0x7cf56ee0       mgrinfo.txt  00000060  00000163  ffff
  **  0x7cf56f60            rbtcnt  00000004  00000005  ffff
  **  0x7cf56f90       mgrinfo.txt  00000060  00000005  ffff
      0x7cf57010             delta  00002800  00000005  ffff
  **  0x7cf59830        config.txt  0000199b  00000005  ffff

ProCurve Switch 2626$ read 0x7cf54410
7cf54410  6d 67 72 69 6e 66 6f 2e 74 78 74 00 ff ff ff ff   mgrinfo.txt.....
7cf54420  00 00 00 60 00 00 01 e3 7c f5 44 90 00 00 ff ff   ...`....|.D.....
7cf54430  00 00 00 01 46 4c 47 00 4d 41 52 4b 6e 79 67 68   ....FLG.MARKnygh
7cf54440  51 54 45 68 54 79 00 00 00 00 00 00 00 00 00 00   QTEhTy..........
7cf54450  00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff fb   ................
7cf54460  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
7cf54470  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
7cf54480  00 00 00 00 00 00 00 00 00 00 00 02 06 00 00 00   ................
```

The private crypto key was originally stored in the file host_ssh1 at address 0x7cf54490. The factory reset routine did not mark this memory block inactive, so the contents of this file is completely intact and can even be listed using its original file name or read using memory inspection. Output listing 4-32 shows the file contents of host_ssh1 after the factory reset procedure.

Output listing 4-32: host_ssh1 file after erase

```
ProCurve Switch 2626$ fs od host_ssh1
00000000  53 53 48 20 50 52 49 56 41 54 45 20 4b 45 59 20  SSH PRIVATE KEY
00000010  46 49 4c 45 20 46 4f 52 4d 41 54 20 31 2e 31 0a  FILE FORMAT 1.1.
00000020  00 00 00 00 00 00 00 00 03 80 03 80 c2 e0 89 c1  ................
00000030  dd f6 ec c1 af 5d 75 47 63 b2 29 f0 0f 94 7f 6c  .....]uGc.)....l
00000040  b7 76 fe 7d c0 d3 62 2d 13 6b a1 33 b2 b1 a1 12  .v.}..b-.k.3....
00000050  0f 2e 9c b0 f0 a8 45 a7 00 2c 5f 18 7d 19 c0 a4  ......E..,_.}...
00000060  93 a3 17 76 d3 b9 d5 c4 df 67 93 1e a7 ce ab 86  ...v.....g......
00000070  27 36 d6 5b 54 08 b4 2e 0f 82 29 66 a2 1d 86 21  '6.[T.....)f...!
00000080  70 7e d1 58 68 90 f5 fd fa d5 f0 b3 42 b7 ba 81  p~.Xh.......B...
00000090  92 08 ab 2d 68 26 35 4a 77 3f 2a 1d 00 06 23 00  ...-h&5Jw?*...#.
<SNIP>
ProCurve Switch 2626$ read 0x7cf54490
7cf54490  68 6f 73 74 5f 73 73 68 31 00 ff ff ff ff ff ff   host_ssh1.......
7cf544a0  00 00 01 d4 00 00 01 e8 7c f5 46 90 00 ff ff ff   ........|.F.....
7cf544b0  53 53 48 20 50 52 49 56 41 54 45 20 4b 45 59 20   SSH PRIVATE KEY
7cf544c0  46 49 4c 45 20 46 4f 52 4d 41 54 20 31 2e 31 0a   FILE FORMAT 1.1.
7cf544d0  00 00 00 00 00 00 00 00 03 80 03 80 c2 e0 89 c1   ................
7cf544e0  dd f6 ec c1 af 5d 75 47 63 b2 29 f0 0f 94 7f 6c   .....]uGc.)...l
7cf544f0  b7 76 fe 7d c0 d3 62 2d 13 6b a1 33 b2 b1 a1 12   .v.}..b-.k.3....
7cf54500  0f 2e 9c b0 f0 a8 45 a7 00 2c 5f 18 7d 19 c0 a4   ......E..,_.}...
7cf54510  93 a3 17 76 d3 b9 d5 c4 df 67 93 1e a7 ce ab 86   ...v.....g......
7cf54520  27 36 d6 5b 54 08 b4 2e 0f 82 29 66 a2 1d 86 21   '6.[T.....)f...!
7cf54530  70 7e d1 58 68 90 f5 fd fa d5 f0 b3 42 b7 ba 81   p~.Xh.......B...
7cf54540  92 08 ab 2d 68 26 35 4a 77 3f 2a 1d 00 06 23 00   ...-h&5Jw?*...#.
ProCurve Switch 2626$
```

As we found all 3 markers and the crypto key after performing the HP_2626_BUTTON procedure, it is apparent that the factory reset procedure did not actually sanitize the switch. Perhaps we will be more successful sanitizing the switch with a second procedure. The “erase startup-config” command was issued via the CLI according to the erase procedure HP_2626_CLI in Appendix A and the switch was rebooted. Unfortunately, this procedure produces the same results, i.e., all 3 markers and the crypto key were still present in their original memory locations.

4.2.6.1 Summary of sanitization using vendor factory reset commands.

The overall assessment of the vendor erase routines for the HP ProCurve Switch 2626 (Part number J4900A) running Firmware: H.10.50 from Oct 9 2007 is summarized in Table 4-17. My investigation of this switch showed that these procedures do **not** actually overwrite the sensitive data. Not only is the last copy of the configuration stored, as the new delta configurations are written to new flash regions, but the complete history of old config data may be present in the flash memory. Perhaps the idea of utilizing delta files was to provide wear leveling. However, as the flash chip datasheet [98] guarantees 1 million flash erase cycles and the flash region has to be erased before storing something new, the old flash memory location could be immediately erased after creating a new configuration. Note that 1 million flash cycles would correspond to 50 years of operation at 50 changes in the configuration per day.

Table 4-17: Result of sanitization investigation of HP ProCurve Switch 2626 (Part number J4900A)

Parameter to host marker	Parameter storage	Storage size to search	Result	Evidence Strength	Security severity assessment
SNMP password	Flash	8MB	Not sanitized by either procedure HP_2626_BUTTON or HP_2626_CLI	10 chars in 8MB = $1 - 6*10^{-11}$	Severe
Hostname	Flash	8MB	Not sanitized by either procedure HP_2626_BUTTON or HP_2626_CLI	$1 - 6*10^{-11}$	Severe
Manager password	Flash	8MB	Not sanitized by either procedure HP_2626_BUTTON or HP_2626_CLI	$1 - 6*10^{-11}$	Severe

4.2.6.2 *A method to sanitize the switch configuration*

It would be an easy task to write 0x00 to all bytes in the block at the same time it is flagged as inactive. In fact we can perform this write ourselves using the memory manipulation routines the vendor supplies. According to the datasheet flash programming is done by first writing the sequence 0xAA, 0x55, 0xA0 to any address. This is followed by writing the actual data to the address we want to program. Flash can only be programmed by changing “ones to zeros”, but filling the relevant portions of the memory with 0x00 is sufficient to sanitize the old configuration regions.

We can use the algorithm in Table 4-18 to erase inactive flash file regions. Output listing 4-33 shows how a byte of the manager password is cleared using the flash program algorithm. The header of each file flash region might need to be left intact to avoid destroying the file system’s state. Further details of this approach will be left for future work.

Table 4-18: Procurve 2626 configuration sanitization algorithm

```
FOR EACH ROW R IN “fs nvfswalk”
  If R.A is not “**”               ; not active
    For (x = R.addr; x++; x<R.addr+ R.size
      wr x 0xAA
      wr x 0x55
      wr x 0xA0
      wr x 0x00
```

Output listing 4-33: Procurve 2626, proof of concept, byte erase in flash

```
ProCurve Switch 2626$ sm -b -i -l64
  access:b, display:b, read_length:64, inc addr after read
ProCurve Switch 2626$ read 7cf54438
7cf54438  4d 41 52 4b 6e 79 67 68 51 54 45 68 54 79 00 00   MARKnyghQTEhTy..
7cf54448  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
7cf54458  00 00 00 00 ff ff ff fb 00 00 00 00 00 00 00 00   ................
7cf54468  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
ProCurve Switch 2626$ wr 0x7cf54438 0xAA
ProCurve Switch 2626$ wr 0x7cf54438 0x55
ProCurve Switch 2626$ wr 0x7cf54438 0xA0
ProCurve Switch 2626$ wr 0x7cf54438 0x00
ProCurve Switch 2626$ read 0x7cf54438
7cf54438  00 41 52 4b 6e 79 67 68 51 54 45 68 54 79 00 00   .ARKnyghQTEhTy..
7cf54448  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
7cf54458  00 00 00 00 ff ff ff fb 00 00 00 00 00 00 00 00   ................
7cf54468  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
ProCurve Switch 2626$
```

4.3 Sanitization investigation of a HP ProCurve Switch 2824

Perhaps the poor sanitization result using the vendor commands for the Procurve 2626 was due to some unlucky combination of old software and hardware. In this section we investigate a HP Procurve 2824 (Part# J4903A) with software version I.10.105 with a build date of April 2014.

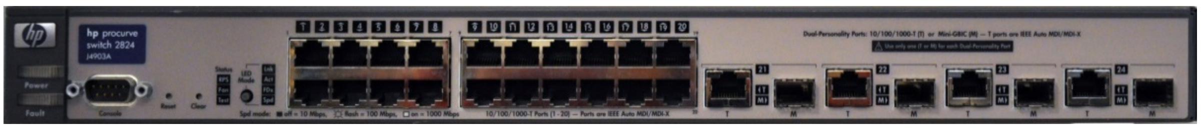


Figure 4-16: Front of ProCurve Switch 2824 (Part# J4903A)

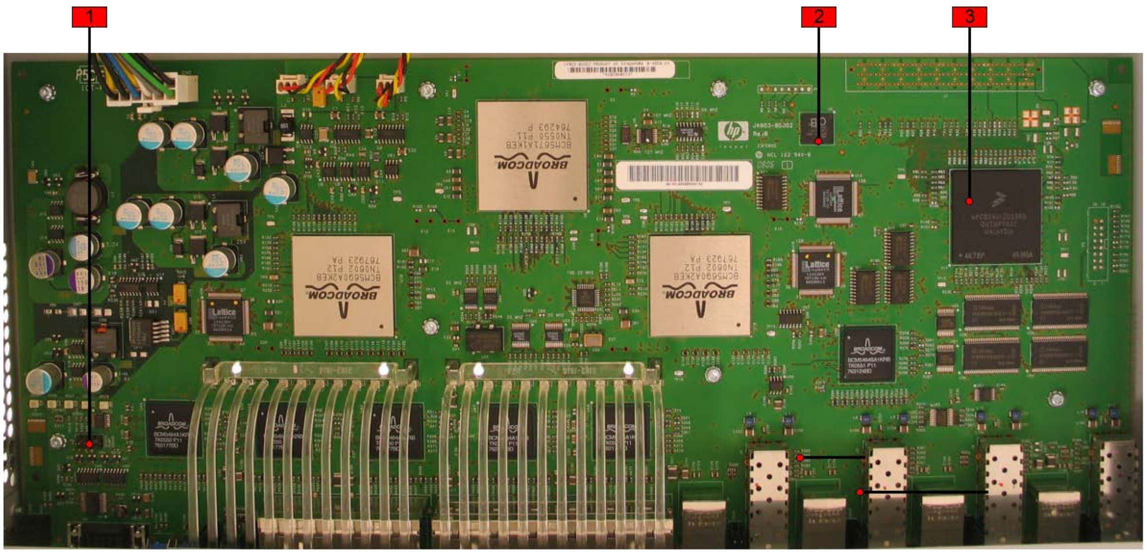


Figure 4-17: Main board of ProCurve Switch 2824 (Part# J4903A)

Table 4-19: Interesting components of the Procurve 2824 mainboard

Object Number	Description
1	2 x 5 pin header labeled P6 with pairs of header pins labeled Debug, Bench and Dump
2	U2 AMD Flash chip labeled L065MU12RI and stamped 0B. Seems to be a 8MB flash Am29LV065MU which are now called S29GL064A [99]
3	MPC8245LZU226 CPU

4.3.1 Investigating ProCurve Switch 2824 sanitization completeness

The same procedure as applied to the Procurve 2626 case was used to inject several markers and generate a crypto key. The markers used are shown in Table 4-20.

Table 4-20: ProCurve Switch 2824 markers

Parameter to host marker	String marker (10 random chars)
SNMP password	MARKwBEKowbKRB
Manager password	MARKcGsXAGVcIG
Hostname	MARKtBHqwwEqpC

The latest manual listed on the vendor's support page [100] for this switch is dated October 2005. This manual outlines the same two procedures for factory reset as used for the Procurve 2626 switch [101 Page C-43]. As such, we can apply the same two procedures, called HP_2626_CLI and HP_2626_BUTTON in Appendix A.

The HP_2626_BUTTON procedure was performed followed by the HP_2626_CLI procedure. After rebooting and trying to enter diagnostic mode a new legal message was displayed (shown in Output listing 4-34). It seems to be difficult to use this mode on this switch and to include any outputs in my thesis without upsetting the vendors' legal department. In fact, the vendor by this message also forbids end users using the "wr" command to properly sanitize the switch's memory themselves.

Output listing 4-34: ProCurve Switch 2824 diagnostic mode legal message

```
ProCurve Switch 2824# edomtset
ProCurve Switch 2824# edomtset
ATTENTION: You are entering a diagnostic mode on this product that is HP
Confidential and Proprietary. This mode, the commands and functionality
specific to this mode, and all output from this mode are HP Confidential
and Proprietary. You may use this mode only by specific permission of, and
under the direction of, an HP support engineer or HP technical engineer.
Unauthorized or improper use of this mode will be considered by HP to be
unauthorized modification of the product, and any resulting defects or
issues are not eligible for coverage under the HP product warranty or any
HP support or service.  UNAUTHORIZED OR IMPROPER USE OF THIS MODE CAN
MAKE THE PRODUCT COMPLETELY INOPERABLE.
```

However, the vendor legal department forgot to add the message when booting with the "Bench" jumper so I hope that I can still show (without risk of being sentenced to jail) that all of the markers are still present after factory reset on this switch. I better whisper it: It's in Output listing 6-10 in 0.

4.4 Sanitization investigation of a ProCurve Switch 2610-48

While working with the development of the sanitty utility (chapter 5) I discovered that a ProCurve Switch 2610-48 (J9088A, Software revision R.11.54) had three new commands in benchtest/edomtset mode:

Output listing 4-35: ProCurve 2610-48 new commands

```
nvfserase             Erase all files in filesystem
nvfsfill              Fill up the filesystem
nvfsdir               Display nvfs directory contents
```

After testing them, this is my interpretation of what they do (See Output listing 6-11 in Appendix F):

- *Nvfserase* erases the nvfs portion of the flash (0xff), then initializes it with the first .bootblock file node record to start the linked node list.
- *Nvfsfill* fills up the empty space of the nvfs area with a file "fill000". The data contents is repeating 0x00 0x01 ... 0xff pattern. The command seems to do some tests as well, perhaps to find bad blocks.
- *Nvfsdir* Displays a node listing similar to "fs nvfswalk", but with some extra information.

I repeated the same sanitization investigations method of the previous switches but also use the *nvfserase* command. The markers in Table 4-21 were set. After that, the system was rebooted. The password prompt was confirmed. Erase procedure HP_2626_BUTTON from Appendix A was performed. This let us in without using password and our newly developed Sanitty tool (command *dumpnvfs)* was used to read out the inactive nodes. All the markers could be found (see Output listing 6-12, Appendix F). Same result after following up with the HP_2626_CLI procedure. But after the *nvfserase* command (followed by a reboot) all memory locations previously holding the markers were erased (now 0xff). Since this investigation was done after the creation of the Sanitty tool I also dumped the entire 16MB flash (using the Sanitty *dumpflash* command). The markers could not be found in any of the flash data.

So apparently this switch has a command to sanitize the whole nv fs area. Although it is hidden and undocumented. The recommended erase routines still both leave the old config revisions intact in flash.

Table 4-21: ProCurve Switch 2610-48 markers

Parameter to host marker	String marker (10 random chars)
SNMP password	MARKwBEKowbKRB
Manager password	MARKcGsXAGVcIG
Hostname	MARKtBHqwwEqpC

4.5 HP Procurve physical access security

The document “Hardening ProCurve Switches” [102] suggests two commands to “fully secure the device” and disable the factory reset using the front buttons:

ProCurve Switch(config)# **no front-panel-security factory-reset**

ProCurve Switch(config)# **no front-panel-security password-clear**

If a switch has a password set, would these commands make it impossible to disable the password using the HP_2626_BUTTON “erase” procedure in Appendix A? This would effectively prevent us from entering the CLI and edomtset mode, hence making it difficult to sanitize the device. Additionally, from a security perspective it would prevent access to the sensitive data unless the flash is physically removed and read in an external programmer.

The vendor’s document “Configuring Username and Password Security” addresses the concern for physical access security when the device is installed in a non-secure location [103 pp. 2–9].

Let us test how this works. A HP Procurve J9088A Switch 2610-48 was prepared with the most recent software revision (R.11.54) and three markers were generated and injected as hostname, SNMP password, and manager password according to Table 4-22 and Output listing 4-36. The switch was then restarted.

The two commands (shown above) worked as intended: The HP_2626_BUTTON “erase” procedure involving the *reset* and *clear* buttons on the front would no longer let us bypass the password, hence we could no longer enter into the CLI.

However, the physical security was only limited to the front buttons. The *bench* jumper on the logic board still worked and made the switch boot into the bench mode without asking for password. From there it was possible to extract all the markers as shown in Output listing 4-37.

So the physical access security commands *do* offer security by preventing a user with front panel access to push the buttons and stop the operation of the switch. However, these do *not* protect any sensitive data from being accessed from the flash by someone who has can open the switch up (a process which takes less than a minute). Via the *bench* jumper all current and possibly the long history of previous configurations and passwords are exposed, the later occurs due to the fact that old information is kept intact in the flash.

For future releases, the vendor *could* implement disable of the bench jumper functionaility when the front-panel-security is activated. However, from a refurbisher’s perspective it is annoying to have to bypass strong password security. Furthermore, the data could probably still be read from the flash if were to be desoldered or read via JTAG. Therefore, my advice to the vendor would be to *allow* the use of the bench jumper, but to halt the boot process and display a message such as that shown below. If the user enters “y”, then the configuration area of the flash should be securely erased.

```
This device has front-panel-security activated. Continuing will sanitize (erase)
the configuration area of the flash. Do you want to continue [y/n]?
```

Table 4-22: Procurve Switch 2610-48 physical security test martkers

Parameter to host marker	String marker (10 random chars)
SNMP password	MARKtiLrWdJSZJ
Manager password	MARKBOOrEjmshO
Hostname	MARKgScepSuzMT

Output listing 4-36: Procurve front panel security preparation

```
ProCurve Switch 2610-48(config)# hostname MARKgScepSuzMT
MARKgScepSuzMT(config)# snmp-server community MARKtiLrWdJSZJ
MARKgScepSuzMT(config)# password manager
New password for manager: **************
Please retype new password for manager: **************

DHCP based config file download from a TFTP server is disabled when an operator
or manager password is set.
MARKgScepSuzMT(config)# no front-panel-security factory-reset
                              **** CAUTION ****
Disabling the factory reset option prevents switch configuation and passwords
from being easily reset or recovered.  Ensure that you are familiar with the
front panel security options before proceeding.

Continue with disabling the factory reset option[y/n]? y
MARKgScepSuzMT(config)# no front-panel-security password-clear
                              **** CAUTION ****
Disabling the clear button prevents switch passwords from being easily reset or
recovered.  Ensure that you are familiar with the front panel security options
before proceeding.

Continue with disabling the clear button [y/n]? y
MARKgScepSuzMT(config)# crypto key generate ssh rsa
Installing new key pair.  If the key/entropy cache is
depleted, this could take several minutes.
MARKgScepSuzMT(config)# write memory
```

Output listing 4-37: **ProCurve Switch 2610-48 markers found in bench mode (red highlight)**

```
MARKgScepSuzMT=> read 0xbcf96710
bcf96710  64 65 6c 74 61 00 ff ff ff ff ff ff ff ff ff ff   delta...........
bcf96720  00 00 28 00 00 00 00 1f bc f9 8f 30 00 ff ff ff   ..(........0....
bcf96730  00 02 89 00 01 0a 00 02 89 00 0d 53 4e 4d 50 4e   ...........SNMPN
bcf96740  4f 54 49 46 59 20 28 0a 00 02 89 00 0d 52 4f 57   OTIFY (......ROW
bcf96750  5f 53 54 41 54 55 53 3d 31 0a 00 02 89 00 12 4e   _STATUS=1......N
bcf96760  41 4d 45 3d 7e 73 74 61 63 6b 74 72 61 70 73 7e   AME=~stacktraps~
bcf96770  0a 00 02 89 00 0e 54 41 47 3d 73 74 61 63 6b 74   ......TAG=stackt
bcf96780  72 61 70 0a 00 02 89 00 0e 53 54 4f 52 41 47 45   rap......STORAGE
bcf96790  54 59 50 45 3d 35 0a 00 02 8b 00 12 53 4e 4d 50   TYPE=5......SNMP
bcf967a0  56 33 43 4f 4d 4d 55 4e 49 54 59 20 28 0a 00 02   V3COMMUNITY (...
bcf967b0  8b 00 0d 52 4f 57 5f 53 54 41 54 55 53 3d 31 0a   ...ROW_STATUS=1.
bcf967c0  00 02 8b 00 09 4e 41 4d 45 3d 7e 31 7e 0a 00 02   .....NAME=~1~...
bcf967d0  8b 00 13 43 4f 4d 4d 5f 4e 41 4d 45 3d 7e 70 75   ...COMM_NAME=~pu
bcf967e0  62 6c 69 63 7e 0a 00 02 8b 00 25 53 45 43 5f 4e   blic~.....%SEC_N
bcf967f0  41 4d 45 3d 7e 43 6f 6d 6d 75 6e 69 74 79 4d 61   AME=~CommunityMa
bcf96800  6e 61 67 65 72 52 65 61 64 57 72 69 74 65 7e 0a   nagerReadWrite~.

MARKgScepSuzMT=>  read
bcf96810  00 02 8b 00 0e 53 54 4f 52 41 47 45 54 59 50 45   .....STORAGETYPE
bcf96820  3d 32 0a 00 02 8b 00 02 29 0a 00 02 8b 00 02 29   =2......)......)
bcf96830  0a 00 02 8b 00 01 0a 00 03 da 00 0d 44 48 43 50   ............DHCP
bcf96840  53 4e 4f 4f 50 72 20 28 0a 00 03 da 00 0d 52 4f   SNOOPr (......RO
bcf96850  57 5f 53 54 41 54 55 53 3d 33 0a 00 03 da 00 02   W_STATUS=3......
bcf96860  29 0a 00 03 da 00 01 0a fe 02 00 01 00 3c 3b 20   ).............<;
bcf96870  4a 39 30 38 38 41 20 43 6f 6e 66 69 67 75 72 61   J9088A Configura
bcf96880  74 69 6f 6e 20 45 64 69 74 6f 72 3b 20 43 72 65   tion Editor; Cre
bcf96890  61 74 65 64 20 6f 6e 20 72 65 6c 65 61 73 65 20   ated on release
bcf968a0  23 52 2e 31 31 2e 31 30 37 0a 01 00 02 00 02 fe   #R.11.107.......
bcf968b0  00 00 02 00 01 0a 00 02 45 00 01 0a 00 02 45 00   ........E.....E.
bcf968c0  10 53 59 53 4c 4f 47 5f 4e 4f 54 49 46 59 20 28   .SYSLOG_NOTIFY (
bcf968d0  0a 00 02 45 00 13 53 59 53 4c 4f 47 5f 4e 4f 54   ...E..SYSLOG_NOT
bcf968e0  49 46 59 5f 49 44 3d 31 0a 00 02 45 00 02 29 0a   IFY_ID=1...E..).
bcf968f0  fe 02 00 04 00 16 4e 41 4d 45 3d 7e 4d 41 52 4b   ......NAME=~MARK
bcf96900  67 53 63 65 70 53 75 7a 4d 54 7e 0a 00 00 05 00   gScepSuzMT~.....

MARKgScepSuzMT=> read
bcf96910  15 43 4f 4e 46 49 47 5f 46 49 4c 45 5f 55 50 44   .CONFIG_FILE_UPD
bcf96920  41 54 45 3d 32 0a 00 02 9c 00 01 0a 00 02 9c 00   ATE=2...........
bcf96930  12 53 4e 4d 50 56 33 43 4f 4d 4d 55 4e 49 54 59   .SNMPV3COMMUNITY
bcf96940  20 28 0a 00 02 9c 00 0d 52 4f 57 5f 53 54 41 54    (......ROW_STAT
bcf96950  55 53 3d 31 0a 00 02 9c 00 09 4e 41 4d 45 3d 7e   US=1......NAME=~
bcf96960  32 7e 0a 00 02 9c 00 1b 43 4f 4d 4d 5f 4e 41 4d   2~......COMM_NAM
bcf96970  45 3d 7e 4d 41 52 4b 74 69 4c 72 57 64 4a 53 5a   E=~MARKtiLrWdJSZ
bcf96980  4a 7e 0a 00 02 9c 00 25 53 45 43 5f 4e 41 4d 45   J~.....%SEC_NAME
bcf96990  3d 7e 43 6f 6d 6d 75 6e 69 74 79 4f 70 65 72 61   =~CommunityOpera
bcf969a0  74 6f 72 52 65 61 64 4f 6e 6c 79 7e 0a 00 02 9c   torReadOnly~....
bcf969b0  00 0e 53 54 4f 52 41 47 45 54 59 50 45 3d 32 0a   ..STORAGETYPE=2.
bcf969c0  00 02 9c 00 02 29 0a 00 02 a4 00 08 53 4e 4d 50   .....)......SNMP
bcf969d0  53 20 28 0a 00 02 a4 00 0d 52 4f 57 5f 53 54 41   S (......ROW_STA
bcf969e0  54 55 53 3d 31 0a 00 02 a4 00 09 43 4f 4d 5f 49   TUS=1......COM_I
bcf969f0  44 3d 32 0a 00 02 a4 00 16 4e 41 4d 45 3d 7e 4d   D=2......NAME=~M
bcf96a00  41 52 4b 74 69 4c 72 57 64 4a 53 5a 4a 7e 0a 00   ARKtiLrWdJSZJ~..

MARKgScepSuzMT=> read 0xbcf98f90
bcf98f90  6d 67 72 69 6e 66 6f 2e 74 78 74 00 ff ff ff ff   mgrinfo.txt.....
bcf98fa0  00 00 00 60 00 00 15 8c bc f9 90 10 00 00 ff ff   ...`............
bcf98fb0  00 00 00 01 46 4c 47 00 4d 41 52 4b 42 4f 4f 72   ....FLG.MARKBOOr
bcf98fc0  45 6a 6d 73 68 4f 00 00 00 00 00 00 00 00 00 00   EjmshO..........
bcf98fd0  00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff   ................
bcf98fe0  6d 61 6e 61 67 65 72 00 00 00 00 00 00 00 00 00   manager.........
bcf98ff0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
bcf99000  00 00 00 00 00 00 00 00 00 00 00 02 06 00 00 00   ................
bcf99010  6d 67 72 69 6e 66 6f 2e 74 78 74 00 ff ff ff ff   mgrinfo.txt.....
bcf99020  00 00 00 60 00 00 15 a0 bc f9 90 90 00 00 ff ff   ...`............
bcf99030  00 00 00 01 46 4c 47 00 4d 41 52 4b 42 4f 4f 72   ....FLG.MARKBOOr
bcf99040  45 6a 6d 73 68 4f 00 00 00 00 00 00 00 00 00 00   EjmshO..........
bcf99050  00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff   ................
bcf99060  6d 61 6e 61 67 65 72 00 00 00 00 00 00 00 00 00   manager.........
bcf99070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
bcf99080  00 00 00 00 00 00 00 01 00 00 00 02 06 00 00 00   ................
```

4.6 Summary of vendor sanitization routines for the HP Procurve switches

I have investigated the sanitization completeness of the three HP Procurve switches listed in Table 4-23. None of these switches were properly sanitized using the vendor's proposed method (from their Management and Configuration Guide). The J9088A was able to be sanitized using a hidden and undocumented command. Investigating additional models of HP Procurve switches is left to future work.

Table 4-23: Summary of Procurve switches tested

Part#	**Model name**	**Firmware**
J4900A	2626	H.10.50 Build date Oct 9 2007
J4903A	2824	I.10.105 Build date April 2014
J9088A	2610-48	R.11.54

5 Sanitty – Making a sanitizer utility for Procurve switches

As a result of the poor sanitization results on the Procurve switches using the vendor's proposed methods from the vendor's public documentation and even hidden commands, I decided to write my own sanitization utility: *sanitty*. It communicates over the serial port and uses the Procurve's hidden *wr* and *read* edomtset CLI commands to read and write to memory. The tool is primarily for HP's Procurve switches, but is written in a modular form so adding similar functionality by using the Cisco Rommon mode as a backend should be easy.

I considered two different software development approaches: A big beautiful Java project and a small C hack. In the end, I chose to develop A big ugly C hack.

The Java project had three benefits:

Truly object oriented	Inheritance is useful to model the different devices to sanitize. A Procurve J9088A switch can inherit from a generic Procurve switch class and only override device specific functionality.
Robust string and buffer handling	No magic numbers in buffer memory allocation.
Easy GUI handling	User interface components such as buttons, dialogs, list and even a hexeditor and interactive terminal input/output window could be added easily.

However, the Java project had one problem: RS232 connectivity is not supported by default. As a result, the user would have to install a third party RS232 library separately to use the tool or use a TCP/IP serial port adapter. For these reasons, I decided to develop a small C program using Visual Studio Express 2013 for Windows. Visual Studio Express is a free Integrated Development Environment (IDE) with editor, debugger, and compiler.

5.1 Software layers

This section briefly describes the different components of Sanitty. Figure 5-1 shows the software and hardware stack.

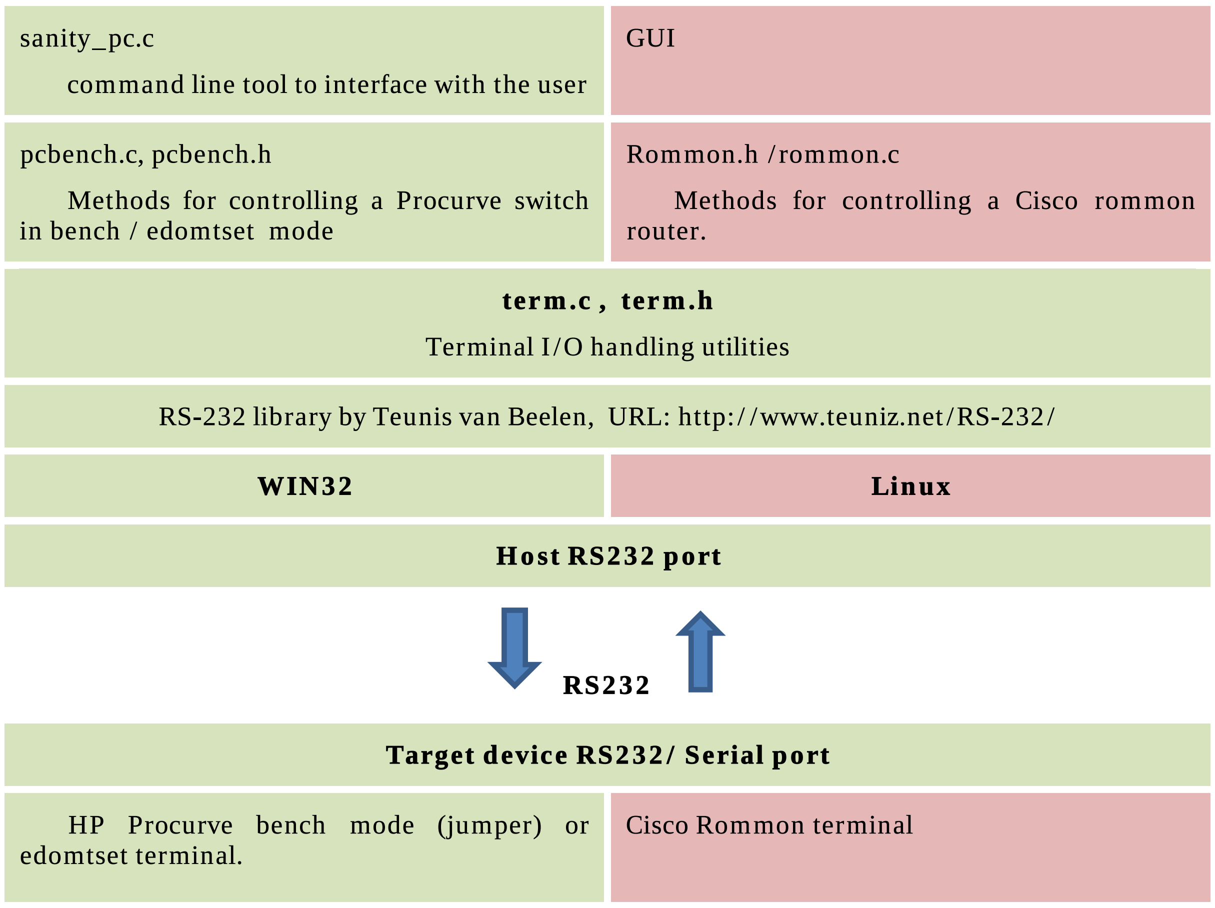

sanity_pc.c
command line tool to interface with the user
GUI
pcbench.c, pcbench.h
Methods for controlling a Procurve switch in bench / edomtset mode
Rommon.h /rommon.c
Methods for controlling a Cisco rommon router.
term.c , term.h
Terminal I/O handling utilities
RS-232 library by Teunis van Beelen, URL: http://www.teuniz.net/RS-232/
WIN32
Linux
Host RS232 port
RS232
Target device RS232/ Serial port
HP Procurve bench mode (jumper) or edomtset terminal.
Cisco Rommon terminal

Figure 5-1: Sanitty software layers. Green = implemented. Red is future add-ons.

5.1.1 RS232 layer

I used the rs-232 library by Teunis van Beelen (URL: http://www.teuniz.net/RS-232/). It provides a general interface to configure, send, and receive data from a RS232 port. The methods are translated into Microsoft Windows or native calls depending on the platform the source is compiled on. I have only tried to compile Sanitty on Windows 7, but it was designed to use ANSI C as much as possible in order to make it portable.

The OS kernel puts received data in its internal 4 KB receive buffer (interrupt triggered). Later this buffer is polled by RS232_PollComport().The read function is non-blocking, hence it reads whatever amount of data happens to be present in the OS's buffer. Writes are done with a blocking call. The entire program runs in one thread which makes it quite simple, but it is a bit ugly and suffers from a big danger: If the OS kernel buffer is not emptied regularly, then incoming data will over run the 4 KB kernel buffer and cause data loss. For instance a big (~4 KB) blocking write to the RS232 port would echo back into the receive buffer and overfill it. This could also happen if the OS kernel scheduler suspends our thread for too long.

Fortunately, this is not a big problem for us - because most commands that are sent are quite short. I tried to feed 115.200bps into a continuous RS232_PollComport() loop on a Windows 7 Core i7. It filled 1000 bytes of the buffer (25%) all the time so it seems the windows scheduler let us run often enough to empty the buffer safely. Changing the priority via the Windows Taskmanager did **not** make any difference.

However, to be safe I added a check. If the OS kernel buffer is filled, then an error will be logged and the read will be treated as failed. The robust method of doing this would be to make a separate read thread which moves data from OS kernel buffer periodically (such as every 100 ms).

5.1.2 Term layer

This layer adds functionality to the raw RS232 methods. Especially to interact with a prompt based terminal host. For instance, the method term_sendCmdGetRes() sends a command string, waits for the prompt to arrive to indicate completion, then ensures the command string itself is echoed (for robustness) and return the point of execution between the command and the new prompt.

The terminal functions are generic and have no bindings to any particular device. It might even be possible to add a telnet backend instead of RS232.

5.1.3 PCbench (ProCurveBench) layer

The pcbench layer controls a Procurve switch using the bench or edomtset mode. Bench mode is entered by setting the bench jumper on the logicboard. Edomtset mode is entered by executing the command “edomtset” twice. Once we are in one of these mode, then we can use the hidden commands to access the device’s memory. The problem is to ensure we arrive in this mode in a consistent and robust way. When the program starts, Sanitty handles the switch being in any of the states listed below. The Pcbench_enterBenchMode() procedure is responsible for bringing the switch to either edomtset or bench mode at the highest possible RS232 speed rate (115.200bps or user selectable).

- Booting or at copyright screen waiting for two 2 carriage causes the device to autosense the line-speed.
- Logging in at the enable prompt selects a certain speed. Sanitty iterates through the RS232 speeds to find the current line-speed.
- Already in edomtset mode ($ prompt) or bench mode (= prompt) at a certain speed. Again, Sanitty has to try all the speeds to find the current one.

One state which is not currently handled is a device with a username and/or password set. It is understood and reported as unhandled though. I did not find any secure, user friendly method to have the user add the credentials in ANSI C. Future improvements could add a GUI via which the user can enter these details.

The memory of interest in these switches is that implement in the flash chips(s). All of the different switch models all flash chips, but they differ in some important properties:

- Chip storage size,
- Memory region the flash is mapped into, and
- Unlock sequence which must prepend any writes (a security feature in the flash chip to prevent unintentional overwrites).

Sanitty stores these features in a database. The function pcbench_getModelInfo() takes the switch part number, such as J4900A and returns the flash info record for the particular switch. I tried to parse the Part# from the entry screen but for some reason could not make is robust. Therefore this data is currently entered as a mandatory parameter on the command line.

5.1.4 Sanity_pc

The command line program executable. contains main() and is responsible for parsing command line options and commands. Only one command can be sent at a time.

5.2 Commands

The current sanity_pc.exe executable can perform the commands shown in Table 5-1.

Table 5-1: sanitty commands

Command	**Result**
Sanitize	Writes 0x00 to the data portion of every ghost node in the nv flash file system. A ghost node is the copy of a previous file which was deleted (i.e., it was marked as inactive).
dumpflash [filename]	Fetches the binary contents of the entire flash memory and writes this data to a file in the local filesystem.
dumpnvfs [filename]	Fetches each nv fs node and writes them, hexdump style, into a file on the local filesystem.

In the first program version I had the sanitize command erase all nodes, even the active ones (except for the .bootblock.). That caused a J9088A switch to go into a bootloop when rebooted with an error message “CONFIGURATION INITIALIZATION FAILED: Corrupt or invalid Config”. It was possible to boot into bench mode using a logic board jumper, but the switch would still crash when trying to inspect the config (Error: TLB Miss: Virtual Addr=0x0000001”. The error was recoverable by issuing a *nvfserase*, but since that command is not available on all devices in order to not upset the switch the sanitize command now only erases inactive nodes.

5.3 Performance

Communicating via RS-232 with a Procurve Switch is easy since the port is located on the outside of the unit. The downside is slow data transfer speed.

Most devices (including the procurves) use a RS232 terminal character framing referred to as 8N1[*]. Every character is transferred as 10 bits (one start bit, an 8 bit character, and a stop bit). With a maximum RS-232 speed of 115,200 we could potentially transfer 11,520 characters per second.

The Procurves built-in read command can read 256 bytes at a time and present it as a hexdump style output, as shown in in Output listing 5-1 coming from the Sanitty terminal log. Currently 1,517 characters are transferred to represent 256 bytes of memory. Thus the memory read bandwidth is theoretically 11,520 * 256 / 1,517 = 1,944bytes/s.

In practice, the read bandwidth is 1,700 bytes/sec. As a result, a 16 MB entire flash takes 2.5 hours to dump. A completely full 1 MB nv fs filesystem takes 10 minutes to dump. This read bandwidth could be improved by configuring the *read* command to present data in a more compact form. This is done via the “sm” command. It might be possible to skip the address and ASCII columns. Additionally, the prompt could be set to something shorter than the default prompt. However, using hexadecimal encoding two characters will be required to represent one byte - so the theoretical read maximum bandwidth limit using hexadecimal coding over RS-232 is 11,520/2=5,700 bytes/sec.

[*] http://en.wikipedia.org/wiki/8-N-1

Output listing 5-1: Procurve Reading a 256 byte block through Sanitty

```
tty=none ProCurve Switch 2610-48$ read 0xbcee3100

bcee3100  50 20 28 0a 49 4e 44 45 58 3d 31 31 0a 29 0a 0a   P (.INDEX=11.)..

bcee3110  43 4f 53 5f 44 53 43 50 20 28 0a 49 4e 44 45 58   COS_DSCP (.INDEX

bcee3120  3d 31 32 0a 29 0a 0a 43 4f 53 5f 44 53 43 50 20   =12.)..COS_DSCP

bcee3130  28 0a 49 4e 44 45 58 3d 31 33 0a 29 0a 0a 43 4f   (.INDEX=13.)..CO

bcee3140  53 5f 44 53 43 50 20 28 0a 49 4e 44 45 58 3d 31   S_DSCP (.INDEX=1

bcee3150  34 0a 29 0a 0a 43 4f 53 5f 44 53 43 50 20 28 0a   4.)..COS_DSCP (.

bcee3160  49 4e 44 45 58 3d 31 35 0a 29 0a 0a 43 4f 53 5f   INDEX=15.)..COS_

bcee3170  44 53 43 50 20 28 0a 49 4e 44 45 58 3d 31 36 0a   DSCP (.INDEX=16.

bcee3180  29 0a 0a 43 4f 53 5f 44 53 43 50 20 28 0a 49 4e   )..COS_DSCP (.IN

bcee3190  44 45 58 3d 31 37 0a 29 0a 0a 43 4f 53 5f 44 53   DEX=17.)..COS_DS

bcee31a0  43 50 20 28 0a 49 4e 44 45 58 3d 31 38 0a 29 0a   CP (.INDEX=18.).

bcee31b0  0a 43 4f 53 5f 44 53 43 50 20 28 0a 49 4e 44 45   .COS_DSCP (.INDE

bcee31c0  58 3d 31 39 0a 29 0a 0a 43 4f 53 5f 44 53 43 50   X=19.)..COS_DSCP

bcee31d0  20 28 0a 49 4e 44 45 58 3d 32 30 0a 29 0a 0a 43    (.INDEX=20.)..C

bcee31e0  4f 53 5f 44 53 43 50 20 28 0a 49 4e 44 45 58 3d   OS_DSCP (.INDEX=

bcee31f0  32 31 0a 29 0a 0a 43 4f 53 5f 44 53 43 50 20 28   21.)..COS_DSCP (
```

Current write performance to flash is extremely slow, about 5 bytes/second with an RS-232 speed of 115,200bps. The reason for this can be seen in Output listing 5-2 where the byte 0x00 is written to flash address 0xbcee075e. However, to enable this flash address for writing the special 3 byte security sequence has to be written, thus we transfer 274 characters to write a single byte(!). This gives a theoretical write bandwidth of 11,520 * 1 / 274 = 42 bytes/s. However, I suspect that there is a lot of delays in the send/wait for prompt interaction (the OS scheduler might not let us run exactly when there is enough input present). There is a lot that can be done to improve this write performance bandwidth. These optimizations include:

- Set a shorter prompt (about 70% of bandwidth is now consumed by the prompt text)
- Write a word (16 bits) at a time. The flash chip of the J9088A ProCurve Switch 2610 (S29GL128P) actually writes 16 bytes even though 8 bit writes are issued (the other 8 bits become 0x00).
- Disable the read protect feature for the flash completely, perform the writes; and then activate protection again.
- Find a format to feed multiple address / data pairs into the *wr* command in one execution. In fact it seems to be a separate command to do just that called *fill* (See Output listing 4-18).

One way to improve both read and write bandwidth would be to interact with the terminal over Telnet or SSH to overcome the limit of RS-232 asynchronous speed. For example, the Sanitty tool could setup an IP configuration in “running-config”, without committing it to flash, and then switch over to TCP/IP communication.

Output listing 5-2: Procurve writing of a single byte through Sanitty

```
ProCurve Switch 2610-48=> wr 0xbc000aaa 0xaa

ProCurve Switch 2610-48=>

ProCurve Switch 2610-48=> wr 0xbc000555 0x55

ProCurve Switch 2610-48=>

ProCurve Switch 2610-48=> wr 0xbc000aaa 0xa0

ProCurve Switch 2610-48=>

ProCurve Switch 2610-48=> wr 0xbcee075e 0x00
```

5.4 Compatibility

The current two devices supported by the Sannitty program are the J9088A ProCurve Switch 2610 and J4900A ProCurve Switch 2626. Selecting one of these models and reading NV records of a different model may work for other devices that share the same file system structure. However, the flash writes must share flash properties with the device selected, i.e., using the format: base address, size, and write protect deactivation pattern. Additional devices may be added within the method makeModelInfoDB() in pcbench.c

5.5 Future improvements

These are some improvements that can be made (in addition to the performance improvements suggested in the earlier section). These improvments are described in the following subsections.

5.5.1 Flash chip autodetect

Flash chips often have a special method to probe them for their size, vendor, and type by writing a predetermined magic sequence to them. The tool’s robustness could be improved by first verifying that the flash chip is correct, before interacting with it. This functionality could even be necessary if switches of the same part number use different flash chips.

5.5.2 Sense nvfserase command presence

The *nvfserase* command was able to sanitize the J9088A ProCurve Switch 2610 – 48 switch (see Section 4.4). However, this command is not present in all switches. The sanitize function could check for the presence of this command and use it for sanitization (as using this command is must faster that doing all of the individual writes necessary to perform the equivalent task).

The entire program could be rewritten in Java to create a more robust, easier to maintain, and more user-friendly tool.

5.6 Forensic value and sanitization trust level

The edomtset/bench mode code seems to be in the actual executable main flash image. This is somewhat different from the Cisco Rommon, where Rommon is a separate executable file (sometimes even in a ROM chip). When using the sanitty tool for Procurve forensics, one must take into account that malicious firmware could intercept the read and wr methods and present different data to the tool than is actual present in the flash. In this way it could block sanitization writes. Since it is flash and we do not have an erase method for all Procurve devices, we can only do ones to zeros transitions based sanitization. Thus, we cannot prevent data interception by using the high entropy fill techniques discussed in Section 2.2.3 “Delete and overwrite free space”.

A higher trust level could of course be achieved using JTAG or de-soldering the flash and using an external reader. I believe the sanitization trust level offered by the built in memory access commands is sufficient for the refurbishing industry as long as one understands its limitations.

The next section will show the data extracted from the tool is equivalent to the contents read from the flash using an external flash programmer.

5.7 Comparison with chip read by an external programmer

In the device investigations of Chapter 4, hidden built in debug commands were used to read and write to flash memory. An interesting question is whether these commands accurately present a snapshot of the data in the flash. To confirm the correctness of the Sanitty *dumpflash* method and to serve as a “proof of concept” of the validity of the forensic value it offers, I compared the program’s output to the flash data extracted from a desoldered chip (as read by an external programmer). This section describes the complete process, from desoldering to comparison.

5.7.1 Desoldering of the flash chip

The flash chip is a 16 MB Spansion S29GL128P11TFI01 in a 56 pin Standard Thin Small Outline Package (TSOP). This chip is surface mounted at position U8 on the board as shown in Figure 5-2 and Figure 5-3.

Figure 5-2: Procurve J9088A-2610-48-flash chip location

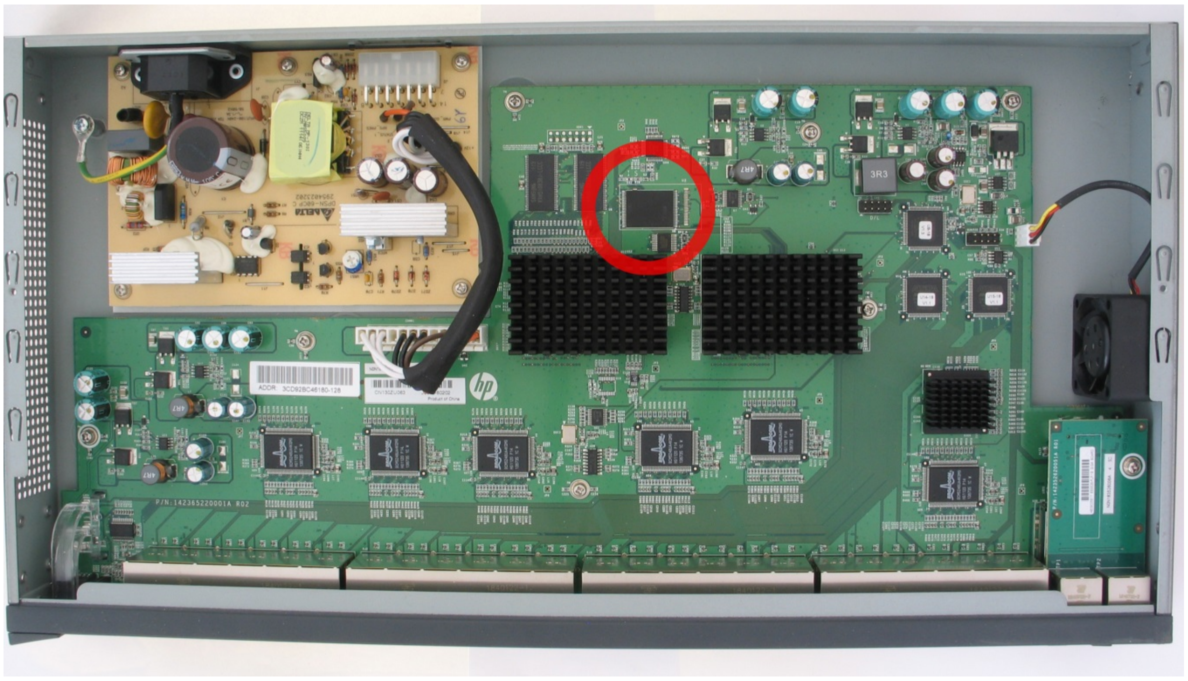
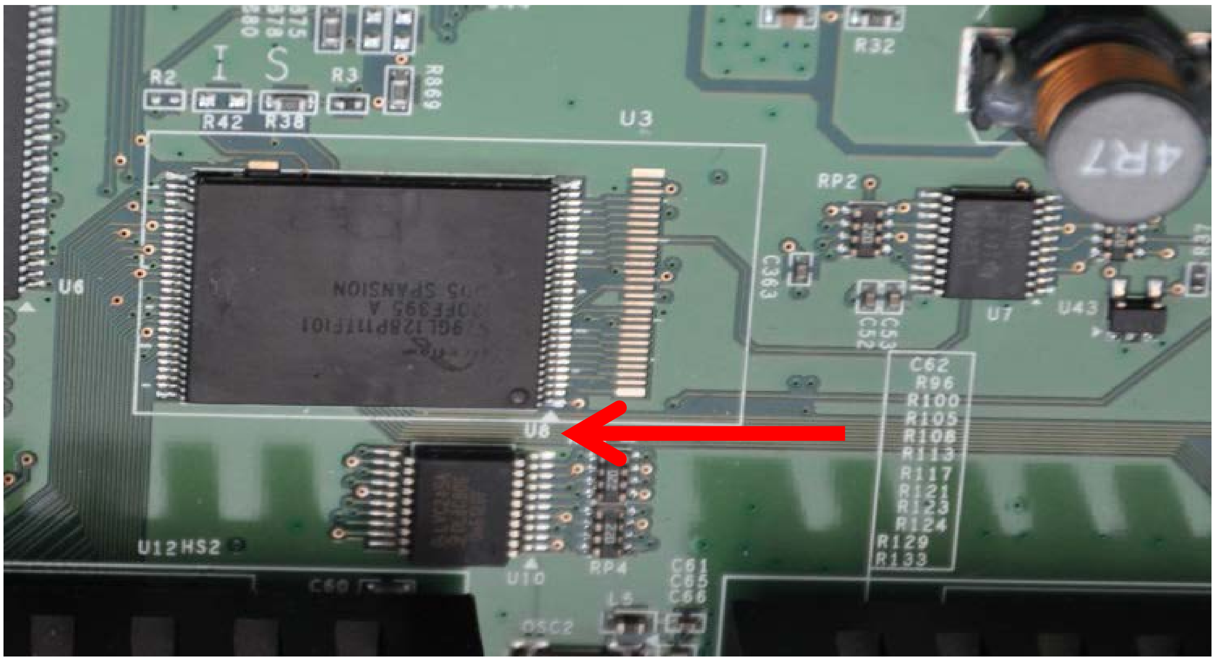


Figure 5-3: Procurve J9088A-2610-48-flash chip location close up



I desoldered the chip by using a standard hot air gun set at 350 degrees Celsius placed in a drill stand (see Figure 5-4). I found this to be as good as a “professional” desoldering station. The air stream was not as focused, but since the side effect of adjacent components being desoldered did not matter, this method was efficient and left both hands free to work with the chip. An angled tweezer with adhesive tape turned out to be as efficient as a vacuum picker of a professional station.

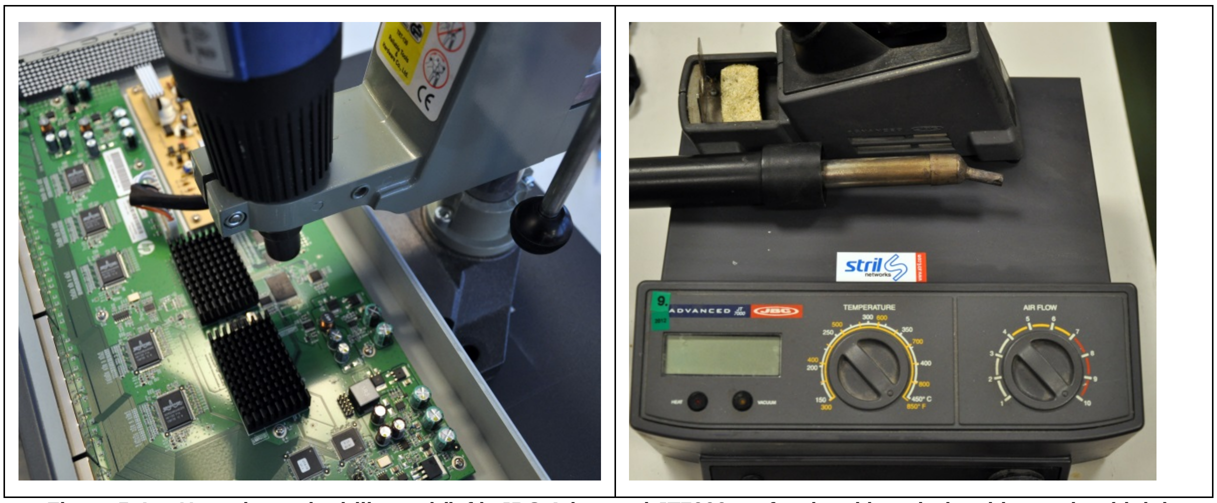
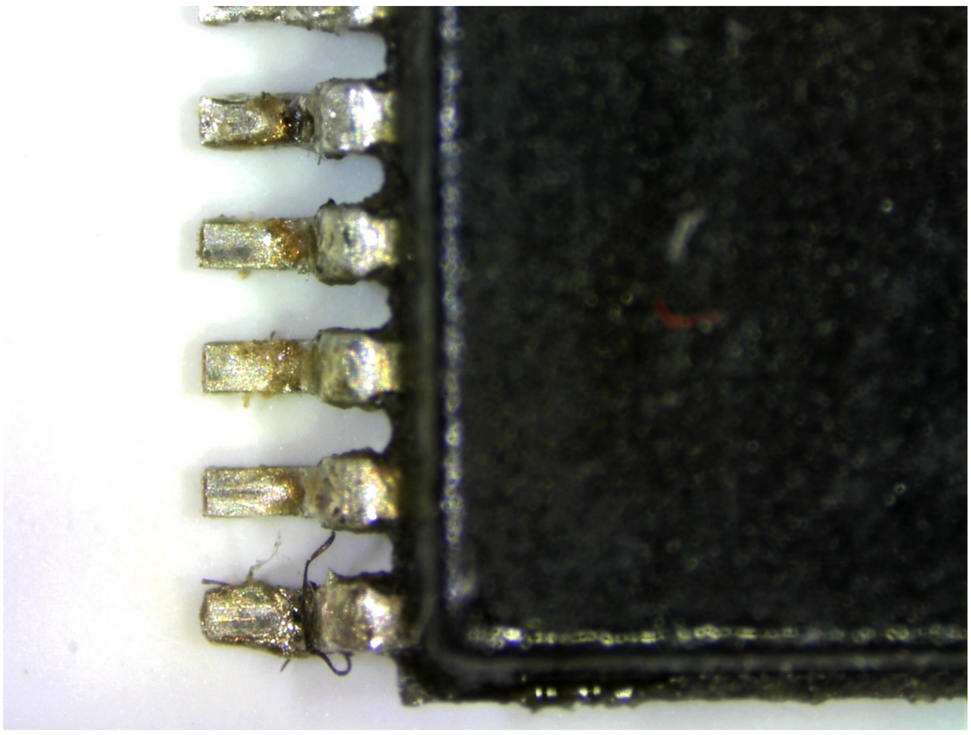


Figure 5-4: Heat air gun in drill stand (left). JBC Advanced JT7000 professional hot air desolder station (right)

5.7.2 Cleaning the chip

The distance between the chip's pins is only 0.28 mm. Solder causing bridges between the legs has to be removed. I first tried to cover the pins with tack flux and absorb the solder on a copper braid under the hot air stream. Then I cleaned the pins with flux cleaning fluid on a cotton swab. The chip looked nice under a 4X magnifying glass. However, under a microscope the pins looked worse. Residue of the flux was still present and treads from the cotton swab had been glued on and between the legs. Additionally, the flux had left a crust on the pins which I could only remove by scratching with a fine dental pick.

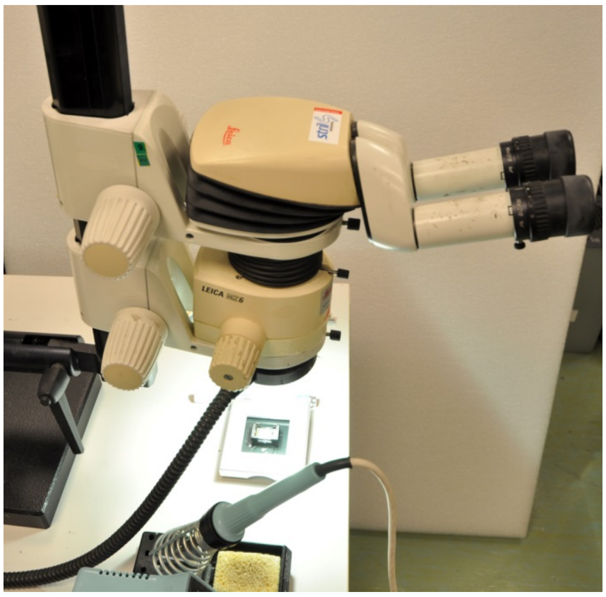
Figure 5-5: Tack flux residue and dirt on chip pins



I repeated the procedure using a non-tack flux on the brad and pins and a lint free cloth. This worked better. The procedure was recorded in a video (url: https://www.youtube.com/watch?v=KrILApdpnFY). I learned it is absolutely necessary to work under a microscope with at least 50X magnification to see what is going on. A low-cost digital Vtiny UM6 microscope turned out to be sufficient, but a professional Leica stereo microscope offered a better working view due to the stereo perspective.

Figure 5-6: VTiny low cost microscope

Figure 5-7: Leica stereo microscope



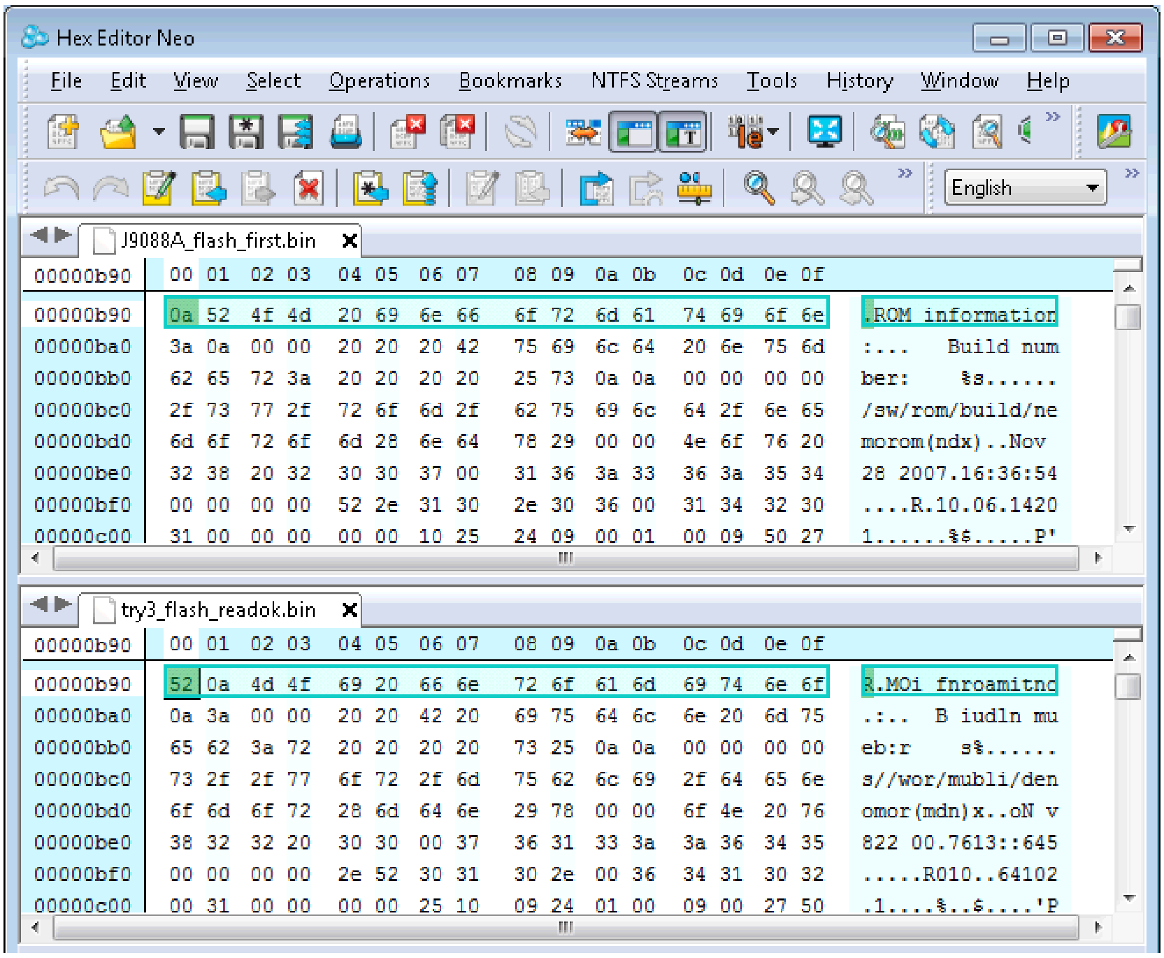
5.7.3 Reading out the data

Aligning the chip in the TSOP56 adapter of the Xeltek SP6100 was a difficult task. It took about 20 tries until the programmer reported a good connection to all 56 leads .In fact it was necessary to align the chip with the adapter under a microscope. This video link shows one of the unsuccessful tries using the VTiny microscope: url: http://www.youtube.com/watch?v=yBq5lGwqyw8

It seemed the most difficult part was to place it properly centered vertically. Again, it was easier to work under the stereo microscope..

The 16 MB data extracted from the chip through the programmer was different from the data extracted from Sanitty. The data from the programmer had every two bytes of a16 bit word swapped. That is, the most significant byte was stored at the lowest address (called Big-Endian system). Figure 5-8 shows the two variations in an hex-editor. The reason for this difference is that the CPU reads and writes 16 bit data at a time using the big endian system. But when we read data via the read command in the edomtset mode we do it by byte access and the command presents the least significant byte as the contents of the lower address.

Figure 5-8: Flash contents. Sanitty source (upper), Flash programmer source (lower)


Hex Editor Neo
File Edit View Select Operations Bookmarks NTFS Streams Tools History Window Help
English

J9088A_flash_first.bin

```
00000b90  00 01 02 03  04 05 06 07  08 09 0a 0b  0c 0d 0e 0f

00000b90  0a 52 4f 4d  20 69 6e 66  6f 72 6d 61  74 69 6f 6e   .ROM information
00000ba0  3a 0a 00 00  20 20 20 42  75 69 6c 64  20 6e 75 6d   :...   Build num
00000bb0  62 65 72 3a  20 20 20 20  25 73 0a 0a  00 00 00 00   ber:    %s......
00000bc0  2f 73 77 2f  72 6f 6d 2f  62 75 69 6c  64 2f 6e 65   /sw/rom/build/ne
00000bd0  6d 6f 72 6f  6d 28 6e 64  78 29 00 00  4e 6f 76 20   morom(ndx)..Nov
00000be0  32 38 20 32  30 30 37 00  31 36 3a 33  36 3a 35 34   28 2007.16:36:54
00000bf0  00 00 00 00  52 2e 31 30  2e 30 36 00  31 34 32 30   ....R.10.06.1420
00000c00  31 00 00 00  00 00 10 25  24 09 00 01  00 09 50 27   1......%$.....P'
```

try3_flash_readok.bin

```
00000b90  00 01 02 03  04 05 06 07  08 09 0a 0b  0c 0d 0e 0f

00000b90  52 0a 4d 4f  69 20 66 6e  72 6f 61 6d  69 74 6e 6f   R.MOi fnroamitno
00000ba0  0a 3a 00 00  20 20 42 20  69 75 64 6c  6e 20 6d 75   .:..  B iudln mu
00000bb0  65 62 3a 72  20 20 20 20  73 25 0a 0a  00 00 00 00   eb:r    s%......
00000bc0  73 2f 2f 77  6f 72 2f 6d  75 62 6c 69  2f 64 65 6e   s//wor/mubli/den
00000bd0  6f 6d 6f 72  28 6d 64 6e  29 78 00 00  6f 4e 20 76   omor(mdn)x..oN v
00000be0  38 32 32 20  30 30 00 37  36 31 33 3a  3a 36 34 35   822 00.7613::645
00000bf0  00 00 00 00  2e 52 30 31  30 2e 00 36  34 31 30 32   .....R010..64102
00000c00  00 31 00 00  00 00 25 10  09 24 01 00  09 00 27 50   .1....%..$....'P
```

5.7.4 Method comparison and conclusion on Sanitty correctness

Two consecutive Sanitty *dumpflash* commands were issued to read out the flash contents twice via the edomtset debug mode . It took almost 3 hours per dump at 115200bps. The two reads were both binary equivalents (compared using the windows *fc* command). From this I conclude two things: (1) The Sanitty tool is robust and consistent when it comes to reading large memory contents, and, (2) the Procurve switch did not modify the flash spontaneously during these 6 hours.

After the double Sanitty read, the device was shut down by removing power. The flash chip was desoldered and its contents extracted as described in Sections 5.7.1, 5.7.2, 5.7.3. The Sanitty file was binary equivalent to a byteswapped version of the flash contents extracted from the chip using the programmer.

This demonstrates that the Sanitty tool correctly produce a snapshot of the current data of the flash memory on a Procurve switch. The benefit of using Sanitty for the process is summarized in

Table 5-2 where it is compared to the process of desoldering the chip then reading it in an external flash programmer and resolder it back on the board (Sanitty *dumpflash* command leaves a functional switch so it is fair to compare it with a method which can resolder the chip successfully).

Table 5-2: Comparison of the Sanitty and external flash reader data extraction method[*]

	Sanitty dumpflash command	**Flash put in external reader (resoldered to board after read)**
Tools required	Serial cable Sanitty program	Hot air and/or infrared rework station to have a robust reflow process when soldering the chip back. Solder paste dispenser Stereo microscope TSOP56 Flash reader Solder iron. Dental pick, tweezers, solder paste, flux Electrostatic discharge protected work area
Cost of tools	$5	US$5.000-US$50.000 depending on quality level on the rework station, flash reader and microscope.
Time required	5 minutes of setup + 3 hours of fika[†]	5 hours of labor. No fika.

5.8 Sanitization confirmation using Sanitty

In Section 4.4 we concluded that the HP built in procedure HP_2626_BUTTON from Appendix A did not properly sanitize the Procurve 2610-48 switch. Let's repeat the procedure but and follow up with the Sanitty *sanitize* command to investigate the effect of the sanitization.

A HP Procurve 2610-48 switch J9088A running firmware R.11.107 was prepared with the 10 character markers in Table 4-21. The HP_PROCURVE_SANITTY procedure in Appendix A was performed. The flash content was inspected (using the Sanitty dumpflash command) just after the vendor sanitization method was performed. All markers were found. Then the Sanitty *sanitize* command was performed followed by a merker search in the entire 16MB flash again. None of the markers were found.

Can we conclude from this test that the Sanitty tool properly sanitizes this Procurve? No! Not finding any markers is equivalent to an *unknown* sanitization result (see Chapter 3). For instance the device could store a copy of the manager password string backwords in some area of the flash which would not be found in a string search. However, an *unknown* sanitization result is of course still better than the vendor recommended procedure which was proven *unsafe*.

[*] All costs are rough guestimations
[†] Fika is a Swedish word for a break with tea or coffee

Table 5-3: ProCurve Switch 2610-48 markers

Parameter to host marker	String marker (10 random chars)
SNMP password	MARKaoQvioTAGp
Manager password	MARKrzeNUuCONZ
Hostname	MARKncfmQxDMTD

6 Conclusions and Future work

This chapter draws conclusion based upon the investigations that were done. It describes the goals that were achieved and areas where future work could be done to continue the work. The chapter concludes with some reflections on the societal impacts of this thesis project.

6.1 Conclusions

In my research I found that common enterprise network devices have flaws in their sanitization routines. A CISCO 1712 router and two HP Procurve switches were investigated in detail. After following the vendor's sanitization procedures, the Cisco router still contained the VLAN information in flash and the Procurve switches still contained the entire configuration (not only the last configuration, but also copies of previous configurations). Clearly, the developers of the software in their devices did have not the transfer of equipment from one owner to another in their minds when designing their sanitization commands. The equipment needed to extract old (thought to be erased) configurations is a computer with a RS232 port adapter and an RS232 cable. The cost is less than $20 (excluding the computer) and the time needed is a only a few minutes.

Before starting with these investigations, I thought JTAG would be the best method to access the nonvolatile memories of these devices. I even purchased five JTAG tools and a $4000 flash programmer to use for the tests. However, in practice these access methods were complicated to develop and difficult to use from a professional refurbisher's point of view (especially for someone who may have to process thousands of devices per month). The easiest method to access these devices was via an external asynchronous RS-232 port using the hidden development commands that the vendor left in the firmware's CLI. These commands provide easy access to the device's nonvolatile memories.

6.1.1 Proposal to vendors

What could vendors do to plan for future sanitization of their devices? It would be convenient to have a single "sanitize all" command to sanitize the entire device, boards, and expansion modules currently present in the device. This command would essentially return the device to the state it had when it was shipped from the factory.

Given the difficulties of sanitizing certain types of storage, such as managed NAND flash, there is another approach: the cryptographic erase discussed in Section 2.3.6. A cryptographic key would be generated the first time the end user boots the device and this key would be stored in a easy to sanitize memory location (e.g. preferably in EEPROM and never in managed flash). All subsequent writes to any non-volatile storage must be encrypted using this key. All subsequent reads from these devices would use this key to decrypt the stored contents. Upon decommissioning a device the key is simply overwritten. As a result the stored data cannot be decrypted (assuming that the key length is sufficiently long and that the encryption method used is sufficiently safe).

6.2 Limitations

The aim of this thesis project was to test a large variety of devices from multiple vendors. However, the work required per device was more than I anticipated so I ran out of time. With the limited set of devices tested, this thesis does not attempt to identify any specific vendor as having worse sanitization routines than any other. However, from this small sample size it is clear that sanitization of devices has **not** been a design goal for the software found in this sample of devices.

6.3 Future work

This section summarizes work during the investigations which was left for the future either because of time constraints or because the topic was outside the focus of this thesis project.

6.3.1 Further development of Sanitty

While the tool provides the basic functionality for sanitizing two different HP Procurve switches, there are a number of obvious improvements that should be made to this tool. These include:

- Add more Procurve devices (i.e. add information about their flash chips to the Sanitty device database).
- Add support for issuing commands to Cisco's Rommon in order to sanitize Cisco devices.
- Integrate support utilities, such as decoding the configuration of the CPU and memory controller.
- Improve the program's performance. Possibly by connecting over telnet/ssh. Investigate whether the remote debugger interface is more efficient and requires less overhead to transferring memory contents, especially when writing to flash.
- Rewrite the program in Java to make a robust command line and GUI tool.

6.3.2 JTAGulator extest pin mapper

It would be nice to add a "EXTEST pinout probe test" to the JTAGulator to aid in automating the mapping between a EXTEST boundary scan register position and an exposed pin on an IC. It would work as follows:

1. A JTAG enabled chip presumed to be connected to the pins of the memory chip of interest is identified.
2. One of the pins on the JTAGulator is configured as input: the probe-pin.
3. A probe is connected to the probe-pin and connected to a pin of the chip to map. For chips with a narrow pin spread the probe could be a plastic film laminated with a conductive tape with appropriate thickness to be squeezed in between the chip pins. Or perhaps a small probe pin operated under a microscope.
4. The JTAGulator would inject a single bit into the extest register and shift it around until it is recognized on the probe pin. This is repeated for all the chip's pins of interest and eventually we have the relation between each chip's pin (of interest) and its corresponding extest register position. We can use this information to interact with the chip.

As far as I know, there is no tool available yet with the above feature. Not even in the $10,000+ extest testers from XJTAG.

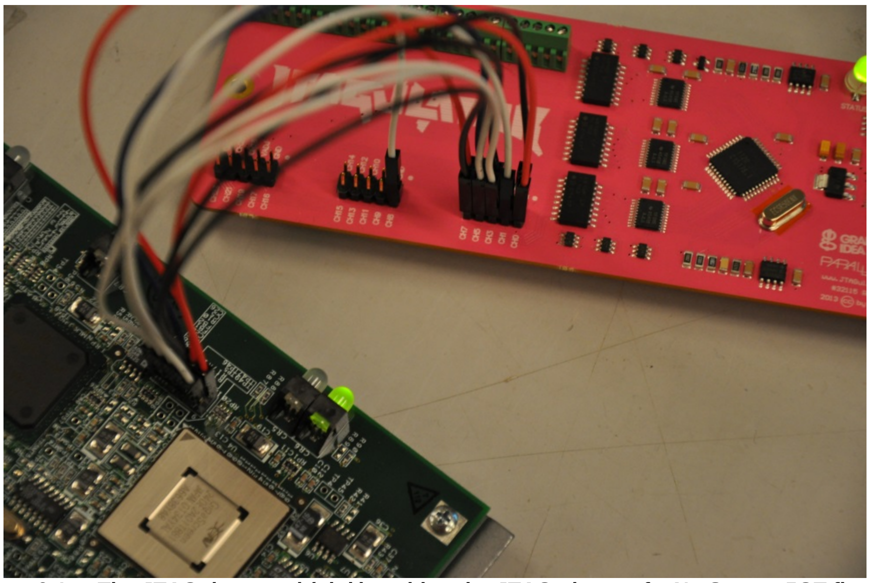


Figure 6-1: The JTAGulator tool (pink) probing the JTAG pinout of a NetScreen 5GT firewall

6.3.3 Writing to NVRAM from Cisco Rommon

It would be nice to have write access to the NVRAM chip via Cisco's Rommon. Further investigation is needed to understand how to activate the WE signal from the CSICO 1712 rommon prompt (as investigated in Section 4.1.3.4).

6.3.4 Investigate more devices

It is a straightforward task to investigate additional Cisco routers with Rommon support by using the methods described in this thesis. The same is true for additional models of HP Procurve switches. It would be interesting to develop JTAG access methods for devices not offering memory access via the terminal CLI.

6.3.5 External storage of sensitive data

One approach[*] is to avoid storing any sensitive data in the device in the first place. Some devices have external memory card sockets such as compact flash or USB ports. Before starting using the device a simple configuration could be done to tell the software to look for the main config in an external memory card After that the WE pins of the chips normally holding the config is electrically tied to a level which prevents further writes, i.e. a physical write protect. Cisco routers have the choice of specifying the startupconfig in the environmental variable CONFIG_FILE or as a config-command "boot config xxx:xxx" [91Para. Specifying the Startup Configuration File]. This approach needed to be further investigated to learn how these devices react when they cannot write to the NVRAM. Upon decommissioning and transfer of ownership the write protect is removed, and the memory card kept.

6.3.6 Tool for the Motorola BDM interface

As covered in Section 0, I failed to read and write flash and NVRAM memory contents of the CISCO 1712 using the PE-Micro Cyclone MAX BDM debugger tool. Since many embedded networking devices use a Freescale Semiconductor CPU it would be nice to further investigate this method of memory access. Either with this or another BDM interface tool.

[*] Suggested by G. Q. Maguire Jr.

Two features that do not (yet) exist in the Cyclone MAX would be especially useful:

1. Take control of the CPU without resetting it. Alternatively, reset it in a “soft” way which keeps the memory controller configuration intact.
2. Extract and decode the memory controller’s configuration registers inside the CPU. This requires the step above. This will provide *very* useful information as to which memory regions contain the non-volatile memories of interest for sanitization.

I believe the Cyclone MAX debugger tool has the appropriate hardware interface to do the job. However, a low level software API is needed to control the tool as the supplied software tools are of no help.

6.3.7 In-circuit programming of a parallel EEPROM

In Section 4.1.6 I failed to read (in-circuit) the CISCO 1712 EEPROM soldered to the logic board. It would be interesting to see if a programmer could be built with sufficient “power” to control the TTL levels of the chip in competition with other drivers. Of course, this should also be done without damaging the chip or any components connected to it.

The web article by Andromeda Research Labs entitled “Understanding In-Circuit EEPROM and Microcontroller Reading and Programming” [81] has some advice on serial EEPROMs which may be used as a starting point for future work.

6.3.8 Proof of concept: Malicious code inside a flash controller

Create firmware for the embedded CPU on a Compact Flash or USB memory stick that interferes with sanitization attempts. The purpose would be to prove that the precense of a malicious firmware would need to be considered when performing sanitization.

6.3.9 Proof of concept: Remote VTP packet injection

Test if a VTP packet can be tunneled remotely into a LAN, like a Trojan horse, with the “Ethernet frame in IP datagram” technique discussed in Secti0n 4.1.8.1.

6.3.10 Fake a file spanning the entire flash

I didn’t find any way to dump the whole flash over a IP based protocol such as TFTP or FTP in either the Cisco Rommon nor HP procurve edomtset mode. However both Procurve firmware and Cisco IOS has methods to copy a *file* to a TFTP server. So if we could create or modify an existing file to span the *entire* flash range we could “fool” the device to send it out over TFTP as any regular file.

For Cisco rommon based routers this could be done by modifying the length field of the first file to include the entire flash. If the first file is the IOS executable the router won’t boot as the checksum now is invalid. We could either recalculate the checksum or load an IOS image into ram via TFTP and boot that.

In HP Procurve emotset mode I didn’t find any flash erase or memory copy commands. If a memory copy command could be found we could copy the while flash into some unused portion of RAM, create a file header according to the fields in Table 4-15, and link it in from the last existing file. That “next” address of the last file header is currently 0xFFFFFFFF so it is possible to flash-write any RAM address to it. If the Procurve filesystem implementors forgot to make a sanity check on the value set to next address it might just copy the file even if it is in RAM.

Both these methods require modifying the flash contents slightly, but since we know the old values we could write them back to the forensic image of the flash.

6.3.11 Try to force a Procurve switch to reinitialize its flash file system

In two switches (J4900A, 2626 and J4903A ,2824) I couldn't find any command to sanitize the flash non-volatile file system and remove the inactive ghost file blocks by erasing the whole area. It would be interesting to see if it is possible to provoke and force the switch to do it. Such as filling up the space entirely with a dummy file and then ask it to create a new file. Or by setting the next pointer of the first block to zero to cause a file system error.

6.3.12 Non perfect random number generator impact on marker strength

In the theoretical research of marker strength (see Section 3.2 and Appendix I) the markers were assumed to be constructed using a perfect random number generator (RNG). In reality RNGs will not be perfect. One problem is if a pseudorandom number generator (PRNG) is used and which, for some "bad" seed, has a very short period. That would cause markers to repeat which is bad because we assume that each marker is independent to all other data *including* other markers. Since markers are short, such as 15 bytes, it shouldn't be a big problem but it has to be understood. More importantly it is essential to not use the same seed when initializing the PRNG because that will obviously cause marker repetition.

The effect of a non-perfect RNG or PRNG on the marker strength is left to future work. Perhaps the effect can be quantified theoretically, or proved so small we don't have to consider it given that the RNG/PRNG possess some minimum level of "perfectness".

6.4 Reflections

Equipment reuse is environmentally friendly. Additionally, it is economically advantageous for many customers, for whom older equipment meets their requirements. Moreover, many customers are able to utilize networking equipment that they would not normally have been able to afford (if they had to buy only new equipment), hence the secondary market offers many benefits to society. However, uncertainty about the correctness of the built in vendor commands to properly sanitize a unit may deter companies to sell their equipment in a secondary market.

This thesis has shown that vendors' recommendations for configuration erasures should **not** be trusted. Companies may choose to scrap their equipment to prevent leaking sensitive information. I assume that equipment vendors have no real incentive to facilitate transfer of ownership of their devices, as they make their money selling new equipment and probably prefer that decommissioned devices be taken out of market - so they do not interfere with the sale of their new equipment. Therefore, the pressure to implement proper sanitization methods in the equipment has to come from customer demand or by governmental environmental legislation. Thus there is a need for societal pressure to "encourage" equipment reuse (as appropriate).

Currently sanitization is often the responsibility of the equipment refurbisher. Knowledge, research, and tools could be shared via trade organizations such as United Network Equipment Dealer Association (UNEDA). Additionally, a trade organization might introduce a process to offer some type of certification to the end users asserting that a given refurbisher has the tools and knowhow to properly sanitize specific types of equipment. Alternatively, vendors might offer certification of a refurbisher for a given vendor's equipment.

References

[1] ‘Definitions of the SI units: The binary prefixes’. [Online]. Available: http://physics.nist.gov/cuu/Units/binary.html. [Accessed: 22-Jan-2015]

[2] ‘EUR-Lex - 32012L0019 - EN - EUR-Lex’. [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32012L0019. [Accessed: 13-Nov-2014]

[3] ‘Erase - Definition and More from the Free Merriam-Webster Dictionary’. [Online]. Available: http://www.merriam-webster.com/dictionary/erase. [Accessed: 13-Jan-2015]

[4] ‘Chapter 8. Remote OS Detection’. [Online]. Available: http://nmap.org/book/osdetect.html. [Accessed: 20-Jan-2015]

[5] P. Gutmann, ‘Secure Deletion of Data from Magnetic and Solid-state Memory’, in *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, Berkeley, CA, USA, 1996, pp. 8–8 [Online]. Available: http://dl.acm.org/citation.cfm?id=1267569.1267577. [Accessed: 12-Jan-2015]

[6] G. Rostky, ‘Remembering the PROM knights of Intel | EE Times’, 07-Mar-2002. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1144961&. [Accessed: 03-Jan-2015]

[7] Spansion, ‘Flash Memory: An Overview’. [Online]. Available: http://www.spansion.com/Support/Application%20Notes/FlashOverview_AN.pdf

[8] ATMEL, ‘AT28C010, 1-megabit Paged Parallel EEPROM’. [Online]. Available: http://www.atmel.com/Images/doc0353I.pdf

[9] ‘Basic information about memory chips and programming’. [Online]. Available: http://www.batronix.com/shop/electronic/eprom-programming.html#06. [Accessed: 03-Jan-2015]

[10] ‘M24C08-WBN6P - STMICROELECTRONICS - IC, EEPROM I2C 8K, 24C08, DIP8 | Farnell element14 UK’. [Online]. Available: http://uk.farnell.com/stmicroelectronics/m24c08-wbn6p/ic-eeprom-i2c-8k-24c08-dip8/dp/9882820. [Accessed: 03-Jan-2015]

[11] Xeltek, ‘目录 - IS01_Manual.pdf’. [Online]. Available: https://www.xeltek.com/software/spIS01/IS01_Manual.pdf. [Accessed: 03-Jan-2015]

[12] A. Tal, ‘Two Flash Technologies Compared: NOR vs NAND’. Oct-2002 [Online]. Available: http://focus.ti.com/pdfs/omap/diskonchipvsnor.pdf. [Accessed: 29-Dec-2014]

[13] Cactus Technologies Limited, ‘NAND Flash Data Storage Overview – SLC, MLC and TLC - Embedded Computing Design’. [Online]. Available: http://embedded-computing.com/news/nand-slc-mlc-tlc/. [Accessed: 29-Dec-2014]

[14] ‘TN-29-19: NAND Flash 101 Introduction’. Micron [Online]. Available: http://www.eng.umd.edu/~blj/CS-590.26/micron-tn2919.pdf. [Accessed: 29-Dec-2014]

[15] ‘Open NAND Flash Interface Specification 4.0’. 04-Feb-2014 [Online]. Available: http://www.onfi.org/-/media/onfi/specs/onfi_4_0%20gold.pdf. [Accessed: 30-Dec-2014]

[16] ‘COMMON FLASH INTERFACE (CFI): | JEDEC’. [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd-6801. [Accessed: 05-Jan-2015]

[17] J. Heidecker, ‘NAND Flash Qualification Guideline’, 11-Jun-2012. [Online]. Available: http://nepp.nasa.gov/workshops/etw2012/talks/tuesday/t04_heidecker_flash_qualification.pdf. [Accessed: 29-Dec-2014]

[18] *The Exploration and Exploitation of an SD Memory Card [30c3]*. 2014 [Online]. Available: https://www.youtube.com/watch?v=CPEzLNh5YIo&feature=youtube_gdata_player. [Accessed: 29-Dec-2014]

[19] 'K9F5608X0D_1.3_final.fm - ds_k9f5608x0d_rev13.pdf'. [Online]. Available: http://download.siliconexpert.com/pdfs/2007/08/05/semi_ap/manual/sam/ds/ds_k9f5608x0d_rev13.pdf. [Accessed: 29-Dec-2014]

[20] 'flash_mem_summit_jcooke_inconvenient_truths_nand.pdf'. [Online]. Available: https://www.micron.com/~/media/Documents/Products/Presentation/flash_mem_summit_jcooke_inconvenient_truths_nand.pdf. [Accessed: 29-Dec-2014]

[21] 'Theory and practice of flash memory mobile forensics - viewcontent.cgi'. [Online]. Available: http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1066&context=adf. [Accessed: 29-Dec-2014]

[22] 'NAND Flash Memories and Programming NAND Flash Memories Using Elnec Device Programmers'. .

[23] M. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, 'Reliably Erasing Data from Flash-based Solid State Drives', in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, Berkeley, CA, USA, 2011, pp. 8–8 [Online]. Available: http://dl.acm.org/citation.cfm?id=1960475.1960483. [Accessed: 12-Jan-2015]

[24] J. Wise, 'Reverse Engineering a NAND Flash Device Management Algorithm | Joshua Wise's domain'. [Online]. Available: http://joshuawise.com/projects/ndfslave. [Accessed: 03-Jan-2015]

[25] 'Flash Extractor Library'. [Online]. Available: http://www.flash-extractor.com/library/. [Accessed: 03-Jan-2015]

[26] 'PC-3000 flash'. [Online]. Available: http://www.pc-3000flash.com/solbase/?lang=eng. [Accessed: 03-Jan-2015]

[27] 'Hyperstone F2-16X, 32-Bit Flash Memory Controller Specification'. Hyperstone AG, 10-Mar-2006.

[28] 'AT28C16 16K (2K x 8) Parallel EEPROMs - doc0540.pdf'. [Online]. Available: http://www.atmel.com/Images/doc0540.pdf. [Accessed: 07-Jan-2015]

[29] Oracle, 'Writing PCMCIA Device Drivers'. [Online]. Available: https://docs.oracle.com/cd/E19957-01/802-6321/802-6321.pdf. [Accessed: 07-Jan-2015]

[30] 'PC CARD STANDARD'. [Online]. Available: http://affon.narod.ru/02el80.pdf. [Accessed: 07-Jan-2015]

[31] 'Joint Test Action Group - Wikipedia, the free encyclopedia'. [Online]. Available: http://en.wikipedia.org/wiki/Joint_Test_Action_Group. [Accessed: 21-Dec-2014]

[32] 'IEEE Standard Test Access Port and Boundary Scan Architecture', *IEEE Std 1149.1-2001*, pp. 1–212, Jul. 2001. DOI: 10.1109/IEEESTD.2001.92950

[33] 'JTAG - A Technical Overview - TAP Signals and Instructions'. [Online]. Available: http://www.xjtag.com/support-jtag/jtag-technical-guide.php. [Accessed: 21-Dec-2014]

[34] 'In-circuit test - Bed of Nails Testing'. [Online]. Available: http://en.wikipedia.org/wiki/In-circuit_test. [Accessed: 27-Dec-2014]

[35] 'JTAG Pinouts'. [Online]. Available: http://www.jtagtest.com/pinouts/. [Accessed: 28-Dec-2014]

[36] 'JTAGulator™ | Grand Idea Studio'. [Online]. Available: http://www.grandideastudio.com/portfolio/jtagulator/. [Accessed: 28-Dec-2014]

[37] I. M. F. Breeuwsma, 'Forensic imaging of embedded systems using JTAG (boundary-scan)', *Digital Investigation*, vol. 3, no. 1, pp. 32–42, Mar. 2006 [Online]. DOI: 10.1016/j.diin.2006.01.003

[38] 'Testing Non-JTAG Devices with Boundary Scan - e.g. Memory Testing'. [Online]. Available: http://www.xjtag.com/support-jtag/jtag-memory-testing.php. [Accessed: 03-Jan-2015]

[39] 'User Manual for USBJTAG NT'. Sep-2010 [Online]. Available: http://www.usbjtag.com/jtagnt/usbjtagnt.pdf

[40] 'Non-intrusive On-chip Debug Hardware Accellerates Development for MIPS RISC Processors'. [Online]. Available: http://read.pudn.com/downloads93/doc/363855/ejtag_debug_eetimes.pdf. [Accessed: 05-Jan-2015]

[41] 'Debug Adapter Hardware - OpenOCD User's Guide'. [Online]. Available: http://openocd.sourceforge.net/doc/html/Debug-Adapter-Hardware.html#Debug-Adapter-Hardware. [Accessed: 05-Jan-2015]

[42] 'OpenOCD User's Guide', 05-Jan-2015. [Online]. Available: http://openocd.sourceforge.net/doc/pdf/openocd.pdf. [Accessed: 05-Jan-2015]

[43] 'Instructions on doing (semi-)manual JTAG boundary scan with OpenOCD'. [Online]. Available: http://permalink.gmane.org/gmane.comp.debugging.openocd.devel/23336. [Accessed: 05-Jan-2015]

[44] 'Rommon memory dump'. [Online]. Available: https://supportforums.cisco.com/sites/default/files/legacy/6/4/3/64346-Document1.pdf. [Accessed: 05-Jan-2015]

[45] 'Schneier on Security: Cisco Harasses Security Researcher'. [Online]. Available: https://www.schneier.com/blog/archives/2005/07/cisco_harasses.html. [Accessed: 12-Jan-2015]

[46] 'KDV Electronics - uClinux Cisco 2500'. [Online]. Available: http://www.kdvelectronics.eu/uClinux-cisco2500/uClinux-cisco2500.html. [Accessed: 04-Jan-2015]

[47] A. Weiss, 'The Open Source WRT54G Story', *WIFI-PLANET*, 08-Nov-2005. [Online]. Available: http://www.wi-fiplanet.com/tutorials/article.php/3562391. [Accessed: 04-Jan-2015]

[48] 'Cisco - LinuxMIPS'. [Online]. Available: http://www.linux-mips.org/wiki/Cisco#The_Boot_ROM_API. [Accessed: 04-Jan-2015]

[49] 'How do I reset the Linksys Wi-Fi Router, E1000 to factory defaults?'. [Online]. Available: http://kb.linksys.com/Linksys/ukp.aspx?pid=80&app=vw&vw=1&login=1&json=1&docid=0021aa1b87a14d83b94350ce576af242_KB_EN_v1.xml. [Accessed: 06-Jan-2015]

[50] J. Claudius, 'Getting Terminal Access to a Cisco Linksys E-1000 - SpiderLabs Anterior', 31-Dec-2012. [Online]. Available: http://blog.spiderlabs.com/2012/12/getting-terminal-access-to-a-cisco-linksys-e-1000.html. [Accessed: 28-Dec-2014]

[51] 'RS232enum'. [Online]. Available: https://github.com/cyphunk/RS232enum. [Accessed: 05-Jan-2015]

[52] 'MAX232', *Wikipedia, the free encyclopedia*. 24-Nov-2014 [Online]. Available: http://en.wikipedia.org/w/index.php?title=MAX232&oldid=635260926. [Accessed: 06-Jan-2015]

[53] 'List of integrated circuit package dimensions - Wikipedia, the free encyclopedia'. [Online]. Available: http://en.wikipedia.org/wiki/List_of_integrated_circuit_package_dimensions. [Accessed: 04-Jan-2015]

[54] 'MCUmall Electronics Inc. A low cost EPROM EEPROM Atmel PIC I2C SPI programmer online store'. [Online]. Available:

http://www.mcumall.com/comersus/store/comersus_viewItem.asp?idProduct=4312. [Accessed: 04-Jan-2015]
[55] 'Xeltek SuperPro 6100 Universal IC Chip Device Programmer'. [Online]. Available: http://www.xeltek.com/universal-programmers/superpro-6100-universal-ic-chip-device-programmer. [Accessed: 04-Jan-2015]
[56] Pomona Electronics, 'Pomona Electronics, PLCC Test clips'. [Online]. Available: http://www.pomonaelectronics.com/pdf/d5515_100.pdf. [Accessed: 03-Jan-2015]
[57] J. Oh, 'Reverse Engineering Flash Memory for Fun and Benefit'. [Online]. Available: https://www.blackhat.com/docs/us-14/materials/us-14-Oh-Reverse-Engineering-Flash-Memory-For-Fun-And-Benefit-WP.pdf. [Accessed: 03-Jan-2015]
[58] 'Draft NIST Special Publication 800-88 Revision 1, Guidelines for Media Sanitization - sp800_88_r1_draft.pdf'. [Online]. Available: http://csrc.nist.gov/publications/drafts/800-88-rev1/sp800_88_r1_draft.pdf. [Accessed: 14-Nov-2014]
[59] P. Gutmann, 'Secure Deletion of Data from Magnetic and Solid-State Memory', *Secure Deletion of Data from Magnetic and Solid-State Memory*, 25-Nov-2014. [Online]. Available: https://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html. [Accessed: 25-Nov-2014]
[60] F. Domke, 'Blackbox JTAG Reverse Engineering', 27-Nov-2009. [Online]. Available: http://events.ccc.de/congress/2009/Fahrplan/attachments/1435_JTAG.pdf. [Accessed: 05-Jan-2015]
[61] *[27C3] (en) JTAG/Serial/FLASH/PCB Embedded Reverse Engineering Tools and Techniques*. 2011 [Online]. Available: https://www.youtube.com/watch?v=8Unisnu-cNo&feature=youtube_gdata_player. [Accessed: 05-Jan-2015]
[62] '27C3: JTAG/Serial/FLASH/PCB Embedded Reverse Engineering Tools and Techniques', 09-Feb-2011. [Online]. Available: http://events.ccc.de/congress/2010/Fahrplan/events/4011.en.html. [Accessed: 05-Jan-2015]
[63] 'PCMCIA Filesystem Compatibility Matrix and Filesystem Information - Cisco'. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/routers/7200-series-routers/6145-pcmciamatrix.html. [Accessed: 04-Jan-2015]
[64] 'Cisco IOS Command Modes'. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf019.html#wp1000905. [Accessed: 07-Jan-2015]
[65] '10 Reasons to Buy SEDs_Sept.2010.pdf'. [Online]. Available: https://www.trustedcomputinggroup.org/files/resource_files/0B942977-1A4B-B294-D0CFD24A431444FF/10%20Reasons%20to%20Buy%20SEDs_Sept.2010.pdf. [Accessed: 27-Dec-2014]
[66] S. Swanson and M. Wei, 'SAFE: Fast, Verifiable Sanitization for SSDs', 13-Oct-2010. [Online]. Available: http://cseweb.ucsd.edu/~swanson/papers/TR-cs2011-0963-Safe.pdf. [Accessed: 27-Dec-2014]
[67] 'Floating-point arithmetic may give inaccurate results in Excel'. [Online]. Available: http://support.microsoft.com/kb/78113. [Accessed: 27-Dec-2014]
[68] L. Råde and B. Westergren, *Mathematics Handbook for Science and Engineering*, Fourth. 1998.
[69] 'Password Recovery Procedure for Cisco Aironet Equipment - Cisco'. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/wireless/aironet-1200-series/9215-pwrec-2.html. [Accessed: 07-Jan-2015]

[70] 'Cisco 1712 Security Access Router', *Cisco*. [Online]. Available: http://cisco.com/c/en/us/products/routers/1712-security-access-router/index.html. [Accessed: 17-Jan-2015]

[71] 'Cisco 1721 and Cisco 1720 Modular Access Routers [Cisco 1700 Series Modular Access Routers] - Cisco Systems'. [Online]. Available: http://www.cisco.com/en/US/products/hw/routers/ps221/products_data_sheet09186a00800920ec.html. [Accessed: 17-Jan-2015]

[72] 'Understanding the Cisco IOS Software'. [Online]. Available: http://www.cisco.com/E-Learning/bulk/public/tac/cim/cib/using_cisco_ios_software/01_understanding_ios.htm. [Accessed: 22-Jan-2015]

[73] Cisco Systems, *Internetworking Troubleshooting Handbook*, 2 edition. Indianapolis, IN: Cisco Press, 2001.

[74] Y. C. Hoong, '- itcertnotes -: The Cisco Router Cookie'. [Online]. Available: http://www.itcertnotes.com/2011/03/cisco-router-cookie.html. [Accessed: 17-Jan-2015]

[75] Catalyst, 'CAT28C256, 256K-Bit Parallel EEPROM'. [Online]. Available: http://ecee.colorado.edu/~mcclurel/Catalyst_Parallel_EEPROM_28C256.pdf. [Accessed: 17-Jan-2015]

[76] Freescale Semiconductor, Inc, 'MPC862 PowerQUICC Integrated Communications Processor Family Reference Manual - MPC862UM.pdf'. [Online]. Available: http://cache.freescale.com/files/product/doc/MPC862UM.pdf. [Accessed: 18-Jan-2015]

[77] 'Data Sheet: MAX 7000 Programmable Logic Device Family - m7000.pdf'. [Online]. Available: http://www.altera.com/literature/ds/m7000.pdf. [Accessed: 23-Jan-2015]

[78] 'Intel 3 Volt Intel StrataFlash Memory 28F128J3A, 28F640J3A 28F320J3A Datasheet'. [Online]. Available: http://www-mtl.mit.edu/Courses/6.111/labkit/datasheets/28F128J3A.pdf. [Accessed: 09-Jan-2015]

[79] 'Cisco Flash File System tool'. [Online]. Available: http://si.org/cffs/. [Accessed: 17-Feb-2015]

[80] S. Howard, 'A Background Debugging Mode Driver Package for Modular Microcontrollers'. [Online]. Available: http://cache.freescale.com/files/microcontrollers/doc/app_note/AN1230.pdf. [Accessed: 10-Feb-2015]

[81] Andromeda Research Labs, 'In-circuit reading and programming of eeproms and microcontrollers'. [Online]. Available: http://www.arlabs.com/incircht.htm. [Accessed: 27-Feb-2015]

[82] Freescale Semiconductor, 'MPC862/857T/857DSL PowerQUICC Family Hardware Specifications - MPC862EC.pdf'. [Online]. Available: http://cache.freescale.com/files/32bit/doc/data_sheet/MPC862EC.pdf#page=1&zoom=auto,-265,792. [Accessed: 28-Feb-2015]

[83] 'Data Sheet: MAX 7000 Programmable Logic Device Family - m7000.pdf'. [Online]. Available: http://www.altera.com/literature/ds/m7000.pdf. [Accessed: 23-Jan-2015]

[84] 'Virtual LAN Security: weaknesses and countermeasures - virtual-lan-security-weaknesses-countermeasures-1090'. [Online]. Available: http://www.sans.org/reading-room/whitepapers/networkdevs/virtual-lan-security-weaknesses-countermeasures-1090. [Accessed: 16-Feb-2015]

[85] 'Catalyst 6500 Release 12.2SX Software Configuration Guide - Layer 2 LAN Ports [Cisco Catalyst 6500 Series Switches]', *Cisco*. [Online]. Available:

http://cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/layer2.html. [Accessed: 16-Feb-2015]

[86] *Black Hat USA 2013 - Fully Arbitrary 802.3 Packet Injection: Maximizing the Ethernet Attack Surface*. 2013 [Online]. Available: https://www.youtube.com/watch?v=j_sqwo1xjIA&feature=youtube_gdata_player. [Accessed: 16-Feb-2015]

[87] 'Understanding VLAN Trunk Protocol (VTP)', *Cisco*. [Online]. Available: http://cisco.com/c/en/us/support/docs/lan-switching/vtp/10558-21.html. [Accessed: 16-Feb-2015]

[88] 'Erase Vlan Data on Cisco Switches - Spiceworks'. [Online]. Available: http://community.spiceworks.com/how_to/47462-erase-vlan-data-on-cisco-switches. [Accessed: 16-Feb-2015]

[89] 'Cisco Lab Erasing Router And Switch Configs – Even VLAN.DAT'. [Online]. Available: http://www.thebryantadvantage.com/CCNA%20CCNP%20Home%20Lab%20Tutorial%20Erasing%20Configurations%20On%20Routers%20And%20Switches.htm. [Accessed: 16-Feb-2015]

[90] 'Resetting Catalyst Switches to Factory Defaults - Cisco'. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/switches/catalyst-2900-xl-series-switches/24328-156.html. [Accessed: 16-Feb-2015]

[91] 'Managing Configuration Files', *Cisco*. [Online]. Available: http://cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf007.html. [Accessed: 06-Apr-2015]

[92] 'HP ProCurve Switch 2600 Series - 59906036-e1.pdf'. [Online]. Available: ftp://ftp.hp.com/pub/networking/software/59906036-e1.pdf. [Accessed: 16-Feb-2015]

[93] 'ispLSI 5128VE Data Sheet - 5128ve.pdf'. [Online]. Available: http://download.siliconexpert.com/pdfs/source/qd/lat/5128ve.pdf. [Accessed: 14-Feb-2015]

[94] Freescale Semiconductor, 'MPC8245 Integrated Processor Reference Manual'. [Online]. Available: http://cache.freescale.com/files/product/doc/MPC8245UM.pdf. [Accessed: 16-Feb-2015]

[95] 'Hidden ProCurve commands — Evil Routers'. [Online]. Available: http://evilrouters.net/2010/04/06/hidden-procurve-commands/. [Accessed: 13-Feb-2015]

[96] Texas Instruments, 'TL16C752B | UART | Interface | Description & parametrics'. [Online]. Available: http://www.ti.com/product/tl16c752b. [Accessed: 17-Feb-2015]

[97] 'Management and Configuration Guide - 59906023-1004-Management-Guide.pdf', 20004-10. [Online]. Available: ftp://ftp.hp.com/pub/networking/software/59906023-1004-Management-Guide.pdf. [Accessed: 13-Feb-2015]

[98] 'am29lv065d_23544c3.book - AM29LV065D_EOL_23544c3.pdf'. [Online]. Available: http://www.spansion.com/Support/Datasheets/AM29LV065D_EOL_23544c3.pdf. [Accessed: 13-Feb-2015]

[99] 'S29GL064A_brief.indd - S29GL064A_overview.pdf'. [Online]. Available: http://www.spansion.com/Support/Related%20Product%20Info/S29GL064A_overview.pdf. [Accessed: 17-Feb-2015]

[100] 'Support – Manuals - HP ProCurve Networking'. [Online]. Available: http://www.hp.com/rnd/support/manuals/2800.htm. [Accessed: 17-Feb-2015]

[101] ProCurve Networking, ‘2600-2800-4100-6108-Management Configuration Guide’. [Online]. Available: http://ftp.hp.com/pub/networking/software/2600-2800-4100-6108-MgmtConfig-Oct2005-59906023.pdf. [Accessed: 17-Feb-2015]

[102] HP, ‘Hardening ProCurve Switches - Hardening_ProCurve_Switches_White_Paper.pdf’. [Online]. Available: http://www.hp.com/rnd/pdfs/Hardening_ProCurve_Switches_White_Paper.pdf. [Accessed: 17-Feb-2015]

[103] ‘Configuring Username and Password Security - 6400-5300-4200-3400-Security-Oct2005-Ch-02-Passwords.pdf’. [Online]. Available: ftp://ftp.hp.com/pub/networking/software/6400-5300-4200-3400-Security-Oct2005-Ch-02-Passwords.pdf. [Accessed: 20-Apr-2015]

[104] Cisco Systems, ‘Reset a Cisco Router to Factory Default Settings - Cisco’, 02-Aug-2006. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-software-releases-123-mainline/46509-factory-default.html. [Accessed: 04-Jan-2015]

[105] ‘Resetting Catalyst Switches to Factory Defaults - Cisco’. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/switches/catalyst-2900-xl-series-switches/24328-156.html. [Accessed: 07-Jan-2015]

Appendices

The appendices contains procedures, console output listings, and source code relevant to this thesis project.

Appendix A. List of Erase procedures

This section contains the list of erase procedures referenced in the documents. Each procedure has a unique name.

ERASE PROCEDURE	**CISCO_IOS_1**
Type:	Vendor Recommendation
Source:	Reset a Cisco Router to Factory Default Settings, (document ID 46509) [104] (method 1) Link
Applies to:	Cisco router running IOS 12.3 Mainline
Prerequisites:	Router CLI is in “enable” mode
Proposed Procedure:	

```
router#configure terminal
router(config)#config-register 0x2102
router(config)#end
router#write erase
router#reload
System configuration has been modified. Save? [yes/no]: n
Proceed with reload? [confirm]
```

Once the router reloads, the System Configuration Dialog appears.

```
         --- System Configuration Dialog ---
Would you like to enter the initial configuration dialog? [yes/no]:
```

“The router is now reset to the original factory defaults.”

Warning: Above procedure is proved unsafe in Section 4.1.8

ERASE PROCEDURE	**CISCO_IOS_2**
Type:	Vendor Recommendation
Source:	Reset a Cisco Router to Factory Default Settings, (document ID 46509) [104] (method 2) Link
Applies to:	Cisco router running IOS 12.3 Mainline
Prerequisites:	Router CLI is in global config mode
Proposed Procedure:	

```
router(config)#config-register 0x2142
router#reload
System configuration has been modified. Save? [yes/no]: no
Proceed with reload? [confirm]
[reload...]
Would you like to enter initial configuration dialog? no

router(config)#config-register 0x2102
router#write memory
```

“The router is now reset to the original factory defaults.” Except warm-reboot and memory-size iomem setting.

Warning: Above procedure is proved unsafe in Section 4.1.8

ERASE PROCEDURE	**HP_2626_CLI**
Type:	Vendor Recommendation
Source:	Management and Configuration Guide [97pp. C–43 Section Restoring the Factory-Default Configuration]
Applies to:	Hp Procurve Switches: 2600 Series, 2600-PWR Series, 2800 Series, 4100gl Series, 6108
Prerequisites:	This command operates at any level *except* the Operator level.
Proposed Procedure:	

```
erase startup-config
```

Vendor declaration:

“Deletes the startup-config file in flash so that the switch will reboot with its factory-default configuration. The erase startup-config command does not clear passwords.”

Warning: Above procedure is proved unsafe in Section 4.2.6

ERASE PROCEDURE	**HP_2626_BUTTON**
Type:	Vendor Recommendation
Source:	Management and Configuration Guide [97pp. C–43 Section Restoring the Factory-Default Configuration]
Applies to:	Hp Procurve Switches: 2600 Series, 2600-PWR Series, 2800 Series, 4100gl Series, 6108
Prerequisites:	None
Proposed Procedure:	

To execute the factory default reset, perform these steps:

1. Using pointed objects, simultaneously press both the Reset and Clear buttons on the front of the switch.
2. Continue to press the Clear button while releasing the Reset button.
3. When the Self Test LED begins to flash, release the Clear button. The switch will then complete its self test and begin operating with the configuration restored to the factory default settings.

Warning: Above procedure it proved unsafe in Section 4.2.6

ERASE PROCEDURE	**HP_PROCURVE_SANITTY**
Type:	Method developed as part of this thesis
Source:	This thesis
Applies to:	J9088A ProCurve Switch 2610 and J4900A ProCurve Switch 2626 Possible other Procurve models too, even though it has not been confirmed.
Prerequisites:	None
Proposed Procedure:	

Perform the HP_2626_BUTTON procedure to bypass the current manager password and allow access to CLI.

Execute the "sanitty.exe sanitize" command

Appendix B. Excel function generating random string markers

```
'adapted from http://www.extendoffice.com/documents/excel/642-excel-generate-
random-string.html#vba

Public Function RandStr(Length As Integer)
'Update 20131107
Dim Rand As String
Application.Volatile
Do
i = i + 1
Randomize
'Rand = Rand & Chr(Int((85) * Rnd + 38))
If (Rnd > 0.5) Then
Rand = Rand & Chr(Int((26) * Rnd + 65))
Else
Rand = Rand & Chr(Int((26) * Rnd + 97))
End If

Loop Until i = Length
RandStr = "MARK" + Rand
End Function
```

Appendix C. CISCO1712 investigations

Output listing 6-1: CISCO1712 HW explore markers injection

```
Router#enable
Router#conf term
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#
Router(config)#snmp-server community MARKWHKMcflpXC rw
Router(config)#
Router(config)#exit
Router#copy running startup
Destination filename [startup-config]?
Building configuration...

*May  9 14:31:24.275: %SYS-5-CONFIG_I: Configured from console by console[OK]
Router#vlan database
Router(vlan)#vtp password MARKlscAlvXimn
Setting device VLAN database password to MARKlscAlvXimn.
Router(vlan)#exit
APPLY completed.
Exiting....
Router#
Router#
Router##verification
Router#show startup-config | include snmp-server
snmp-server community MARKWHKMcflpXC RW
Router#more flash:/vlan.dat
:[^P
^@^@^@^B^B^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^A000000000000^B$Jn
h;in^D^NO'^BU^NMARKlscAlvXimn^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^E^B^B^@^@^B\PPdefault^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^A^A^E\^@^A^@^A^F!^^@^@^@^@^Cj^Ck^@^@^@^@^
@^@^Lfddi-default^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^B^A^E\^Cj^@^A


^^@^@^@^@^@^A^Ck^B^@^@^@^@^@^Rtoken-ring-
default^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^C^A^E\^Ck^@^A
^K^@^@^A^@^Cm^@^A^Cj^B^A^@^@^Ofddinet-
default^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^D^A^E\^Cl^@^A
^L^^A^B^@^@^@^@^@^@^B^@^@^@^@^@
trnet-default^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^E^A^E\^Cm^@^A


^^A^B^@^@^@^@^@^@^B^@^@^@^@^@^B\O@^@^@^@^A^@^@^^B`+T^E^B^@^@^Cj^Ck^B\O^L^@^@^Cj^@^@
^@^P^B\N
^A^A^@^@^D^A^@^@^E^B^@^@^@^A^Ck^B\Nh^@^@^Ck^@^@^@^T^B\P`^A^A^@^@^B^A^@^A^D^A^Cm^E^B
^@^@^@^A^Cj^B\M$^@^@^Cl^@^@^^B`*\^B^A^@^A^C^A^@^B^@^@^@^@^@^@^Cm^@^@^^B`+^L^B^A^@^A
^C^A^@^B
Router#
```

Output listing 6-2: CISCO1712 Rommon PRIV mode command set

```
rommon 3 > ?
addrloop            walk 1 thru range of addresses
alias               set and display aliases command
alter               alter locations in memory
berrscan            scan range of addresses for bus errors
boot                boot up an external process
break               set/show/clear the breakpoint
call                call a subroutine at address with converted hex args
cat                 concatenate files
checksum            checksum a block of memory
clrerr              clear the error log
compare             compare two blocks of memory
confreg             configuration register utility
cont                continue executing a downloaded image
context             display the context of a loaded image
cookie              display contents of cookie PROM in hex
cpu                 cpu / system information and control
dev                 list the device table
dir                 list files in file system
dis                 display instruction stream
dnld                serial download a program module
dump                display a block of memory
echo                monitor echo command
errlog              display the error log
fdump               file dump utility
fill                fill a block of memory
flash               flash services command
frame               print out a selected stack frame
help                monitor builtin command help
history             monitor command history
ifill               fill a block of memory w/incrementing pattern
initfs              re-initialize the file system access structures
jump                call a subroutine at address with argc/argv
launch              launch a downloaded image
memdebug            write/read/verify scope loop
meminfo             main memory information
memloop             write or read scope loop
memtest             simple memory test
menu                main diagnostic menu
move                move a block of memory
repeat              repeat a monitor command
reset               system reset
set                 display the monitor variables
sleep               millisecond sleep command
speed               timed performance loop
stack               produce a stack trace
sync                write monitor environment to NVRAM
sysret              print out info from last system return
tcal                timer calibration test
tftpdnld            tftp image download
tscope              timer scope loop
unalias             unset an alias
```

```
unset               unset a monitor variable
watchdog            test watchdog rebooting of the box
xmodem              x/ymodem image download
rommon 4 > menu


      Main Diagnostic Menu
a: alter diag flags
b: basic utilities
c: do all diags in this menu
d: do group of diags in this menu
e: bus error test
f: monitor image checksum test
g: internal interrupt test
h: ip state test
i: timer interrupt test
j: size main memory
k: main memory test
l: main memory refresh test
m: flash memory test
n: aux loopback test
o: aux port interrupt test
p: data cache test
q: mpc862 test
r: nvram test
x: return to previous menu
FLAGS: Continuous OFF  Stop on error OFF  Loop on error OFF  Quiet mode OFF

enter Main Diagnostic Menu item > b


      Diagnostic Utilities Menu
a: alter memory
b: bus error scan
c: compare memory block
d: display memory
e: move memory block
f: fill memory
g: memory test
h: memory read or write loop
i: memory debug loop
j: address loop
k: console break interrupt test
l: system reset
m: AUX port echo test
n: serial cookie utility
o: show 862 registers
x: return to previous menu

enter Diagnostic Utilities Menu item >
```

Output listing 6-3: CISCO1712 PRIV NVRAM Dump (from memory offset 0x68000000) truncated at 0xBF0 bytes.

```
rommon 16 > dump 0x68000000 0xfff
68000000  8d74 0101 0013 7f3b df68 3300 01ff 0448 .t.....;.h3....H
68000010  0011 0000 0000 0000 0000 464f 4309 1030 ..........FOC..0
68000020  5737 5001 0100 0000 0000 00ff ffff 5804 W7P...........X.
68000030  4923 0901 ffff ffff ffff ffff ffff ffff I#..............
68000040  ffff ffff ffff ffff ffff ffff ffff ffff ................
68000050  ffff ffff ffff ffff ffff ffff ffff ffff ................
68000060  ffff ffff ffff ffff ffff ffff ffff ffff ................
68000070  ffff ffff ffff ffff ffff ffff ffff ffff ................
68000080  ffff 1342 ffff ffff 0000 0000 0000 0000 ...B............
68000090  2102 defd feed face 0000 0000 0000 000a !...............
680000a0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680000b0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680000c0  0000 0000 0000 0000 0000 0013 0000 0000 ................
680000d0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680000e0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680000f0  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000100  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000110  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000120  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000130  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000140  0000 0000 0000 0000 5053 3100 726f 6d6d ........PS1.romm
68000150  6f6e 2021 203e 2000 5446 5450 5f43 4845 on ! > .TFTP_CHE
68000160  434b 5355 4d00 3100 5341 5645 5f32 5f52 CKSUM.1.SAVE_2_R
68000170  5453 0031 333a 3536 3a33 3920 5554 4320 TS.13:56:39 UTC 
68000180  5475 6520 4d61 7920 3920 3230 3036 003f Tue May 9 2006.?
68000190  0032 3800 524f 4d5f 5045 5253 4953 5445 .28.ROM_PERSISTE
680001a0  4e54 5f55 5443 0031 3134 3731 3833 3337 NT_UTC.114718337
680001b0  3500 5245 545f 325f 5254 5300 0052 414e 5.RET_2_RTS..RAN
680001c0  444f 4d5f 4e55 4d00 3433 3133 3838 3338 DOM_NUM.43138838
680001d0  3100 4253 4900 3000 5245 545f 325f 5243 1.BSI.0.RET_2_RC
680001e0  414c 5453 0000 0000 524f 4d5f 5045 5253 ALTS....ROM_PERS
680001f0  4953 5445 4e54 5f55 5443 0031 3134 3731 ISTENT_UTC.11471
68000200  3833 3337 3500 0038 3132 3633 0042 5349 83375..81263.BSI
68000210  0030 0052 4554 5f32 5f52 4341 4c54 5300 .0.RET_2_RCALTS.
68000220  0053 4156 455f 325f 5254 5300 3133 3a35 .SAVE_2_RTS.13:5
68000230  363a 3339 2055 5443 2054 7565 204d 6179 6:39 UTC Tue May
68000240  2039 2032 3030 3600 3f00 3000 0053 0000  9 2006.?.0..S..
68000250  0045 5200 3130 2e31 2e31 2e31 0000 0000 .ER.10.1.1.1....
68000260  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000270  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000280  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000290  0000 0000 0000 0000 0000 0000 0000 0000 ................
680002a0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680002b0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680002c0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680002d0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680002e0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680002f0  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000300  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000310  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000320  0000 0000 0000 0000 0000 0000 0000 0000 ................
```

```
68000330  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000340  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000350  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000360  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000370  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000380  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000390  0000 0000 0000 0000 0000 0000 0000 0000 ................
680003a0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680003b0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680003c0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680003d0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680003e0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680003f0  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000400  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000410  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000420  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000430  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000440  0000 0000 0000 0000 7200 7265 7065 6174 ........r.repeat
68000450  0068 0068 6973 746f 7279 003f 0068 656c .h.history.?.hel
68000460  7000 6200 626f 6f74 006c 7300 6469 7200 p.b.boot.ls.dir.
68000470  6900 7265 7365 7400 6b00 7374 6163 6b00 i.reset.k.stack.
68000480  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000490  0000 0000 0000 0000 0000 0000 0000 0000 ................
680004a0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680004b0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680004c0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680004d0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680004e0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680004f0  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000500  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000510  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000520  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000530  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000540  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000550  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000560  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000570  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000580  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000590  0000 0000 0000 0000 0000 0000 0000 0000 ................
680005a0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680005b0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680005c0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680005d0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680005e0  0000 0000 0000 0000 0000 0000 0000 0000 ................
680005f0  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000600  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000610  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000620  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000630  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000640  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000650  0000 0000 0000 0000 4331 3730 3020 536f ........C1700 So
68000660  6674 7761 7265 2028 4331 3730 302d 4b39 ftware (C1700-K9
68000670  4f33 5359 372d 4d29 2c20 5665 7273 696f O3SY7-M), Versio
```

```
68000680  6e20 3132 2e33 2831 3129 5439 2c20 5245 n 12.3(11)T9, RE
68000690  4c45 4153 4520 534f 4654 5741 5245 2028 LEASE SOFTWARE (
680006a0  6663 3329 0a54 6563 686e 6963 616c 2053 fc3).Technical S
680006b0  7570 706f 7274 3a20 6874 7470 3a2f 2f77 upport: http://w
680006c0  7777 2e63 6973 636f 2e63 6f6d 2f74 6563 ww.cisco.com/tec
680006d0  6873 7570 706f 7274 0a43 6f6d 7069 6c65 hsupport.Compile
680006e0  6420 5475 6520 3133 2d44 6563 2d30 3520 d Tue 13-Dec-05
680006f0  3035 3a32 3020 6279 2063 6361 690a 496d 05:20 by ccai.Im
68000700  6167 6520 7465 7874 2d62 6173 653a 2030 age text-base: 0
68000710  7838 3030 3038 3136 432c 2064 6174 612d x8000816C, data-
68000720  6261 7365 3a20 3078 3831 3543 3544 3743 base: 0x815C5D7C
68000730  0a0a 0000 0000 0000 0000 0000 0000 0000 ................
68000740  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000750  0000 0000 0000 0000 0000 bdef 8d74 0101 ............t..
68000760  0013 7f3b df68 3300 01ff 0448 0011 0000 ...;.h3....H....
68000770  0000 0000 0000 464f 4309 1030 5737 5001 ......FOC..0W7P.
68000780  0100 0000 0000 00ff ffff 5804 4923 0901 ..........X.I#..
68000790  ffff ffff ffff ffff ffff ffff ffff ffff ................
680007a0  ffff ffff ffff ffff ffff ffff ffff ffff ................
680007b0  ffff ffff ffff ffff ffff ffff ffff ffff ................
680007c0  ffff ffff ffff ffff ffff ffff ffff ffff ................
680007d0  ffff ffff ffff ffff ffff ffff ffff 1342 ...............B
680007e0  ffff ffff ffff ffff ffff ffff ffff ffff ................
680007f0  ffff ffff ffff ffff ffff ffff ffff ffff ................
68000800  ffff ffff ffff f0a5 abcd 0001 6a0e 0c03 ............j...
68000810  0000 0024 826d b87c 0000 0328 0000 0000 ...$.m.|...(....
68000820  0000 0000 0000 0000 0000 0000 0a21 0a76 .............!.v
68000830  6572 7369 6f6e 2031 322e 330a 7365 7276 ersion 12.3.serv
68000840  6963 6520 7469 6d65 7374 616d 7073 2064 ice timestamps d
68000850  6562 7567 2064 6174 6574 696d 6520 6d73 ebug datetime ms
68000860  6563 0a73 6572 7669 6365 2074 696d 6573 ec.service times
68000870  7461 6d70 7320 6c6f 6720 6461 7465 7469 tamps log dateti
68000880  6d65 206d 7365 630a 6e6f 2073 6572 7669 me msec.no servi
68000890  6365 2070 6173 7377 6f72 642d 656e 6372 ce password-encr
680008a0  7970 7469 6f6e 0a21 0a68 6f73 746e 616d yption.!.hostnam
680008b0  6520 526f 7574 6572 0a21 0a62 6f6f 742d e Router.!.boot-
680008c0  7374 6172 742d 6d61 726b 6572 0a62 6f6f start-marker.boo
680008d0  742d 656e 642d 6d61 726b 6572 0a21 0a21 t-end-marker.!.!
680008e0  0a6d 6d69 2070 6f6c 6c69 6e67 2d69 6e74 .mmi polling-int
680008f0  6572 7661 6c20 3630 0a6e 6f20 6d6d 6920 erval 60.no mmi
68000900  6175 746f 2d63 6f6e 6669 6775 7265 0a6e auto-configure.n
68000910  6f20 6d6d 6920 7076 630a 6d6d 6920 736e o mmi pvc.mmi sn
68000920  6d70 2d74 696d 656f 7574 2031 3830 0a6e mp-timeout 180.n
68000930  6f20 6161 6120 6e65 772d 6d6f 6465 6c0a o aaa new-model.
68000940  6970 2073 7562 6e65 742d 7a65 726f 0a21 ip subnet-zero.!
68000950  0a21 0a21 0a21 0a69 7020 6365 660a 6970 .!.!.!.ip cef.ip
68000960  2069 7073 2070 6f20 6d61 782d 6576 656e  ips po max-even
68000970  7473 2031 3030 0a6e 6f20 6674 702d 7365 ts 100.no ftp-se
68000980  7276 6572 2077 7269 7465 2d65 6e61 626c rver write-enabl
68000990  650a 210a 210a 210a 210a 2120 0a6e 6f20 e.!.!.!.!.! .no
680009a0  6372 7970 746f 2069 7361 6b6d 7020 6363 crypto isakmp cc
680009b0  6d0a 210a 210a 210a 696e 7465 7266 6163 m.!.!.!.interfac
680009c0  6520 4252 4930 0a20 6e6f 2069 7020 6164 e BRI0. no ip ad
```

```
680009d0  6472 6573 730a 2073 6875 7464 6f77 6e0a dress. shutdown.
680009e0  210a 696e 7465 7266 6163 6520 4661 7374 !.interface Fast
680009f0  4574 6865 726e 6574 300a 2069 7020 6164 Ethernet0. ip ad
68000a00  6472 6573 7320 6468 6370 0a20 6475 706c dress dhcp. dupl
68000a10  6578 2061 7574 6f0a 2073 7065 6564 2061 ex auto. speed a
68000a20  7574 6f0a 210a 696e 7465 7266 6163 6520 uto.!.interface
68000a30  4661 7374 4574 6865 726e 6574 310a 210a FastEthernet1.!.
68000a40  696e 7465 7266 6163 6520 4661 7374 4574 interface FastEt
68000a50  6865 726e 6574 320a 210a 696e 7465 7266 hernet2.!.interf
68000a60  6163 6520 4661 7374 4574 6865 726e 6574 ace FastEthernet
68000a70  330a 210a 696e 7465 7266 6163 6520 4661 3.!.interface Fa
68000a80  7374 4574 6865 726e 6574 340a 210a 696e stEthernet4.!.in
68000a90  7465 7266 6163 6520 566c 616e 310a 206e terface Vlan1. n
68000aa0  6f20 6970 2061 6464 7265 7373 0a21 0a69 o ip address.!.i
68000ab0  7020 636c 6173 736c 6573 730a 6e6f 2069 p classless.no i
68000ac0  7020 6874 7470 2073 6572 7665 720a 6e6f p http server.no
68000ad0  2069 7020 6874 7470 2073 6563 7572 652d  ip http secure-
68000ae0  7365 7276 6572 0a21 0a21 0a21 0a73 6e6d server.!.!.!.snm
68000af0  702d 7365 7276 6572 2063 6f6d 6d75 6e69 p-server communi
68000b00  7479 204d 4152 4b57 484b 4d63 666c 7058 ty MARKWHKMcflpX
68000b10  4320 5257 0a21 0a21 0a63 6f6e 7472 6f6c C RW.!.!.control
68000b20  2d70 6c61 6e65 0a21 0a21 0a6c 696e 6520 -plane.!.!.line
68000b30  636f 6e20 300a 6c69 6e65 2061 7578 2030 con 0.line aux 0
68000b40  0a6c 696e 6520 7674 7920 3020 340a 210a .line vty 0 4.!.
68000b50  656e 640a fedc 0001 0000 035c 826d b8ba end........\.m..
68000b60  0000 002e 0a6b 6572 6265 726f 7320 7061 .....kerberos pa
68000b70  7373 776f 7264 200a 736e 6d70 2d73 6572 ssword .snmp-ser
68000b80  7665 7220 6863 2070 6f6c 6c20 300a 656e ver hc poll 0.en
68000b90  640a 726f 7320 7061 7373 776f 7264 200a d.ros password .
68000ba0  736e 6d70 2d73 6572 7665 7220 6863 2070 snmp-server hc p
68000bb0  6f6c 6c20 300a 656e 640a 0000 0000 0000 oll 0.end.......
68000bc0  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000bd0  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000be0  0000 0000 0000 0000 0000 0000 0000 0000 ................
68000bf0  0000 0000 0000 0000 0000 0000 0000 0000 ................
<snip>
```

Output listing 6-4: CISCO1712 cookie fields

```
rommon 18 > cookie

View/alter bytes of serial cookie by field --
Input hex byte(s) or: CR -> skip field; ? -> list values
byte  0x00       - version: 01
                          >

byte  0x01       - vendor (Recommended value: 01): 01
                                                 >

bytes 0x02-0x07  - ethernet Hw address: 00 13 7f 3b df 68
                                      >

byte  0x08-0x08  - processor (Recommended value: 33): 33
                                                    >

byte  0x09-0x09  - nvram size (Recommended values: 32K - 00, 64K - 01): 00
                                                                      >

byte  0x0a-0x0a  - cpu speed (Recommended value: 50Mhz - 01): 01
                                                            >

byte  0x0b-0x0b  - unused: ff
                         >

bytes 0x0c-0x0d  - on board PM ID (Recommended value for 1720: 00 b2): 04 48
                                                                     >

bytes 0x0e-0x0f  - mac address allocated: 00 11
                                        >

bytes 0x10-0x17: 00 00 00 00 00 00 00 00
               >

bytes 0x18-0x22: 46 4f 43 09 10 30 57 37 50 01 01
               >

bytes 0x23-0x24  - deviation: 00 00
                            >

bytes 0x25-0x2c: 00 00 00 00 ff ff ff 58
               >

bytes 0x2d-0x2d  - board config (Recommended value: 04): 04
                                                       >

bytes 0x2e-0x37: 49 23 09 01 ff ff ff ff ff ff
               >

bytes 0x38-0x3f: ff ff ff ff ff ff ff ff
               >
```

```
bytes 0x40-0x47: ff ff ff ff ff ff ff ff
               >

bytes 0x48-0x4f: ff ff ff ff ff ff ff ff
               >

bytes 0x50-0x57: ff ff ff ff ff ff ff ff
               >

bytes 0x58-0x5f: ff ff ff ff ff ff ff ff
               >

bytes 0x60-0x67: ff ff ff ff ff ff ff ff
               >

bytes 0x68-0x6f: ff ff ff ff ff ff ff ff
               > ?

<CR> to skip field; otherwise, enter 8 byte(s)
bytes 0x68-0x6f: ff ff ff ff ff ff ff ff
               >

bytes 0x70-0x77: ff ff ff ff ff ff ff ff
               >

bytes 0x78-0x7f: ff ff ff ff ff ff ff ff
               >
rommon 19 >
```

Output listing 6-5: Rommon "show 862 registers"

```
      Diagnostic Utilities Menu
a: alter memory
b: bus error scan
c: compare memory block
d: display memory
e: move memory block
f: fill memory
g: memory test
h: memory read or write loop
i: memory debug loop
j: address loop
k: console break interrupt test
l: system reset
m: AUX port echo test
n: serial cookie utility
o: show 862 registers
x: return to previous menu

enter Diagnostic Utilities Menu item > o

MPC862 Register Dump: Registers at 0xff000000

SIU - System Interface Unit :
-----------------------------
              siu_mcr : 0x00230440
            siu_sypcr : 0xffffff88
              siu_swt : 0xffff0000
             siu_swsr : 0x00000000
           siu_sipend : 0x00000000
         siu_sienmask : 0x20000000
             siu_siel : 0x00000000
            siu_sivec : 0x3c000000
             siu_tesr : 0x00002030
            sdma_sdcr : 0x00000001

PCMCIA :
--------
          pcmcia_pbr0 : 0x00000001
          pcmcia_por0 : 0x80030044
          pcmcia_pbr1 : 0x00000014
          pcmcia_por1 : 0x00000010
          pcmcia_pbr2 : 0x00000000
          pcmcia_por2 : 0x80002444
          pcmcia_pbr3 : 0x00000000
          pcmcia_por3 : 0x88042000
          pcmcia_pbr4 : 0x00800000
          pcmcia_por4 : 0x00012a6c
          pcmcia_pbr5 : 0x00000000
          pcmcia_por5 : 0x000c1200
          pcmcia_pbr6 : 0x00000000
          pcmcia_por6 : 0x00000240
          pcmcia_pbr7 : 0x20000004
          pcmcia_por7 : 0x00000610
         pcmcia_pgcra : 0x00000000
         pcmcia_pgcrb : 0x00000000
          pcmcia_pscr : 0xfe70fe40
          pcmcia_pipr : 0xff00ff00
           pcmcia_per : 0x00000000

MEMC - Memory Controller :
--------------------------
             memc_br0 : 0xfff00401
             memc_or0 : 0xfff005a6
             memc_br1 : 0x00000081
             memc_or1 : 0x7e000600
             memc_br2 : 0x040000c1
             memc_or2 : 0x7f000600
             memc_br3 : 0x050000c1
             memc_or3 : 0x7f000600
             memc_br4 : 0x02000081
             memc_or4 : 0x7e000600
             memc_br5 : 0x00000000
             memc_or5 : 0x00000000
             memc_br6 : 0x60000901
             memc_or6 : 0xfe000190
             memc_br7 : 0x68000401
             memc_or7 : 0xfff001a8
             memc_mar : 0x00000088
             memc_mcr : 0x4080003f
            memc_mamr : 0x0c804111
            memc_mbmr : 0x06804111
           memc_mstat : 0x00000000
           memc_mptpr : 0x00000400
             memc_mdr : 0xfffffc05

SIMT - System Integration Timers :
----------------------------------
           simt_tbscr : 0x00000001
         simt_tbreff0 : 0x7d6b53ea
         simt_tbreff1 : 0xf5fbe112
           simt_rtcsc : 0x00000080
             simt_rtc : 0x5c3450ab
           simt_rtsec : 0x42d80000
           simt_rtcal : 0xca7d41ce
           simt_piscr : 0x00000000
            simt_pitc : 0xfdde0000
            simt_pitr : 0xdfff0000

CLKR - Clocks and Reset :
-------------------------
            clkr_sccr : 0x03820000
          clkr_plprcr : 0x001050c0
             clkr_rsr : 0x00000000

SIMTK - System Integration Timer Keys :
```

```
---------------------------------------
         simtk_tbscrk : 0x00010000
       simtk_tbreff0k : 0x7d6b53ea
       simtk_tbreff1k : 0xf5fbe112
            simtk_tbk : 0x0001cf53
         simtk_rtcsck : 0x00800000
           simtk_rtck : 0x5c3450ab
         simtk_rtseck : 0x42d80000
         simtk_rtcalk : 0xca7d41ce
         simtk_piscrk : 0x00000000
          simtk_pitck : 0xfdde0000

CLKRK - Clocks and Reset Keys :
-------------------------------
          clkrk_sccrk : 0x03820000
        clkrk_plprcrk : 0x001050c0
           clkrk_rsrk : 0x00000000

I2C :
-----
            i2c_i2mod : 0x00000000
            i2c_i2add : 0x00000000
            i2c_i2brg : 0x000000ff
            i2c_i2com : 0x00000000
            i2c_i2cer : 0x00000000
            i2c_i2cmr : 0x00000000

DMA :
-----
             dma_sdar : 0x3c67ea42
             dma_sdsr : 0x00000000
             dma_sdmr : 0x00000000
            dma_idsr1 : 0x00000000
            dma_idmr1 : 0x00000000
            dma_idsr2 : 0x00000000
            dma_idmr2 : 0x00000000

CPIC - CPM Interrupt Controller :
---------------------------------
            cpic_civr : 0x00000000
            cpic_cicr : 0x00007f80
            cpic_cipr : 0x00000040
            cpic_cimr : 0x00000000
            cpic_cisr : 0x00000000

PIO - Parallel I/O :
--------------------
            pio_padir : 0x00000000
            pio_papar : 0x00000000
            pio_paodr : 0x00000000
            pio_padat : 0x0000ffff
            pio_pcdir : 0x00000000
            pio_pcpar : 0x00000000
```

```
             pio_pcso : 0x00000000
            pio_pcdat : 0x00000dfe
            pio_pcint : 0x00000000
            pio_pddir : 0x00001fff
            pio_pdpar : 0x00001fff
            pio_pddat : 0x00001cc8

TMR - CPM Timers :
------------------
             tmr_tgcr : 0x00000000
             tmr_tmr1 : 0x00000000
             tmr_tmr2 : 0x00000000
             tmr_trr1 : 0x0000ffff
             tmr_trr2 : 0x0000ffff
             tmr_tcr1 : 0x00000000
             tmr_tcr2 : 0x00000000
             tmr_tcn1 : 0x00000000
             tmr_tcn2 : 0x00000000
             tmr_tmr3 : 0x00000000
             tmr_tmr4 : 0x00000000
             tmr_trr3 : 0x0000ffff
             tmr_trr4 : 0x0000ffff
             tmr_tcr3 : 0x00000000
             tmr_tcr4 : 0x00000000
             tmr_tcn3 : 0x00000000
             tmr_tcn4 : 0x00000000
             tmr_ter1 : 0x00000000
             tmr_ter2 : 0x00000000
             tmr_ter3 : 0x00000000
             tmr_ter4 : 0x00000000

CP - Communications Processor :
-------------------------------
                cp_cr : 0x00000000
              cp_rccr : 0x00000000
              cp_rmds : 0x00000000
              cp_rmdr : 0x00000000
             cp_rctr1 : 0x00000000
             cp_rctr2 : 0x00000000
             cp_rctr3 : 0x00000000
             cp_rctr4 : 0x00000000
              cp_rter : 0x00000000
              cp_rtmr : 0x00000000

BRG - Baud Rate Generator :
---------------------------
            brg_brgc1 : 0x00000000
            brg_brgc2 : 0x00000000
            brg_brgc3 : 0x00000000
            brg_brgc4 : 0x00000000

SCC[1] - Serial Communications Controller 1 :
---------------------------------------------
```

```
 scc[index].scc_gsmrl : 0x00000000
 scc[index].scc_gsmrh : 0x00000000
  scc[index].scc_psmr : 0x00000000
  scc[index].scc_todr : 0x00000000
   scc[index].scc_dsr : 0x00007e7e
  scc[index].scc_scce : 0x00000000
  scc[index].scc_sccm : 0x00000000
  scc[index].scc_sccs : 0x00000000

SCC[2] - Serial Communications Controller 2 :
---------------------------------------------
 scc[index].scc_gsmrl : 0x00000000
 scc[index].scc_gsmrh : 0x00000000
  scc[index].scc_psmr : 0x00000000
  scc[index].scc_todr : 0x00000000
   scc[index].scc_dsr : 0x00007e7e
  scc[index].scc_scce : 0x00000000
  scc[index].scc_sccm : 0x00000000
  scc[index].scc_sccs : 0x00000000

SCC[3] - Serial Communications Controller 3 :
---------------------------------------------
 scc[index].scc_gsmrl : 0x00000000
 scc[index].scc_gsmrh : 0x00000000
  scc[index].scc_psmr : 0x00000000
  scc[index].scc_todr : 0x00000000
   scc[index].scc_dsr : 0x00007e7e
  scc[index].scc_scce : 0x00000000
  scc[index].scc_sccm : 0x00000000
  scc[index].scc_sccs : 0x00000000

SCC[4] - Serial Communications Controller 4 :
---------------------------------------------
 scc[index].scc_gsmrl : 0x00000000
 scc[index].scc_gsmrh : 0x00000000
  scc[index].scc_psmr : 0x00000000
  scc[index].scc_todr : 0x00000000
   scc[index].scc_dsr : 0x00007e7e
  scc[index].scc_scce : 0x00000000
  scc[index].scc_sccm : 0x00000000
  scc[index].scc_sccs : 0x00000000

SMC[1] - Serial Management Controller 1 :
-----------------------------------------
 smc_regs[index].smc_smcmr : 0x00000000
 smc_regs[index].smc_smce : 0x00000000
 smc_regs[index].smc_smcm : 0x00000000

SMC[2] - Serial Management Controller 2 :
-----------------------------------------
 smc_regs[index].smc_smcmr : 0x00000000
 smc_regs[index].smc_smce : 0x00000000
 smc_regs[index].smc_smcm : 0x00000000
```

```
SPI - Serial Peripheral Interface :
-----------------------------------
           spi_spmode : 0x00000000
             spi_spie : 0x00000000
             spi_spim : 0x00000000
            spi_spcom : 0x00000000

PIP - Parallel Interface Port :
-------------------------------
             pip_pipc : 0x00000000
             pip_ptpr : 0x00000000
            pip_pbdir : 0x00000001
            pip_pbpar : 0x00000000
            pip_pbodr : 0x00000000
            pip_pbdat : 0x0003fffe

SI - Serial Interface :
-----------------------
            si_simode : 0x00000000
             si_sigmr : 0x00000000
             si_sistr : 0x00000000
             si_sicmr : 0x00000000
              si_sicr : 0x00000000
              si_sirp : 0x00000000

SI_SIRAM - Serial Interface Routing RAM :
-----------------------------------------
ff000c00  8e66 0000 dd0e 0000 f54a 0000 e0e3 0000 .f.......J......
ff000c10  f996 0000 eee4 0000 503a 0000 27e8 0000 ........P:..'...
ff000c20  f84d 0000 c3fa 0000 f31d 0000 6830 0000 .M..........h0..
ff000c30  952d 0000 f1fc 0000 2b30 0000 30cb 0000 .-......+0..0...
ff000c40  aba2 0000 7fbd 0000 7af3 0000 56d0 0000 ........z...V...
ff000c50  dc69 0000 f41a 0000 b60d 0000 fc33 0000 .i...........3..
ff000c60  9025 0000 b3ed 0000 a006 0000 9cf3 0000 .%..............
ff000c70  9a22 0000 73a0 0000 beb4 0000 205e 0000 ."..s....... ^..
ff000c80  ae66 0000 e823 0000 b856 0000 f0f1 0000 .f...#...V......
ff000c90  0c42 0000 360b 0000 f912 0000 89e4 0000 .B..6...........
ff000ca0  0b5d 0000 357f 0000 8392 0000 b615 0000 .]..5...........
ff000cb0  dcf3 0000 957d 0000 bf1e 0000 c26b 0000 .....}.......k..
ff000cc0  a6ac 0000 c1ec 0000 f5eb 0000 1fce 0000 ................
ff000cd0  9f76 0000 c55f 0000 7dc6 0000 c038 0000 .v..._..}....8..
ff000ce0  538a 0000 c982 0000 f46d 0000 78c8 0000 S........m..x...
ff000cf0  ddd1 0000 79f0 0000 6990 0000 861c 0000 ....y...i.......

FEC - Fast Ethernet Controller :
--------------------------------
          fec_addr_low : 0x98316ebd
         fec_addr_high : 0x00002b38
   fec_hash_table_high : 0x6cc433bb
    fec_hash_table_low : 0xe4426bc7
       fec_r_des_start : 0x82085440
       fec_x_des_start : 0xce3f7233
       fec_r_buff_size : 0x6c85d555
            fec_ecntrl : 0x20000006
            fec_ievent : 0x00000000
             fec_imask : 0x00000000
              fec_ivec : 0x00000000
      fec_r_des_active : 0x00000000
      fec_x_des_active : 0x00000000
          fec_mii_data : 0x60460100
         fec_mii_speed : 0x00000028
           fec_r_bound : 0x00000600
          fec_r_fstart : 0x00000500
          fec_x_fstart : 0x00000440
          fec_fun_code : 0x60000000
           fec_r_cntrl : 0x00000000
            fec_r_hash : 0x370005ee
           fec_x_cntrl : 0x00000000

Special Purpose Registers :
---------------------------
                  cmpa : 0xcffbdfbc
                  cmpb : 0xecdfff7c
                  cmpc : 0xdf9ebf7c
                  cmpd : 0xef5ffffc
                   icr : 0x12200000
                   der : 0x00000000
                counta : 0xb4fc0000
                countb : 0x75f90000
                  cmpe : 0x98d0308a
                  cmpf : 0x205081d9
                  cmpg : 0xe934b88b
                  cmph : 0x9db1bcd9
                lctrl1 : 0x00000000
                lctrl2 : 0x00000000
                 ictrl : 0x00000007
                   bar : 0x0d5faff8
                  dpdr : 0x777a8277
                  dpir : 0x20000000
                  immr : 0xff000700
                ic_cst : 0x80000001
                ic_adr : 0x74220000
                ic_dat : 0xfff13285
                dc_cst : 0x00000000
                dc_adr : 0x00001ff0
                dc_dat : 0x53471800
                mi_ctr : 0x00000000
                 mi_ap : 0x44034b32
                mi_epn : 0x00034200
                mi_twc : 0x00000129
                mi_rpn : 0x4620d40c
              mi_dbcam : 0x00000e10
             mi_dbram0 : 0x90008110
             mi_dbram1 : 0x00000010
                md_ctr : 0x04000000
               m_casid : 0x0000000f
                 md_ap : 0x88600002
                md_epn : 0x22a45802
                 m_twb : 0xfee4b228
                md_twc : 0xc279d114
                md_rpn : 0x8eff33dd
                  m_tw : 0x8fdfef73
              md_dbcam : 0x72f693ef
             md_dbram0 : 0x08080c0c

 enter Diagnostic Utilities Menu item >
```

Output listing 6-6: CISCO1712 MPC862 memory controller banks

```
****Memory Control Register # 0: Base Register 0xfff00401 Option Register
0xfff005a6
BA, Baseaddress:        0x1ffe0   11111111111100000
MA, Maskaddress:        0x1ffe0   11111111111100000
AT, Address Type:       0
PS, Portsize    BIT8
Parity:      false
Write Protect:          false
MS, Machine Select:     GPCM
Reserved (should be 0): 0
Valid:       true

****Memory Control Register # 1: Base Register 0x81 Option Register 0x7e000600
BA, Baseaddress:        0x00
MA, Maskaddress:        0xfc00    1111110000000000
AT, Address Type:       0
PS, Portsize    BIT32
Parity:      false
Write Protect:          false
MS, Machine Select:     UPMA
Reserved (should be 0): 0
Valid:       true

****Memory Control Register # 2: Base Register 0x40000c1 Option Register 0x7f000600
BA, Baseaddress:        0x800     100000000000
MA, Maskaddress:        0xfe00    1111111000000000
AT, Address Type:       0
PS, Portsize    BIT32
Parity:      false
Write Protect:          false
MS, Machine Select:     UPMB
Reserved (should be 0): 0
Valid:       true

****Memory Control Register # 3: Base Register 0x50000c1 Option Register 0x7f000600
BA, Baseaddress:        0xa00     101000000000
MA, Maskaddress:        0xfe00    1111111000000000
AT, Address Type:       0
PS, Portsize    BIT32
Parity:      false
Write Protect:          false
MS, Machine Select:     UPMB
Reserved (should be 0): 0
Valid:       true

****Memory Control Register # 4: Base Register 0x2000081 Option Register 0x7e000600
BA, Baseaddress:        0x400     10000000000
MA, Maskaddress:        0xfc00    1111110000000000
AT, Address Type:       0
PS, Portsize    BIT32
Parity:      false
Write Protect:          false
```

```
MS, Machine Select:     UPMA
Reserved (should be 0): 0
Valid:       true

****Memory Control Register # 5: Base Register 0x0 Option Register 0x0
BA, Baseaddress:        0x00
MA, Maskaddress:        0x00
AT, Address Type:       0
PS, Portsize    BIT32
Parity:      false
Write Protect:          false
MS, Machine Select:     GPCM
Reserved (should be 0): 0
Valid:       false

****Memory Control Register # 6: Base Register 0x60000901 Option Register
0xfe000190
BA, Baseaddress:        0xc000    1100000000000000
MA, Maskaddress:        0x1fc00   11111110000000000
AT, Address Type:       0
PS, Portsize    BIT16
Parity:      false
Write Protect:          true
MS, Machine Select:     GPCM
Reserved (should be 0): 0
Valid:       true

****Memory Control Register # 7: Base Register 0x68000401 Option Register
0xfff001a8
BA, Baseaddress:        0xd000    1101000000000000
MA, Maskaddress:        0x1ffe0   11111111111100000
AT, Address Type:       0
PS, Portsize    BIT8
Parity:      false
Write Protect:          false
MS, Machine Select:     GPCM
Reserved (should be 0): 0
Valid:       true
```

Output listing 6-7: CISCO1712 marker locations in NVRAM using “show memory” command

```
68000000: 8D740101 00136073 40EB3300 01FF0448  .t....`s@k3....H
68000010: 00110000 00000000 0000464F 43090831  ..........FOC..1
68000020: 314A4D01 01000000 000000FF FFFF5804  1JM...........X.
68000030: 49230901 FFFFFFFF FFFFFFFF FFFFFFFF  I#..............
68000040: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
68000050: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
68000060: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
68000070: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
68000080: FFFF2511 FFFFFFFF 00000000 00000000  ..%.............
68000090: 2102DEFD FEEDFACE 00000000 0000000A  !.^}~mzN........
680000A0: 00000000 00000000 00000000 00000000  ................
680000B0: 00000000 00000000 00000000 00000000  ................
680000C0: 00000000 00000000 00000013 00000000  ................
680000D0: 00000000 00000000 00000000 00000000  ................
680000E0: 00000000 00000000 00000000 00000000  ................
680000F0: 00000000 00000000 00000000 00000000  ................
68000100: 00000000 00000000 00000000 00000000  ................
68000110: 00000000 00000000 00000000 00000000  ................
68000120: 00000000 00000000 00000000 00000000  ................
68000130: 00000000 00000000 00000000 00000000  ................
68000140: 00000000 00000000 50533100 726F6D6D  ........PS1.romm
68000150: 6F6E2021 203E2000 54465450 5F434845  on ! > .TFTP_CHE
68000160: 434B5355 4D003100 424F4F54 0000524F  CKSUM.1.BOOT..RO
68000170: 4D5F5045 52534953 54454E54 5F555443  M_PERSISTENT_UTC
68000180: 00313134 33353532 38383000 5245545F  .1143552880.RET_
68000190: 325F5254 5300003F 00300054 4654505F  2_RTS..?.0.TFTP_
680001A0: 46494C45 004D4152 4B4E7541 6A514179  FILE.MARKNuAjQAy
680001B0: 4A747600 52414E44 4F4D5F4E 554D0031  Jtv.RANDOM_NUM.1
680001C0: 38363937 36383434 35004253 49003000  869768445.BSI.0.
680001D0: 5245545F 325F5243 414C5453 00005341  RET_2_RCALTS..SA
680001E0: 56455F32 5F525453 0031333A 33393A34  VE_2_RTS.13:39:4
680001F0: 30205554 43205475 65204D61 72203238  0 UTC Tue Mar 28
68000200: 20323030 36000032 38383000 00545300   2006..2880..TS.
68000210: 31333A32 383A3532 20555443 20547565  13:28:52 UTC Tue
68000220: 204D6172 20323820 32303036 00000053   Mar 28 2006...S
68000230: 0000003F 00300000 00005245 545F325F  ...?.0....RET_2_
68000240: 5243414C 54530000 3F003000 00534552  RCALTS..?.0..SER
68000250: 56455200 31302E31 2E312E31 00000000  VER.10.1.1.1....
68000260: 00000000 00000000 00000000 00000000  ................
68000270: 00000000 00000000 00000000 00000000  ................
68000280: 00000000 00000000 00000000 00000000  ................
68000290: 00000000 00000000 00000000 00000000  ................
680002A0: 00000000 00000000 00000000 00000000  ................
680002B0: 00000000 00000000 00000000 00000000  ................
680002C0: 00000000 00000000 00000000 00000000  ................
680002D0: 00000000 00000000 00000000 00000000  ................
680002E0: 00000000 00000000 00000000 00000000  ................
680002F0: 00000000 00000000 00000000 00000000  ................
68000300: 00000000 00000000 00000000 00000000  ................
68000310: 00000000 00000000 00000000 00000000  ................
68000320: 00000000 00000000 00000000 00000000  ................
68000330: 00000000 00000000 00000000 00000000  ................
```

```
68000340: 00000000 00000000 00000000 00000000  ................
68000350: 00000000 00000000 00000000 00000000  ................
68000360: 00000000 00000000 00000000 00000000  ................
68000370: 00000000 00000000 00000000 00000000  ................
68000380: 00000000 00000000 00000000 00000000  ................
68000390: 00000000 00000000 00000000 00000000  ................
680003A0: 00000000 00000000 00000000 00000000  ................
680003B0: 00000000 00000000 00000000 00000000  ................
680003C0: 00000000 00000000 00000000 00000000  ................
680003D0: 00000000 00000000 00000000 00000000  ................
680003E0: 00000000 00000000 00000000 00000000  ................
680003F0: 00000000 00000000 00000000 00000000  ................
68000400: 00000000 00000000 00000000 00000000  ................
68000410: 00000000 00000000 00000000 00000000  ................
68000420: 00000000 00000000 00000000 00000000  ................
68000430: 00000000 00000000 00000000 00000000  ................
68000440: 00000000 00000000 72007265 70656174  ........r.repeat
68000450: 00680068 6973746F 7279003F 0068656C  .h.history.?.hel
68000460: 70006200 626F6F74 006C7300 64697200  p.b.boot.ls.dir.
68000470: 69007265 73657400 6B007374 61636B00  i.reset.k.stack.
68000480: 00000000 00000000 00000000 00000000  ................
68000490: 00000000 00000000 00000000 00000000  ................
680004A0: 00000000 00000000 00000000 00000000  ................
680004B0: 00000000 00000000 00000000 00000000  ................
680004C0: 00000000 00000000 00000000 00000000  ................
680004D0: 00000000 00000000 00000000 00000000  ................
680004E0: 00000000 00000000 00000000 00000000  ................
680004F0: 00000000 00000000 00000000 00000000  ................
68000500: 00000000 00000000 00000000 00000000  ................
68000510: 00000000 00000000 00000000 00000000  ................
68000520: 00000000 00000000 00000000 00000000  ................
68000530: 00000000 00000000 00000000 00000000  ................
68000540: 00000000 00000000 00000000 00000000  ................
68000550: 00000000 00000000 00000000 00000000  ................
68000560: 00000000 00000000 00000000 00000000  ................
68000570: 00000000 00000000 00000000 00000000  ................
68000580: 00000000 00000000 00000000 00000000  ................
68000590: 00000000 00000000 00000000 00000000  ................
680005A0: 00000000 00000000 00000000 00000000  ................
680005B0: 00000000 00000000 00000000 00000000  ................
680005C0: 00000000 00000000 00000000 00000000  ................
680005D0: 00000000 00000000 00000000 00000000  ................
680005E0: 00000000 00000000 00000000 00000000  ................
680005F0: 00000000 00000000 00000000 00000000  ................
68000600: 00000000 00000000 00000000 00000000  ................
68000610: 00000000 00000000 00000000 00000000  ................
68000620: 00000000 00000000 00000000 00000000  ................
68000630: 00000000 00000000 00000000 00000000  ................
68000640: 00000000 00000000 00000000 00000000  ................
68000650: 00000000 00000000 43313730 3020536F  ........C1700 So
68000660: 66747761 72652028 43313730 302D4B39  ftware (C1700-K9
68000670: 4F335359 372D4D29 2C205665 7273696F  O3SY7-M), Versio
68000680: 6E203132 2E332832 2958462C 20454152  n 12.3(2)XF, EAR
```

```
68000690: 4C592044 45504C4F 594D454E 54205245  LY DEPLOYMENT RE
680006A0: 4C454153 4520534F 46545741 52452028  LEASE SOFTWARE (
680006B0: 66633129 0A53796E 63686564 20746F20  fc1).Synched to
680006C0: 74656368 6E6F6C6F 67792076 65727369  technology versi
680006D0: 6F6E2031 322E3328 332E3929 54320A54  on 12.3(3.9)T2.T
680006E0: 41432053 7570706F 72743A20 68747470  AC Support: http
680006F0: 3A2F2F77 77772E63 6973636F 2E636F6D  ://www.cisco.com
68000700: 2F746163 0A436F6D 70696C65 64205468  /tac.Compiled Th
68000710: 75203031 2D4A616E 2D303420 30333A34  u 01-Jan-04 03:4
68000720: 34206279 2065616C 796F6E00 00000000  4 by ealyon.....
68000730: 00000000 00000000 00000000 00000000  ................
68000740: 00000000 00000000 00000000 00000000  ................
68000750: 00000000 00000000 0000818C 8D740101  .............t..
68000760: 00136073 40EB3300 01FF0448 00110000  ..`s@k3....H....
68000770: 00000000 0000464F 43090831 314A4D01  ......FOC..11JM.
68000780: 01000000 000000FF FFFF5804 49230901  ..........X.I#..
68000790: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
680007A0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
680007B0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
680007C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
680007D0: FFFFFFFF FFFFFFFF FFFFFFFF FFFF2511  ..............%.
680007E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
680007F0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  ................
68000800: FFFFFFFF FFFFF0A5 ABCD0001 AF820C03  ......p%+M../...
68000810: 00000024 82221124 00000474 00000000  ...$.".$...t....
68000820: 00000000 00000000 00000000 0A210A21  .............!.!
68000830: 204C6173 7420636F 6E666967 75726174   Last configurat
68000840: 696F6E20 6368616E 67652061 74203133  ion change at 13
68000850: 3A33353A 35372055 54432054 7565204D  :35:57 UTC Tue M
68000860: 61722032 38203230 30360A21 204E5652  ar 28 2006.! NVR
68000870: 414D2063 6F6E6669 67206C61 73742075  AM config last u
68000880: 70646174 65642061 74203133 3A33353A  pdated at 13:35:
68000890: 35392055 54432054 7565204D 61722032  59 UTC Tue Mar 2
680008A0: 38203230 30360A21 0A766572 73696F6E  8 2006.!.version
680008B0: 2031322E 330A7365 72766963 65207469   12.3.service ti
680008C0: 6D657374 616D7073 20646562 75672064  mestamps debug d
680008D0: 61746574 696D6520 6D736563 0A736572  atetime msec.ser
680008E0: 76696365 2074696D 65737461 6D707320  vice timestamps
680008F0: 6C6F6720 64617465 74696D65 206D7365  log datetime mse
68000900: 630A6E6F 20736572 76696365 20706173  c.no service pas
68000910: 73776F72 642D656E 63727970 74696F6E  sword-encryption
68000920: 0A210A68 6F73746E 616D6520 4D41524B  .!.hostname MARK
68000930: 674A6C42 444B6F77 624B0A21 0A626F6F  gJlBDKowbK.!.boo
68000940: 742D7374 6172742D 6D61726B 65720A62  t-start-marker.b
68000950: 6F6F742D 656E642D 6D61726B 65720A21  oot-end-marker.!
68000960: 0A210A6D 6D692070 6F6C6C69 6E672D69  .!.mmi polling-i
68000970: 6E746572 76616C20 36300A6E 6F206D6D  nterval 60.no mm
68000980: 69206175 746F2D63 6F6E6669 67757265  i auto-configure
68000990: 0A6E6F20 6D6D6920 7076630A 6D6D6920  .no mmi pvc.mmi
680009A0: 736E6D70 2D74696D 656F7574 20313830  snmp-timeout 180
680009B0: 0A6E6F20 61616120 6E65772D 6D6F6465  .no aaa new-mode
680009C0: 6C0A6970 20737562 6E65742D 7A65726F  l.ip subnet-zero
680009D0: 0A210A21 0A210A21 0A697020 646F6D61  .!.!.!.!.ip doma
```

```
680009E0: 696E206E 616D6520 646F6D61 696E2E63  in name domain.c
680009F0: 6F6D0A69 70206365 660A6970 20617564  om.ip cef.ip aud
68000A00: 6974206E 6F746966 79206C6F 670A6970  it notify log.ip
68000A10: 20617564 69742070 6F206D61 782D6576   audit po max-ev
68000A20: 656E7473 20313030 0A6E6F20 6674702D  ents 100.no ftp-
68000A30: 73657276 65722077 72697465 2D656E61  server write-ena
68000A40: 626C650A 210A210A 210A2120 0A6E6F20  ble.!.!.!.! .no
68000A50: 63727970 746F2069 73616B6D 7020656E  crypto isakmp en
68000A60: 61626C65 0A210A21 0A210A21 0A696E74  able.!.!.!.!.int
68000A70: 65726661 63652042 5249300A 206E6F20  erface BRI0. no
68000A80: 69702061 64647265 73730A20 73687574  ip address. shut
68000A90: 646F776E 0A210A69 6E746572 66616365  down.!.interface
68000AA0: 20466173 74457468 65726E65 74300A20   FastEthernet0.
68000AB0: 69702061 64647265 73732064 6863700A  ip address dhcp.
68000AC0: 20647570 6C657820 6175746F 0A207370   duplex auto. sp
68000AD0: 65656420 6175746F 0A210A69 6E746572  eed auto.!.inter
68000AE0: 66616365 20466173 74457468 65726E65  face FastEtherne
68000AF0: 74310A20 6E6F2069 70206164 64726573  t1. no ip addres
68000B00: 730A2073 68757464 6F776E0A 210A696E  s. shutdown.!.in
68000B10: 74657266 61636520 46617374 45746865  terface FastEthe
68000B20: 726E6574 320A206E 6F206970 20616464  rnet2. no ip add
68000B30: 72657373 0A207368 7574646F 776E0A21  ress. shutdown.!
68000B40: 0A696E74 65726661 63652046 61737445  .interface FastE
68000B50: 74686572 6E657433 0A206E6F 20697020  thernet3. no ip
68000B60: 61646472 6573730A 20736875 74646F77  address. shutdow
68000B70: 6E0A210A 696E7465 72666163 65204661  n.!.interface Fa
68000B80: 73744574 6865726E 6574340A 206E6F20  stEthernet4. no
68000B90: 69702061 64647265 73730A20 73687574  ip address. shut
68000BA0: 646F776E 0A210A69 6E746572 66616365  down.!.interface
68000BB0: 20566C61 6E310A20 6E6F2069 70206164   Vlan1. no ip ad
68000BC0: 64726573 730A210A 69702063 6C617373  dress.!.ip class
68000BD0: 6C657373 0A6E6F20 69702068 74747020  less.no ip http
68000BE0: 73657276 65720A6E 6F206970 20687474  server.no ip htt
68000BF0: 70207365 63757265 2D736572 7665720A  p secure-server.
68000C00: 210A210A 210A736E 6D702D73 65727665  !.!.!.snmp-serve
68000C10: 7220636F 6D6D756E 69747920 4D41524B  r community MARK
68000C20: 79655442 47664D57 45462052 570A736E  yeTBGfMWEF RW.sn
68000C30: 6D702D73 65727665 7220656E 61626C65  mp-server enable
68000C40: 20747261 70732074 74790A21 0A210A63   traps tty.!.!.c
68000C50: 6F6E7472 6F6C2D70 6C616E65 0A210A21  ontrol-plane.!.!
68000C60: 0A6C696E 6520636F 6E20300A 6C696E65  .line con 0.line
68000C70: 20617578 20300A6C 696E6520 76747920   aux 0.line vty
68000C80: 3020340A 210A6E6F 20736368 6564756C  0 4.!.no schedul
68000C90: 65722061 6C6C6F63 6174650A 656E640A  er allocate.end.
68000CA0: FEDC0001 000004A8 8222159B 00000467  ~\.....(.".....g
68000CB0: 0A6B6572 6265726F 73207061 7373776F  .kerberos passwo
68000CC0: 7264200A 63727970 746F2052 53412D6B  rd .crypto RSA-k
68000CD0: 65792D70 61697220 4D41524B 674A6C42  ey-pair MARKgJlB
68000CE0: 444B6F77 624B2E64 6F6D6169 6E2E636F  DKowbK.domain.co
68000CF0: 6D203020 31313433 35353239 34370A20  m 0 1143552947.
68000D00: 33303832 30313533 20303230 31303033  30820153 0201003
68000D10: 30203044 30363039 32412038 36343838  0 0D06092A 86488
68000D20: 36463720 30443031 30313031 20303530  6F7 0D010101 050
```

```
68000D30: 30303438 32203031 33443330 38322030  00482 013D3082 0
68000D40: 31333930 32303120 0A203030 30323431  1390201 . 000241
68000D50: 30302043 46393644 45363720 32394545  00 CF96DE67 29EE
68000D60: 43464533 20424233 32333037 44203736  CFE3 BB32307D 76
68000D70: 30363537 39462031 41413230 39413920  06579F 1AA209A9
68000D80: 44434146 33314538 20444137 45434235  DCAF31E8 DA7ECB5
68000D90: 31200A20 30324644 38333732 20373933  1 . 02FD8372 793
68000DA0: 30343837 41203631 41314244 34302044  0487A 61A1BD40 D
68000DB0: 41394636 35303920 41443131 46414446  A9F6509 AD11FADF
68000DC0: 20373442 44454330 46203930 34353830   74BDEC0F 904580
68000DD0: 42312044 32394533 35324420 0A203137  B1 D29E352D . 17
68000DE0: 33373339 33312030 32303330 31303020  373931 02030100
68000DF0: 30313032 34303736 20304233 31384546  01024076 0B318EF
68000E00: 32203645 36353732 46372035 38463345  2 6E6572F7 58F3E
68000E10: 46463320 46343639 35313035 20323130  FF3 F4695105 210
68000E20: 44363342 35200A20 32353345 42393444  D63B5 . 253EB94D
68000E30: 20424446 41333941 37204643 41373038   BDFA39A7 FCA708
68000E40: 41322046 33343236 31383120 34424545  A2 F3426181 4BEE
68000E50: 44334346 20424230 44374544 42203336  D3CF BB0D7EDB 36
68000E60: 44353137 32392041 44393544 33464220  D51729 AD95D3FB
68000E70: 0A203933 35453334 35422031 32424242  . 935E345B 12BBB
68000E80: 33333920 33454435 36443032 20323130  339 3ED56D02 210
68000E90: 30464443 39204333 45343643 44382036  0FDC9 C3E46CD8 6
68000EA0: 37333334 36344520 35453546 33424430  733464E 5E5F3BD0
68000EB0: 20303241 30453936 31200A20 43393034   02A0E961 . C904
68000EC0: 41464636 20393834 44324330 44203638  AFF6 984D2C0D 68
68000ED0: 39334542 46382030 45364630 32323120  93EBF8 0E6F0221
68000EE0: 30304431 36363037 20323536 45374539  00D16607 256E7E9
68000EF0: 43203442 33433142 41462031 41353335  C 4B3C1BAF 1A535
68000F00: 32424520 0A203138 36363835 37352039  2BE . 18668575 9
68000F10: 42414641 38333720 42414141 41453034  BAFA837 BAAAAE04
68000F20: 20314641 31343330 32203546 30323230   1FA14302 5F0220
68000F30: 31302034 35413138 33433120 44454436  10 45A183C1 DED6
68000F40: 32463139 20373045 38453831 45200A20  2F19 70E8E81E .
68000F50: 34334136 30373746 20324633 46444534  43A6077F 2F3FDE4
68000F60: 37203131 46453844 45342032 37324645  7 11FE8DE4 272FE
68000F70: 46354320 30464439 38443032 20323032  F5C 0FD98D02 202
68000F80: 35344245 46204546 45394442 36312034  54BEF EFE9DB61 4
68000F90: 39343133 45383920 0A203134 42453443  9413E89 . 14BE4C
68000FA0: 39392039 31344430 39364520 36363339  99 914D096E 6639
68000FB0: 38363533 20434543 41433143 44204638  8653 CECAC1CD F8
68000FC0: 33383437 46302032 46303232 30304620  3847F0 2F02200F
68000FD0: 30393235 32323235 20384334 44303431  09252225 8C4D041
68000FE0: 41200A20 37373532 41353930 20434145  A . 7752A590 CAE
68000FF0: 34443236 43203630 42384438 36342037  4D26C 60B8D864 7
68001000: 41413131 30343120 33413237 43304241  AA11041 3A27C0BA
68001010: 20464646 3233330A 20717569 740A2033   FFF233. quit. 3
68001020: 30354333 30304420 30363039 32413836  05C300D 06092A86
68001030: 20343838 36463730 44203031 30313031   4886F70D 010101
68001040: 30352030 30303334 42303020 33303438  05 00034B00 3048
68001050: 30323431 20303043 46393644 45203637  0241 00CF96DE 67
68001060: 32394545 4346200A 20453342 42333233  29EECF . E3BB323
68001070: 30203744 37363036 35372039 46314141  0 7D760657 9F1AA
```

```
68001080: 32303920 41394443 41463331 20453844  209 A9DCAF31 E8D
68001090: 41374543 42203531 30324644 38332037  A7ECB 5102FD83 7
680010A0: 32373933 30343820 37413631 41314244  2793048 7A61A1BD
680010B0: 200A2034 30444139 46363520 30394144   . 40DA9F65 09AD
680010C0: 31314641 20444637 34424445 43203046  11FA DF74BDEC 0F
680010D0: 39303435 38302042 31443239 45333520  904580 B1D29E35
680010E0: 32443137 33373339 20333130 32303330  2D173739 3102030
680010F0: 31203030 30310A20 71756974 0A736E6D  1 0001. quit.snm
68001100: 702D7365 72766572 20686320 706F6C6C  p-server hc poll
68001110: 20300A65 6E640A00 00000000 00000000   0.end.........
68001120: 00000000 00000000 00000000 00000000  ................
68001130: 00000000 00000000 00000000 00000000  ................
68001140: 00000000 00000000 00000000 00000000  ................
68001150: 00000000 00000000 00000000 00000000  ................
68001160: 00000000 00000000 00000000 00000000  ................
68001170: 00000000 00000000 00000000 00000000  ................
68001180: 00000000 00000000 00000000 00000000  ................
68001190: 00000000 00000000 00000000 00000000  ................
680011A0: 00000000 00000000 00000000 00000000  ................
680011B0: 00000000 00000000 00000000 00000000  ................
680011C0: 00000000 00000000 00000000 00000000  ................
680011D0: 00000000 00000000 00000000 00000000  ................
680011E0: 00000000 00000000 00000000 00000000  ................
680011F0: 00000000 00000000 00000000 00000000  ................
68001200: 00000000 00000000 00000000 00000000  ................
<SNIP>
```

Appendix D. ProCurve Switch 2626 investigation

Output listing 6-8: ProCurve Switch 2626 Bench jumper mode commands

```
tty=noneProCurve Switch 2626=>
 logout                 Terminate this console/telnet session.
 DEBUGIO                Redirects output from all printf()'s to the screen
 FORCE_REDRAW           Forces the redraw of field labels in config screens
 LABprototype           Change LAB Prototype status
 UPTIMESHOW             Shows time the switch has been up
 DATAProtshow           Show information on all dataProt semaphores
 MSGPoolshow            Dumps the MSG pool
 PKTPoolshow            Dumps the PKT pool
 BUFSHOW                Dumps a MSG or PKT buffer
 PKTpoolStatsShow       Show the PKT pool allocation statistics
 MSGpoolStatsShow       Show the MSG pool allocation statistics
 PKTPoolDatashow        Dumps the PKT pool data
 PKTpoolcrashifless     Crash if pkt pool goes below this
 CRASHData              Show crash information
 CRASHLogfileshow       Show all recorded crash records
 CRASHLOGTest           Crash Log Test: crashLogTest -[b][i][I][s][f][a][o][u]b
                        = Bus/Address Error, i = Infinite loop with tasks
                        locked, I = HW watchdog resets = task Infinite loop, f
                        = FATAL, a = ASSERTo = operation fault (illegal inst.)
                        u = unaligned instr
 CRASHLOGClear          Clear Crash Log: crashLogClear
 EVENTLogfileshow       Show contents of the event log file

 LLshow                 Detailed directory listing: llShow <filesystem (eg.
 LSshow                 Directory listing: lsShow <filesystem (eg.
 FS                     File system commands
 SInfo                  Information on registered Servers
 CSConninfo             Information on registered Client-Server connections
 MEM_Rpt                Show memory usage info: mem_rpt [-d]
 MEM_Chk                Check memory allocation data structures
 MEM_Chk_Add            Turns on memory checking at task switch: mem_chk_add
                        [-f]
 MEM_Chk_Rem            Turns off memory checking
 I                      Task Info
 CHECKSTACK             checkStack()
 SEMSHOW                semShow(semid) - semaphore show
 TASKSUSPEND            taskSuspend (taskId) - suspend a task
 TASKRESUME             taskResume (taskId) - resume a suspended task
 SEMAllshow             Show information on all switch semaphores
 EXCeption-ignore       Manage the exception list
 DMACOUNTERSshow        Show DMA Driver counters
 DMACLEARcounters       ClearDMA Driver counters
 VERsion                Display firmware version stamp
 ROMVERsion             Display ROM Version

 SETTERM                set the terminal to vt100 or ASCII
 BOOTCOUNTER            Number of times this switch has been powered up.
 UPLINK                 Select and configure the uplinks
```

```
HReset                Hard Reset of the Switch
STREBOOT              Reboot to Benchmode
GETOS                 GETOS <ipaddr> <remote-file>
UPDMAC                Update the MAC address (AABBCC-DDEEFF) :
UPDMACNUM             Update the number of MAC addresses
Read                  Read memory: r [MOPT] <ADDR>
WR                    Write memory: w [MOPT] <ADDR> <VALUE>
FILL                  Fill memory: fill [MOPT] <ADDR> <ADDR> <VALUE>
UPDSN                 Update the Serial Number
UPDMFG                Update the specified manufacturing info
CLRMFG                Clear the specified manufacturing info
LED                   Turn all possible LEDs [on|blink|off]
SMode                 Set Memory Mode: sm [-l<READ_LENGTH> -b -h -w -a<bhw>
                      -d<bhw> -n -i -c -s] Set default memory operation modes
                      (MOPT).
LIST                  List available tests matching <TSPEC>: list <TSPEC>
TEST                  Execute tests matching <TSPEC>, <NUM> times: test
                      <TSPEC> <NUM>

ST_mode               Set the Selftest mode: st_mode <mfg|norm|spec>
IGNore                Set Selftest to ignore failures
STOP                  Set Selftest to stop on failure
ST_DIsplay            Set Selftest reporting verbosity: st_display
                      <none|low|med|hi>
VIEW                  View the Selftest test ring: view [-s -d -i] <count>
ARLTEST               Runs Arl Test on the Asic(s): arltest
ST_JUMP_pc            Resume product code initialization from benchmode
LINKVALID             Detect link: linkvalid <port> <timeout>
TXRX                  Port pkt test: txrx <TX port> <RX port> <speed>
                      <duplex> <mode> count>
macloopback           Usage: macloopback <TX port> <# of pkt>
ST_LIST               Display loopback tests for port: st_list <port>
X_TYPE                Display Transceiver Type for <slot port>
BIST                  Run the BCM bist test on the chip
MEMTEST               Performs the Switching memory test: memtest <timeout>
GBICINFO              Retreives miniGBIC info : gbicinfo <port>
GBICHOTSWAP           Performs a hot-swap test on specified GBIC module:
                      gbicHotswap <port> <timeout>
GBICTXDISABLETEST     Performs TX Disable test on specified port:
                      gbicTxDisableTest <port>

GBPTest               Test memory interface to ASIC
dType                 debug type set/clear
PDSHOW                Show various PowerDsine information
PDPOWER               Set PowerDsine Power Supply Value
PDCAP                 Set PowerDsine Capacitor Detection
PDDISCON              Set PD33000 AC/DC Detection Mode
POE_PORT              Set user configurable port parameters.
POE_STATUS_PORT       Display port statistics and measurements.
POE_DEBUG             Change the POE Debug level.
POE_READ_EPS          Read from the specified EPS register.
POE_WRITE_EPS         Write to the specified EPS register.
POE_SLOT_UP           Enable a slot for POE functionality.
```

```
POE_START              Enable POE Mgr polling.
POE_STOP               Disable POE Mgr polling.
POE_EPS_TIMEOUT        Enable/Disable EPS timeout.
POE_EPS_COMM_INIT      Send the reset/init sequence to the I2C micro.
POE_PD_CHECK_ALIVE     Test to see if the Tweety PD is alive.
POE_PD_INIT            Hard init the PD unit.
POE_PD_FACTORY         Restore the PD unit to factory defaults.
POE_EPS_SIGNAL         Simulate and EPS Int.
POE_EPS_DEBUG          Enable/Disable EPS debug timeout.

CHASSISshow            Show various chassis information
S_CFG                  Display Cage: s_cfg
WATCHDOG               set watchdog parameters
MEMWATCH               set the wp
CHIPVER                Prints the Chip Versions
RPSset                 Set a wanted RPS state
I2CREAD                Read from the specified PPC I2C device and register
I2CWRITE               Write to the specified PPC I2C device and register
UPGRADE
DOWNGRADE
CONFIGTest             Verify CLI generation/Xlate function
P_BCNTRCLR             Clear all counters for unit/port
P_BCNTRCLRALL          Clear all counters in context.
DROPCOUNT              Online diag to get stacklink drop counts.
BCM                    Broadcom Debug: bcm <string for broadcom debugger>
UNIT_INIT              Recommended Usage: [slot <number(s)>] unit_init
UNIT_UPDATE            Recommended Usage: [slot <number(s)>] unit_update
UNIT_LINK              Checks link state of slot's ports

boot                   Reboot the device.
clear                  Clear table/statistics or authorized client public

                       keys.
configure              Enter the Configuration context.
copy                   Copy datafiles to/from the switch.
debug                  Enable/disable debug logging.
end                    Return to the Manager Exec context.
erase                  Erase the configuration file stored in flash or the
                       primary/secondary flash image.
getMIB                 Retrieve and display the value of the MIB objects
                       specified.
kill                   Kill other active console, telnet, or ssh sessions.
log                    Display log events.
page                   Toggle paging mode.
print                  Execute a command and redirect its output to the device
                       channel for current session.
redo                   Re-execute a command from history.
reload                 Warm reboot of the switch.
repeat                 Repeat execution of a previous command.
setMIB                 Set the value of a MIB object.
setup                  Enter the 'Switch Setup' screen for basic switch
                       configuration.
telnet                 Initiate an outbound telnet session to another network
```

```
                      device.
 terminal             Set the dimensions of the terminal window.
 update               Enter Monitor ROM Console.
 walkMIB              Walk through all instances of the object specified
                      displaying the MIB object names, instances and values.
 write                View or save the running configuration of the switch.

 enable               Enter the Manager Exec context.
 exit                 Return to the previous context or terminate current
                      console/telnet session if you are in the Operator
                      context level.
 link-test            Test the connection to a MAC address on the LAN.
 logout               Terminate this console/telnet session.
 menu                 Change console user interface to menu system.
 ping                 Send IP Ping requests to a device on the network.
 show                 Display switch operation information.
 traceroute           Send traceroute to a device on the network.


tty=noneProCurve Switch 2626=>
```

Output listing 6-9: ProCurve Switch 2626 File system investigation commands

```
ProCurve Switch 2626$ fs pnbfswalk
0x14f5178   cfg
0x14f4b38      running-config
0x14f4a70      startup-config
0x14f50b0   flash
0x14f4fe8   log
0x14f48e0      crash-data
0x14f4818      crash-log
0x14f49a8      event-log
0x14f4f20   os
0x14f4750      primary
0x14f4688      secondary
0x14f4e58   ramfs
0x14f4d90   ssh
0x14f4cc8      mgr_keys
0x14f45c0         authorized_keys
0x14f4c00      oper_keys
0x14f44f8         authorized_keys

ProCurve Switch 2626$ fs ls flash

Name              Size    Date
----------------  ------  -----------------
      .bootblock    1248  02/06/26 06:28:15
     mgrinfo.txt      96  01/01/90 00:00:21
      config.txt    6555  01/01/90 00:00:05
          iflags      50  01/01/90 00:00:14
          rbtcnt       4  01/01/90 00:00:05

fs nvfswalk

ProCurve Switch 2626$ fs nvfswalk
   A        addr          filename      size      date  flgs
  --  ----------  ----------------  --------  --------  ----
  **  0x7cf20000        .bootblock  000004e0  ffffffff  ffff
      0x7cf20500            rbtcnt  00000004  00000002  ffff
      0x7cf20530            rbtcnt  00000004  00000002  ffff
      0x7cf20560            iflags  00000032  00000006  ffff
      0x7cf205c0            rbtcnt  00000004  00000002  ffff
      0x7cf205f0            iflags  00000032  00000006  ffff
      0x7cf20650            rbtcnt  00000004  0000000c  ffff
      0x7cf20680            iflags  00000032  0000001f  ffff
      0x7cf206e0            rbtcnt  00000004  00000002  ffff
      0x7cf20710            iflags  00000032  00000006  ffff
      0x7cf20770            rbtcnt  00000004  00000002  ffff
      0x7cf207a0            iflags  00000032  00000006  ffff
      0x7cf20800            rbtcnt  00000004  00000002  ffff
      0x7cf20830            iflags  00000032  00000006  ffff
      0x7cf20890            rbtcnt  00000004  00000002  ffff
      0x7cf208c0            iflags  00000032  00000006  ffff
      0x7cf20920            rbtcnt  00000004  00000002  ffff
      0x7cf20950            iflags  00000032  00000006  ffff
```

```
        0x7cf209b0           rbtcnt  00000004  00000002  ffff
        0x7cf209e0           iflags  00000032  00000006  ffff
        0x7cf20a40           rbtcnt  00000004  00000002  ffff
        0x7cf20a70           iflags  00000032  00000006  ffff
        0x7cf20ad0           rbtcnt  00000004  00000002  ffff
        0x7cf20b00           iflags  00000032  00000006  ffff
        0x7cf20b60           rbtcnt  00000004  00000002  ffff
        0x7cf20b90           iflags  00000032  00000006  ffff
        0x7cf20bf0           rbtcnt  00000004  10000000  ffff
        0x7cf20c20           iflags  00000032  10000004  ffff
        0x7cf20c80           rbtcnt  00000004  00000002  ffff
        0x7cf20cb0           iflags  00000032  00000006  ffff
        0x7cf20d10           rbtcnt  00000004  00000002  ffff
        0x7cf20d40           iflags  00000032  00000006  ffff
        0x7cf20da0           rbtcnt  00000004  00000002  ffff
        0x7cf20dd0           iflags  00000032  00000006  ffff
        0x7cf20e30           rbtcnt  00000004  00000002  ffff
        0x7cf20e60           iflags  00000032  00000006  ffff
        0x7cf20ec0           rbtcnt  00000004  00000002  ffff
        0x7cf20ef0           iflags  00000032  00000006  ffff
        0x7cf20f50           rbtcnt  00000004  00000002  ffff
        0x7cf20f80           iflags  00000032  00000006  ffff
        0x7cf20fe0           rbtcnt  00000004  00000002  ffff
        0x7cf21010           iflags  00000032  00000006  ffff
        0x7cf21070           rbtcnt  00000004  00000002  ffff
        0x7cf210a0           iflags  00000032  00000006  ffff
        0x7cf21100           rbtcnt  00000004  00000002  ffff
        0x7cf21130           iflags  00000032  00000006  ffff
        0x7cf21190           rbtcnt  00000004  00000002  ffff
        0x7cf211c0           iflags  00000032  00000006  ffff
        0x7cf21220           rbtcnt  00000004  00000003  ffff
        0x7cf21250           iflags  00000032  0000000f  ffff
        0x7cf212b0           rbtcnt  00000004  00000002  ffff
        0x7cf212e0           rbtcnt  00000004  00000002  ffff
        0x7cf21310           iflags  00000032  00000006  ffff
        0x7cf21370            delta  00002800  00000022  ffff
        0x7cf23b90           rbtcnt  00000004  00000002  ffff
        0x7cf23bc0           iflags  00000032  00000007  ffff
        0x7cf23c20           rbtcnt  00000004  00000003  ffff
        0x7cf23c50           iflags  00000032  00000010  ffff
        0x7cf23cb0           rbtcnt  00000004  00000003  ffff
        0x7cf23ce0           rbtcnt  00000004  00000003  ffff
        0x7cf23d10           rbtcnt  00000004  00000003  ffff
        0x7cf23d40           iflags  00000032  00000010  ffff
        0x7cf23da0           rbtcnt  00000004  00000003  ffff
        0x7cf23dd0           iflags  00000032  00000010  ffff
        0x7cf23e30            delta  00002800  00000090  ffff
        0x7cf26650           rbtcnt  00000004  00000003  ffff
        0x7cf26680           rbtcnt  00000004  00000003  ffff
        0x7cf266b0           rbtcnt  00000004  00000003  ffff
        0x7cf266e0           rbtcnt  00000004  00000003  ffff
        0x7cf26710           rbtcnt  00000004  00000003  ffff
        0x7cf26740           iflags  00000032  0000000a  ffff
```

```
       0x7cf267a0            rbtcnt  00000004  00000003  ffff
       0x7cf267d0            rbtcnt  00000004  00000003  ffff
       0x7cf26800            rbtcnt  00000004  00000003  ffff
       0x7cf26830            iflags  00000032  0000000a  ffff
       0x7cf26890             delta  00002800  0000002b  ffff
       0x7cf290b0            rbtcnt  00000004  00000003  ffff
       0x7cf290e0            iflags  00000032  0000000a  ffff
       0x7cf29140            rbtcnt  00000004  00000003  ffff
       0x7cf29170            iflags  00000032  0000000a  ffff
       0x7cf291d0            rbtcnt  00000004  00000003  ffff
       0x7cf29200            rbtcnt  00000004  00000003  ffff
       0x7cf29230            rbtcnt  00000004  00000003  ffff
       0x7cf29260            iflags  00000032  0000000a  ffff
       0x7cf292c0            rbtcnt  00000004  00000003  ffff
       0x7cf292f0            iflags  00000032  0000000a  ffff
       0x7cf29350             delta  00002800  0000004e  ffff
       0x7cf2bb70            rbtcnt  00000004  00000003  ffff
       0x7cf2bba0            iflags  00000032  0000000a  ffff
       0x7cf2bc00            rbtcnt  00000004  00000003  ffff
       0x7cf2bc30            rbtcnt  00000004  00000003  ffff
       0x7cf2bc60            rbtcnt  00000004  00000003  ffff
       0x7cf2bc90            iflags  00000032  0000000a  ffff
       0x7cf2bcf0             delta  00002800  00000023  ffff
       0x7cf2e510            rbtcnt  00000004  00000003  ffff
       0x7cf2e540            iflags  00000032  0000000a  ffff
       0x7cf2e5a0            rbtcnt  00000004  00000003  ffff
       0x7cf2e5d0            rbtcnt  00000004  00000003  ffff
       0x7cf2e600            rbtcnt  00000004  00000003  ffff
       0x7cf2e630            iflags  00000032  0000000a  ffff
       0x7cf2e690             delta  00002800  00000036  ffff
       0x7cf30eb0            rbtcnt  00000004  00000003  ffff
       0x7cf30ee0            iflags  00000032  0000000a  ffff
       0x7cf30f40            rbtcnt  00000004  00000003  ffff
       0x7cf30f70            rbtcnt  00000004  00000003  ffff
       0x7cf30fa0            rbtcnt  00000004  00000003  ffff
       0x7cf30fd0            iflags  00000032  0000000a  ffff
       0x7cf31030            rbtcnt  00000004  00000003  ffff
       0x7cf31060            iflags  00000032  0000000a  ffff
       0x7cf310c0            rbtcnt  00000004  00000003  ffff
       0x7cf310f0            iflags  00000032  0000000a  ffff
       0x7cf31150             delta  00002800  00000079  ffff
       0x7cf33970            rbtcnt  00000004  00000003  ffff
       0x7cf339a0            iflags  00000032  0000000a  ffff
       0x7cf33a00            rbtcnt  00000004  00000003  ffff
       0x7cf33a30            iflags  00000032  0000000a  ffff
       0x7cf33a90            rbtcnt  00000004  00000003  ffff
       0x7cf33ac0            iflags  00000032  0000000a  ffff
       0x7cf33b20             delta  00002800  00000156  ffff
       0x7cf36340       mgrinfo.txt  00000050  000001d6  ffff
       0x7cf363b0            rbtcnt  00000004  00000003  ffff
       0x7cf363e0            iflags  00000032  0000000a  ffff
       0x7cf36440            rbtcnt  00000004  00000003  ffff
       0x7cf36470            iflags  00000032  0000000a  ffff
```

```
      0x7cf364d0          rbtcnt  00000004  00000003  ffff
      0x7cf36500          iflags  00000032  0000000a  ffff
      0x7cf36560     mgrinfo.txt  00000050  00003267  ffff
      0x7cf365d0          rbtcnt  00000004  00000003  ffff
      0x7cf36600          iflags  00000032  0000000a  ffff
      0x7cf36660          rbtcnt  00000004  00000003  ffff
      0x7cf36690          iflags  00000032  0000000a  ffff
      0x7cf366f0          rbtcnt  00000004  00000003  ffff
      0x7cf36720          iflags  00000032  0000000a  ffff
      0x7cf36780          rbtcnt  00000004  00000003  ffff
      0x7cf367b0          iflags  00000032  0000000a  ffff
      0x7cf36810          rbtcnt  00000004  00000003  ffff
      0x7cf36840          iflags  00000032  0000000a  ffff
      0x7cf368a0          rbtcnt  00000004  00000006  ffff
      0x7cf368d0          iflags  00000032  0000000d  ffff
      0x7cf36930           delta  00002800  00000025  ffff
      0x7cf39150     mgrinfo.txt  00000050  0001828c  ffff
      0x7cf391c0          rbtcnt  00000004  00000005  ffff
      0x7cf391f0      config.txt  00001d39  00000006  ffff
      0x7cf3af50           delta  00002800  00000082  ffff
      0x7cf3d770          rbtcnt  00000004  00000005  ffff
      0x7cf3d7a0          rbtcnt  00000004  00000005  ffff
      0x7cf3d7d0          rbtcnt  00000004  00000005  ffff
      0x7cf3d800     mgrinfo.txt  00000060  003395f2  ffff
      0x7cf3d880          rbtcnt  00000004  00000005  ffff
      0x7cf3d8b0     mgrinfo.txt  00000060  0088c9ff  ffff
      0x7cf3d930     mgrinfo.txt  00000060  0088ca00  ffff
      0x7cf3d9b0     mgrinfo.txt  00000060  0088ca00  ffff
      0x7cf3da30     mgrinfo.txt  00000060  0088ca01  ffff
      0x7cf3dab0     mgrinfo.txt  00000060  0088ca17  ffff
      0x7cf3db30     mgrinfo.txt  00000060  0088ca18  ffff
      0x7cf3dbb0     mgrinfo.txt  00000060  0088ca1e  ffff
      0x7cf3dc30     mgrinfo.txt  00000060  0088ca1e  ffff
      0x7cf3dcb0     mgrinfo.txt  00000060  0088ca80  ffff
      0x7cf3dd30          rbtcnt  00000004  00000005  ffff
      0x7cf3dd60          rbtcnt  00000004  01419e18  ffff
      0x7cf3dd90          rbtcnt  00000004  00000005  ffff
      0x7cf3ddc0          rbtcnt  00000004  01f87e7b  ffff
      0x7cf3ddf0          rbtcnt  00000004  00000005  ffff
      0x7cf3de20          rbtcnt  00000004  00000005  ffff
      0x7cf3de50          rbtcnt  00000004  00000005  ffff
      0x7cf3de80          rbtcnt  00000004  00000005  ffff
      0x7cf3deb0          rbtcnt  00000004  00000005  ffff
      0x7cf3dee0          rbtcnt  00000004  00000005  ffff
      0x7cf3df10          rbtcnt  00000004  00000005  ffff
      0x7cf3df40          rbtcnt  00000004  00000005  ffff
      0x7cf3df70          rbtcnt  00000004  00000005  ffff
      0x7cf3dfa0          rbtcnt  00000004  00000005  ffff
      0x7cf3dfd0          rbtcnt  00000004  00000005  ffff
      0x7cf3e000          rbtcnt  00000004  00000005  ffff
  **  0x7cf3e030     mgrinfo.txt  00000060  00000015  ffff
      0x7cf3e0b0          rbtcnt  00000004  00000005  ffff
      0x7cf3e0e0           delta  00002800  00000005  ffff
  **  0x7cf40900        config.txt  0000199b  00000005  ffff
  **  0x7cf422c0            iflags  00000032  0000000e  ffff
  **  0x7cf42320            rbtcnt  00000004  00000005  ffff

ProCurve Switch 2626$ fs ramfswalk
ramFAT | open files   : 0
       | total files  : 7
       | total blocks : 99
       | total memory : 319472

        filename  file  open  ct  type     size      date   blk  C    linklist
----------------  ----  ----  --  ----  -------  --------  ----  -  ----------
               .  4000  0000   0  VOLM        0  00000003     0  -    0x8aadc4
       crash.log  0001  0000   0  CLog     5680  00000003     1  +   0x1ff2258
       event.log  0001  0000   0  Evnt   161188  00000003     1  +   0x1fcaca8
       crash.dat  0004  0000   0  CDat       25  00000003     1  +   0x1ff3894
      config.upd  0004  0001   0  Allc     6555  00000005     7  -   0x18ebe70
      config.cfg  0004  0001   0  Allc     6555  00000005     7  -   0x18ea1a0
          moduli  0004  0002   0  Allc    83922  0000000e    82  -   0x18485f8
```

Appendix E. ProCurve Switch 2824 investigation

Output listing 6-10: Procurve 2824 marker search after factory reset, in Bench mode

```
ty=noneProCurve Switch 2824=> sm -b -i
  access:b, display:b, read_length:256, inc addr after read

tty=noneProCurve Switch 2824=> fs nvfswalk
   A        addr          filename      size      date  flgs
  --  ----------  ----------------  --------  --------  ----
  **  0xfff20000        .bootblock  000004e0  ffffffff  ffff
      0xfff20500            rbtcnt  00000004  0000000a  ffff
      0xfff20530            iflags  00000032  0000000a  ffff
      0xfff20590            rbtcnt  00000004  0000000a  ffff
      0xfff205c0            rbtcnt  00000004  0000000a  ffff
      0xfff205f0            rbtcnt  00000004  0000000a  ffff
      0xfff20620            iflags  00000032  0000000a  ffff
      0xfff20680            rbtcnt  00000004  0000000a  ffff
      0xfff206b0            rbtcnt  00000004  0000000a  ffff
      0xfff206e0             delta  00002800  0000005a  ffff
      0xfff22f00       mgrinfo.txt  00000060  0000005a  ffff
      0xfff22f80        config.txt  000019c1  0000006f  ffff
      0xfff24970             delta  00002800  0000006f  ffff
      0xfff27190       mgrinfo.txt  00000060  000000d3  ffff
      0xfff27210       mgrinfo.txt  00000060  000000e3  ffff
      0xfff27290            rbtcnt  00000004  00000119  ffff
      0xfff272c0       mgrinfo.txt  00000060  00000217  ffff
      0xfff27340            rbtcnt  00000004  0000000a  ffff
      0xfff27370            rbtcnt  00000004  0000000a  ffff
      0xfff273a0            rbtcnt  00000004  0000000a  ffff
      0xfff273d0        config.txt  00002623  21479a96  ffff

      0xfff29a20             delta  00002800  21479aa1  ffff
      0xfff2c240       mgrinfo.txt  00000060  231d2954  ffff
      0xfff2c2c0            rbtcnt  00000004  0000000a  ffff
      0xfff2c2f0            rbtcnt  00000004  00000009  ffff
      0xfff2c320            rbtcnt  00000004  0000000c  ffff
      0xfff2c350            rbtcnt  00000004  00000009  ffff
      0xfff2c380            rbtcnt  00000004  00000009  ffff
      0xfff2c3b0       mgrinfo.txt  00000060  00000009  ffff
      0xfff2c430       mgrinfo.txt  00000060  00000009  ffff
      0xfff2c4b0            rbtcnt  00000004  00000009  ffff
      0xfff2c4e0            iflags  00000032  00000009  ffff
      0xfff2c540            rbtcnt  00000004  00000009  ffff
      0xfff2c570            rbtcnt  00000004  00008ca2  ffff
      0xfff2c5a0             delta  00002800  00008ca2  ffff
      0xfff2edc0        config.txt  00001950  00008ca2  ffff
      0xfff30730             delta  00002800  00008ca8  ffff
      0xfff32f50            rbtcnt  00000004  0000000c  ffff
      0xfff32f80            rbtcnt  00000004  00000009  ffff
      0xfff32fb0            rbtcnt  00000004  00000009  ffff
      0xfff32fe0            rbtcnt  00000004  00000009  ffff
      0xfff33010         host_ssh1  000001d4  00000066  ffff
      0xfff33210           rbtcnt  00000004  00000009  ffff
      0xfff33240            delta  00002800  00000009  ffff
      0xfff35a60       config.txt  00001950  00000009  ffff
      0xfff373d0           iflags  00000032  0000001b  ffff
      0xfff37430            delta  00002800  0000001d  ffff
      0xfff39c50      mgrinfo.txt  00000060  00000046  ffff
  **  0xfff39cd0        host_ssh1  000001d4  0000004d  ffff
      0xfff39ed0           rbtcnt  00000004  00000009  ffff
      0xfff39f00      mgrinfo.txt  00000060  00000009  ffff
  **  0xfff39f80      mgrinfo.txt  00000060  00000009  ffff
      0xfff3a000            delta  00002800  00000009  ffff
      0xfff3c820       config.txt  00001950  00000009  ffff
      0xfff3e190            delta  00002800  0000001d  ffff
      0xfff409b0           rbtcnt  00000004  00000009  ffff
      0xfff409e0           rbtcnt  00000004  00000009  ffff
      0xfff40a10           rbtcnt  00000004  00000009  ffff
      0xfff40a40            delta  00002800  00000009  ffff
  **  0xfff43260       config.txt  00001950  00000009  ffff
  **  0xfff44bd0           iflags  00000032  0000001a  ffff
  **  0xfff44c30            delta  00002800  0000001d  ffff
  **  0xfff47450           rbtcnt  00000004  00000009  ffff

tty=noneProCurve Switch 2824=> read 0xfff37430
fff37430  64 65 6c 74 61 00 ff ff ff ff ff ff ff ff ff ff   delta...........
fff37440  00 00 28 00 00 00 00 1d ff f3 9c 50 00 00 ff ff   ..(........P....
fff37450  00 01 e1 00 01 0a 00 01 e1 00 0d 53 4e 4d 50 4e   ...........SNMPN
fff37460  4f 54 49 46 59 20 28 0a 00 01 e1 00 0d 52 4f 57   OTIFY (......ROW
fff37470  5f 53 54 41 54 55 53 3d 31 0a 00 01 e1 00 12 4e   _STATUS=1......N
fff37480  41 4d 45 3d 7e 73 74 61 63 6b 74 72 61 70 73 7e   AME=~stacktraps~
fff37490  0a 00 01 e1 00 0e 54 41 47 3d 73 74 61 63 6b 74   ......TAG=stackt
fff374a0  72 61 70 0a 00 01 e1 00 0e 53 54 4f 52 41 47 45   rap......STORAGE
fff374b0  54 59 50 45 3d 35 0a 00 01 e3 00 12 53 4e 4d 50   TYPE=5......SNMP
fff374c0  56 33 43 4f 4d 4d 55 4e 49 54 59 20 28 0a 00 01   V3COMMUNITY (...
fff374d0  e3 00 0d 52 4f 57 5f 53 54 41 54 55 53 3d 31 0a   ...ROW_STATUS=1.
fff374e0  00 01 e3 00 09 4e 41 4d 45 3d 7e 31 7e 0a 00 01   .....NAME=~1~...
fff374f0  e3 00 13 43 4f 4d 4d 5f 4e 41 4d 45 3d 7e 70 75   ...COMM_NAME=~pu
fff37500  62 6c 69 63 7e 0a 00 01 e3 00 25 53 45 43 5f 4e   blic~.....%SEC_N
fff37510  41 4d 45 3d 7e 43 6f 6d 6d 75 6e 69 74 79 4d 61   AME=~CommunityMa
fff37520  6e 61 67 65 72 52 65 61 64 57 72 69 74 65 7e 0a   nagerReadWrite~.

tty=noneProCurve Switch 2824=> read
fff37530  00 01 e3 00 0e 53 54 4f 52 41 47 45 54 59 50 45   .....STORAGETYPE
fff37540  3d 32 0a 00 01 e3 00 02 29 0a 00 01 e3 00 02 29   =2......)......)
fff37550  0a 00 01 e3 00 01 0a 00 03 2c 00 0d 44 48 43 50   .........,..DHCP
fff37560  53 4e 4f 4f 50 72 20 28 0a 00 03 2c 00 0d 52 4f   SNOOPr (...,..RO
fff37570  57 5f 53 54 41 54 55 53 3d 33 0a 00 03 2c 00 02   W_STATUS=3...,..
fff37580  29 0a 00 03 2c 00 01 0a fe 02 00 04 00 16 4e 41   )...,.........NA
fff37590  4d 45 3d 7e 4d 41 52 4b 74 42 48 71 77 77 45 71   ME=~MARKtBHqwwEq
fff375a0  70 43 7e 0a 00 01 f0 00 01 0a 00 01 f0 00 12 53   pC~............S
fff375b0  4e 4d 50 56 33 43 4f 4d 4d 55 4e 49 54 59 20 28   NMPV3COMMUNITY (
fff375c0  0a 00 01 f0 00 0d 52 4f 57 5f 53 54 41 54 55 53   ......ROW_STATUS
fff375d0  3d 31 0a 00 01 f0 00 09 4e 41 4d 45 3d 7e 32 7e   =1......NAME=~2~
```

```
fff375e0  0a 00 01 f0 00 1b 43 4f 4d 4d 5f 4e 41 4d 45 3d   ......COMM_NAME=
fff375f0  7e 4d 41 52 4b 77 42 45 4b 6f 77 62 4b 52 42 7e   ~MARKwBEKowbKRB~
fff37600  0a 00 01 f0 00 25 53 45 43 5f 4e 41 4d 45 3d 7e   .....%SEC_NAME=~
fff37610  43 6f 6d 6d 75 6e 69 74 79 4f 70 65 72 61 74 6f   CommunityOperato
fff37620  72 52 65 61 64 4f 6e 6c 79 7e 0a 00 01 f0 00 0e   rReadOnly~......

tty=noneProCurve Switch 2824=> read
fff37630  53 54 4f 52 41 47 45 54 59 50 45 3d 32 0a 00 01   STORAGETYPE=2...
fff37640  f0 00 02 29 0a 00 01 f8 00 08 53 4e 4d 50 53 20   ...)......SNMPS
fff37650  28 0a 00 01 f8 00 0d 52 4f 57 5f 53 54 41 54 55   (......ROW_STATU
fff37660  53 3d 31 0a 00 01 f8 00 09 43 4f 4d 5f 49 44 3d   S=1......COM_ID=
fff37670  32 0a 00 01 f8 00 16 4e 41 4d 45 3d 7e 4d 41 52   2......NAME=~MAR
fff37680  4b 77 42 45 4b 6f 77 62 4b 52 42 7e 0a 00 01 f8   KwBEKowbKRB~....
fff37690  00 07 56 49 45 57 3d 33 0a 00 01 f8 00 02 29 0a   ..VIEW=3......).
fff376a0  00 01 f8 00 01 0a fe ff ff ff ff ff ff ff ff ff   ................
fff376b0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
fff376c0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
fff376d0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
fff376e0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
fff376f0  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
fff37700  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
fff37710  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
fff37720  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................

tty=noneProCurve Switch 2824=> read 0xfff39c50
fff39c50  6d 67 72 69 6e 66 6f 2e 74 78 74 00 ff ff ff ff   mgrinfo.txt.....
fff39c60  00 00 00 60 00 00 00 46 ff f3 9c d0 00 00 ff ff   ...`...F........
fff39c70  00 00 00 01 46 4c 47 00 4d 41 52 4b 63 47 73 58   ....FLG.MARKcGsX
fff39c80  41 47 56 63 49 47 00 00 00 00 00 00 00 00 00 00   AGVcIG..........
fff39c90  00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff fb   ................
fff39ca0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
fff39cb0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
fff39cc0  00 00 00 00 00 00 00 00 00 00 00 02 06 00 00 00   ................
fff39cd0  68 6f 73 74 5f 73 73 68 31 00 ff ff ff ff ff ff   host_ssh1.......
fff39ce0  00 00 01 d4 00 00 00 4d ff f3 9e d0 00 ff ff ff   .......M........
fff39cf0  53 53 48 20 50 52 49 56 41 54 45 20 4b 45 59 20   SSH PRIVATE KEY
fff39d00  46 49 4c 45 20 46 4f 52 4d 41 54 20 31 2e 31 0a   FILE FORMAT 1.1.
fff39d10  00 00 00 00 00 00 00 00 03 80 03 80 db a1 4a 7f   ..............J
fff39d20  76 28 ab 0e 05 c0 3f 69 d8 a0 57 a0 f4 21 99 e5   v(....?i..W..!..
fff39d30  63 86 1f 8e 65 41 bb 9a e2 c1 38 59 be 5c cd 60   c...eA....8Y.\.`
fff39d40  bd 31 5f 61 73 ea 62 6a 23 85 bc 98 c0 9a 61 ed   .1_as.bj#.....a.

tty=noneProCurve Switch 2824=> read 0xfff39cd0
fff39cd0  68 6f 73 74 5f 73 73 68 31 00 ff ff ff ff ff ff   host_ssh1.......
fff39ce0  00 00 01 d4 00 00 00 4d ff f3 9e d0 00 ff ff ff   .......M........
fff39cf0  53 53 48 20 50 52 49 56 41 54 45 20 4b 45 59 20   SSH PRIVATE KEY
fff39d00  46 49 4c 45 20 46 4f 52 4d 41 54 20 31 2e 31 0a   FILE FORMAT 1.1.
fff39d10  00 00 00 00 00 00 00 00 03 80 03 80 db a1 4a 7f   ..............J
fff39d20  76 28 ab 0e 05 c0 3f 69 d8 a0 57 a0 f4 21 99 e5   v(....?i..W..!..
fff39d30  63 86 1f 8e 65 41 bb 9a e2 c1 38 59 be 5c cd 60   c...eA....8Y.\.`
fff39d40  bd 31 5f 61 73 ea 62 6a 23 85 bc 98 c0 9a 61 ed   .1_as.bj#.....a.
fff39d50  43 05 7c b8 d0 e8 7e 47 31 45 ce 46 25 4f 39 d3   C.|...~G1E.F%O9.
fff39d60  80 b7 30 e3 de 34 1c cd 77 6b 56 06 a0 d5 ba cd   ..0..4..wkV.....
```

```
fff39d70  72 09 dd 8e 38 7b 3e 98 68 db f9 88 da cb 20 a1   r...8{>.h..... .
fff39d80  74 0c 0a 0b 10 61 95 d6 15 6e 71 61 00 06 23 00   t....a...nqa..#.
fff39d90  00 00 09 68 6f 73 74 5f 73 73 68 31 59 57 59 57   ...host_ssh1YWYW
fff39da0  03 7d 19 19 bf 5f 06 30 88 93 e3 66 6d a5 b2 5b   .}..._.0...fm..[
fff39db0  77 ba 9f 8e cf c2 71 c6 2f 7d fc f1 91 c8 8e f1   w.....q./}......
fff39dc0  82 c8 6d 86 f2 e6 7c 05 a4 80 2a 81 2f d1 9d a8   ..m...|...*./...

tty=noneProCurve Switch 2824=> read
fff39dd0  e2 5a 99 ab 3e 64 42 2c 83 48 1c 0c a4 66 ac 06   .Z..>dB,.H...f..
fff39de0  9c 90 24 ab ba d3 5b 2c ff 88 02 ff 2b 1f 45 25   ..$...[,....+.E%
fff39df0  13 da ba e1 7e eb 8e 8c 66 c2 4d 4f 48 e2 12 80   ....~...f.MOH...
fff39e00  50 c6 82 d7 56 5f 4d 11 c6 c1 de d7 c0 c8 2a 57   P...V_M.......*W
fff39e10  7c fb 01 bf 4c 1d 61 af 21 9e f1 0a 9d ea f8 43   |...L.a.!......C
fff39e20  a5 fa 6f 13 5a b7 d4 35 01 27 89 53 06 2e 5b 03   ..o.Z..5.'.S..[.
fff39e30  bc 91 e7 ba 51 3d 6a 81 f0 d8 09 61 0d be db 81   ....Q=j....a....
fff39e40  dd db 3f 7b ea 58 10 8e e0 10 7c ee 01 c0 dc 96   ..?{.X....|.....
fff39e50  d6 06 0f a6 62 c9 ef a7 55 71 0e 10 8b 5d 58 7e   ....b...Uq...]X~
fff39e60  64 cf bd f1 6e 5d df 31 e1 40 d4 93 8b db 53 67   d...n].1.@....Sg
fff39e70  28 df 1c 20 80 ab 26 6c 09 32 a7 6c 93 c0 62 69   (.. ..&l.2.l..bi
fff39e80  b9 3e e7 45 25 c3 01 c0 fe e3 09 be c0 65 91 8e   .>.E%........e..
fff39e90  f4 c9 41 a9 14 dc e9 a0 6b 7c fe ac 1c 75 79 ee   ..A.....k|...uy.
fff39ea0  de 6e c2 ff 7e 29 ce 8e c0 9e 57 ff 2a d8 b6 14   .n..~)....W.*...
fff39eb0  bb 82 e3 ac ea 43 eb 2a 50 18 05 bf bb ab c6 0b   .....C.*P.......
fff39ec0  00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff   ................
```

Appendix F. ProCurve Switch 2610-48 investigation

Output listing 6-11: Procurve 2610-48 (J9088A), nvfsdir, nvfserase and nvfsdfill command playaround

```
ProCurve Switch 2610-48# edomtset
ProCurve Switch 2610-48# edomtset
ProCurve Switch 2610-48$ nvfsdir
     filename    |  size  |   date  | writeInProg | active | nextEntry
      .bootblock |   1248 |      -1 |    x00      |  xff   | xbcee0500
          rbtcnt |      4 |       8 |    x00      |  x00   | xbcee0530
    config00.cfg |     18 |       8 |    x00      |  xff   | xbcee0570
    config00.dlt |     13 |       8 |    x00      |  xff   | xbcee05a0
    config01.cfg |      0 |       8 |    x00      |  x00   | xbcee05c0
       index.cfg |    256 |       8 |    x00      |  xff   | xbcee06e0
    config01.dlt |  10240 |       8 |    x00      |  x00   | xbcee2f00
    config01.cfg |   7842 |       8 |    x00      |  xff   | xbcee4dd0
          rbtcnt |      4 |       8 |    x00      |  x00   | xbcee4e00
          rbtcnt |      4 |       8 |    x00      |  x00   | xbcee4e30
          rbtcnt |      4 |       8 |    x00      |  x00   | xbcee4e60
          iflags |     52 |      27 |    x00      |  xff   | xbcee4ec0
       host_ssh2 |   1675 |       0 |    x00      |  xff   | xbcee5570
   host_ssh2.pub |    380 |       0 |    x00      |  xff   | xbcee5710
    config01.dlt |  10240 |      62 |    x00      |  xff   | xbcee7f30
          rbtcnt |      4 |       8 |    x00      |  xff   | xffffffff
diskSize=x100000, freeSize=xfa980, freeSizeWOCompaction=xf8080
ProCurve Switch 2610-48$ fs nvfswal
   A        addr           filename      size      date  flgs
  --  ----------  ----------------  --------  --------  ----
  **  0xbcee0000        .bootblock  000004e0  ffffffff  ffff
      0xbcee0500            rbtcnt  00000004  00000008  ffff
  **  0xbcee0530      config00.cfg  00000012  00000008  fffe
  **  0xbcee0570      config00.dlt  0000000d  00000008  fffe
      0xbcee05a0      config01.cfg  00000000  00000008  ffff
  **  0xbcee05c0         index.cfg  00000100  00000008  ffff
      0xbcee06e0      config01.dlt  00002800  00000008  ffff
  **  0xbcee2f00      config01.cfg  00001ea2  00000008  ffff
      0xbcee4dd0            rbtcnt  00000004  00000008  ffff
      0xbcee4e00            rbtcnt  00000004  00000008  ffff
      0xbcee4e30            rbtcnt  00000004  00000008  ffff
  **  0xbcee4e60            iflags  00000034  0000001b  ffff
  **  0xbcee4ec0         host_ssh2  0000068b  00000000  ffff
  **  0xbcee5570     host_ssh2.pub  0000017c  00000000  ffff
  **  0xbcee5710      config01.dlt  00002800  0000003e  ffff
  **  0xbcee7f30            rbtcnt  00000004  00000008  ffff

ProCurve Switch 2610-48$ read 0xbcee5710
bcee5710  636f6e66 69673031 2e646c74 00ffffff
bcee5720  00002800 0000003e bcee7f30 00ffffff
bcee5730  00028900 010a0002 89000d53 4e4d504e
bcee5740  4f544946 5920280a 00028900 0d524f57
bcee5750  5f535441 5455533d 310a0002 8900124e
bcee5760  414d453d 7e737461 636b7472 6170737e
bcee5770  0a000289 000e5441 473d7374 61636b74
```

```
bcee5780  7261700a 00028900 0e53544f 52414745
bcee5790  54595045 3d350a00 028b0012 534e4d50
bcee57a0  5633434f 4d4d554e 49545920 280a0002
bcee57b0  8b000d52 4f575f53 54415455 533d310a
bcee57c0  00028b00 094e414d 453d7e31 7e0a0002
bcee57d0  8b001343 4f4d4d5f 4e414d45 3d7e7075
bcee57e0  626c6963 7e0a0002 8b002553 45435f4e
bcee57f0  414d453d 7e436f6d 6d756e69 74794d61
bcee5800  6e616765 72526561 64577269 74657e0a
ProCurve Switch 2610-48$ sm
  access:w, display:w, read_length:256
ProCurve Switch 2610-48$ nvfserase
ProCurve Switch 2610-48$ read 0xbcee5710
bcee5710  ffffffff ffffffff ffffffff ffffffff
bcee5720  ffffffff ffffffff ffffffff ffffffff
bcee5730  ffffffff ffffffff ffffffff ffffffff
bcee5740  ffffffff ffffffff ffffffff ffffffff
bcee5750  ffffffff ffffffff ffffffff ffffffff
bcee5760  ffffffff ffffffff ffffffff ffffffff
bcee5770  ffffffff ffffffff ffffffff ffffffff
bcee5780  ffffffff ffffffff ffffffff ffffffff
bcee5790  ffffffff ffffffff ffffffff ffffffff
bcee57a0  ffffffff ffffffff ffffffff ffffffff
bcee57b0  ffffffff ffffffff ffffffff ffffffff
bcee57c0  ffffffff ffffffff ffffffff ffffffff
bcee57d0  ffffffff ffffffff ffffffff ffffffff
bcee57e0  ffffffff ffffffff ffffffff ffffffff
bcee57f0  ffffffff ffffffff ffffffff ffffffff
bcee5800  ffffffff ffffffff ffffffff ffffffff
ProCurve Switch 2610-48$ fs nv
   A        addr          filename      size      date  flgs
  --  ----------  ----------------  --------  --------  ----
  **  0xbcee0000        .bootblock  000004e0  ffffffff  ffff

ProCurve Switch 2610-48$ nvfsdir
     filename    |  size  |   date  | writeInProg | active | nextEntry
      .bootblock |   1248 |      -1 |    x00      |  xff   | xffffffff
diskSize=x100000, freeSize=xffae0, freeSizeWOCompaction=xffae0
ProCurve Switch 2610-48$ nvfsfill
Current diskSize=x100000, freeSize=xffae0, freeSizeWOCompaction=xffae0
opened file "fill000". size=1047264, status=1, fd=x85f55b50
wrote file. size=1047264, seed=0, status=5
closed file. closeStatus=1, llStatus=1
test failed at iteration 1
New diskSize=x100000, freeSize=x1d0, freeSizeWOCompaction=x1d0
ProCurve Switch 2610-48$ nvfsfill
Current diskSize=x100000, freeSize=x1d0, freeSizeWOCompaction=x1d0
opened file "fill000". size=464, status=1, fd=x85f55950
wrote file. size=464, seed=0, status=1
closed file. closeStatus=1, llStatus=1
re-opened file. status=1, fd=x85f55950
read file. status=1, bytesRead=464
closed file. closeStatus=1, llStatus=1
```

```
test passed
New diskSize=x100000, freeSize=xff8f0, freeSizeWOCompaction=x0
ProCurve Switch 2610-48$ nvfsfill
Current diskSize=x100000, freeSize=xff8f0, freeSizeWOCompaction=x0
opened file "fill000". size=1046768, status=1, fd=x85f55b10
wrote file. size=1046768, seed=0, status=1
closed file. closeStatus=1, llStatus=1
re-opened file. status=1, fd=x85f55b10
read file. status=1, bytesRead=1046768
closed file. closeStatus=1, llStatus=1
test passed
New diskSize=x100000, freeSize=x1d0, freeSizeWOCompaction=x1d0
ProCurve Switch 2610-48$
ROM information:
   Build directory: /sw/rom/build/nemorom(ndx)
   Build date:      Nov 28 2007
   Build time:      16:36:54
   Build version:   R.10.06
   Build number:    14201

OS identifier found at @ 0xbc020000
Verifying Image validity ...
CRC on OS image header Passed
CRC on complete OS image file Passed
Valid OS image @ 0xbc020000

Decompressing...done.
CRC of image is 0xb8269f59
CRC @ 0x80001000 Len 10811056 is 0xb8269f59




initializing...initialization done.
```

Output listing 6-12: **Procurve 2610-48 (J9088A), marker inspection (using Sanitty) after HP_2626_BUTTON procedure**

```
<SNIP>
===========================================================================
NODE INFO
Filename:           config01.dlt
Address:            0xbcee5680
Is Active?:         No
Next node address:  0xbcee7ea0
Date:               0x00 0x00 0x00 0x3f
Active flags:       0x00 0x00
Size [bytes]:       10240
Data, first 32bytes are header, next bytes are data:
bcee5680  63 6f 6e 66 69 67 30 31 2e 64 6c 74 00 ff ff ff   config01.dlt....

bcee5690  00 00 28 00 00 00 00 3f bc ee 7e a0 00 00 ff ff   ..(....?..~.....

bcee56a0  00 02 89 00 01 0a 00 02 89 00 0d 53 4e 4d 50 4e   ...........SNMPN

bcee56b0  4f 54 49 46 59 20 28 0a 00 02 89 00 0d 52 4f 57   OTIFY (......ROW

bcee56c0  5f 53 54 41 54 55 53 3d 31 0a 00 02 89 00 12 4e   _STATUS=1......N

bcee56d0  41 4d 45 3d 7e 73 74 61 63 6b 74 72 61 70 73 7e   AME=~stacktraps~

bcee56e0  0a 00 02 89 00 0e 54 41 47 3d 73 74 61 63 6b 74   ......TAG=stackt

bcee56f0  72 61 70 0a 00 02 89 00 0e 53 54 4f 52 41 47 45   rap......STORAGE

bcee5700  54 59 50 45 3d 35 0a 00 02 8b 00 12 53 4e 4d 50   TYPE=5......SNMP

bcee5710  56 33 43 4f 4d 4d 55 4e 49 54 59 20 28 0a 00 02   V3COMMUNITY (...

bcee5720  8b 00 0d 52 4f 57 5f 53 54 41 54 55 53 3d 31 0a   ...ROW_STATUS=1.

bcee5730  00 02 8b 00 09 4e 41 4d 45 3d 7e 31 7e 0a 00 02   .....NAME=~1~...

bcee5740  8b 00 13 43 4f 4d 4d 5f 4e 41 4d 45 3d 7e 70 75   ...COMM_NAME=~pu

bcee5750  62 6c 69 63 7e 0a 00 02 8b 00 25 53 45 43 5f 4e   blic~.....%SEC_N

bcee5760  41 4d 45 3d 7e 43 6f 6d 6d 75 6e 69 74 79 4d 61   AME=~CommunityMa

bcee5770  6e 61 67 65 72 52 65 61 64 57 72 69 74 65 7e 0a   nagerReadWrite~.

bcee5780  00 02 8b 00 0e 53 54 4f 52 41 47 45 54 59 50 45   .....STORAGETYPE

bcee5790  3d 32 0a 00 02 8b 00 02 29 0a 00 02 8b 00 02 29   =2......)......)

bcee57a0  0a 00 02 8b 00 01 0a 00 03 da 00 0d 44 48 43 50   ............DHCP

bcee57b0  53 4e 4f 4f 50 72 20 28 0a 00 03 da 00 0d 52 4f   SNOOPr (......RO

bcee57c0  57 5f 53 54 41 54 55 53 3d 33 0a 00 03 da 00 02   W_STATUS=3......

bcee57d0  29 0a 00 03 da 00 01 0a fe 02 00 04 00 16 4e 41   ).............NA

bcee57e0  4d 45 3d 7e 4d 41 52 4b 73 59 63 4b 70 79 66 66   ME=~MARKsYcKpyff
```

```
bcee57f0  4e 79 7e 0a 00 02 98 00 01 0a 00 02 98 00 12 53   Ny~...........S
bcee5800  4e 4d 50 56 33 43 4f 4d 4d 55 4e 49 54 59 20 28   NMPV3COMMUNITY (
bcee5810  0a 00 02 98 00 0d 52 4f 57 5f 53 54 41 54 55 53   ......ROW_STATUS
bcee5820  3d 31 0a 00 02 98 00 09 4e 41 4d 45 3d 7e 32 7e   =1......NAME=~2~
bcee5830  0a 00 02 98 00 1b 43 4f 4d 4d 5f 4e 41 4d 45 3d   ......COMM_NAME=
bcee5840  7e 4d 41 52 4b 63 48 55 41 66 6d 73 42 49 68 7e   ~MARKcHUAfmsBIh~
bcee5850  0a 00 02 98 00 25 53 45 43 5f 4e 41 4d 45 3d 7e   .....%SEC_NAME=~
bcee5860  43 6f 6d 6d 75 6e 69 74 79 4f 70 65 72 61 74 6f   CommunityOperato
bcee5870  72 52 65 61 64 4f 6e 6c 79 7e 0a 00 02 98 00 0e   rReadOnly~......
bcee5880  53 54 4f 52 41 47 45 54 59 50 45 3d 32 0a 00 02   STORAGETYPE=2...
bcee5890  98 00 02 29 0a 00 02 a0 00 08 53 4e 4d 50 53 20   ...)......SNMPS
bcee58a0  28 0a 00 02 a0 00 0d 52 4f 57 5f 53 54 41 54 55   (......ROW_STATU
bcee58b0  53 3d 31 0a 00 02 a0 00 09 43 4f 4d 5f 49 44 3d   S=1......COM_ID=
bcee58c0  32 0a 00 02 a0 00 16 4e 41 4d 45 3d 7e 4d 41 52   2......NAME=~MAR
bcee58d0  4b 63 48 55 41 66 6d 73 42 49 68 7e 0a 00 02 a0   KcHUAfmsBIh~....
bcee58e0  00 07 56 49 45 57 3d 33 0a 00 02 a0 00 02 29 0a   ..VIEW=3......).
bcee58f0  00 02 a0 00 01 0a fe ff ff ff ff ff ff ff ff ff   ................
bcee5900  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
bcee5910  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
bcee5800  4e 4d 50 56 33 43 4f 4d 4d 55 4e 49 54 59 20 28   NMPV3COMMUNITY (
bcee5810  0a 00 02 98 00 0d 52 4f 57 5f 53 54 41 54 55 53   ......ROW_STATUS
bcee5820  3d 31 0a 00 02 98 00 09 4e 41 4d 45 3d 7e 32 7e   =1......NAME=~2~
bcee5830  0a 00 02 98 00 1b 43 4f 4d 4d 5f 4e 41 4d 45 3d   ......COMM_NAME=
bcee5840  7e 4d 41 52 4b 63 48 55 41 66 6d 73 42 49 68 7e   ~MARKcHUAfmsBIh~
bcee5850  0a 00 02 98 00 25 53 45 43 5f 4e 41 4d 45 3d 7e   .....%SEC_NAME=~
bcee5860  43 6f 6d 6d 75 6e 69 74 79 4f 70 65 72 61 74 6f   CommunityOperato
bcee5870  72 52 65 61 64 4f 6e 6c 79 7e 0a 00 02 98 00 0e   rReadOnly~......
bcee5880  53 54 4f 52 41 47 45 54 59 50 45 3d 32 0a 00 02   STORAGETYPE=2...
bcee5890  98 00 02 29 0a 00 02 a0 00 08 53 4e 4d 50 53 20   ...)......SNMPS
bcee58a0  28 0a 00 02 a0 00 0d 52 4f 57 5f 53 54 41 54 55   (......ROW_STATU
bcee58b0  53 3d 31 0a 00 02 a0 00 09 43 4f 4d 5f 49 44 3d   S=1......COM_ID=

bcee58c0  32 0a 00 02 a0 00 16 4e 41 4d 45 3d 7e 4d 41 52   2......NAME=~MAR

bcee58d0  4b 63 48 55 41 66 6d 73 42 49 68 7e 0a 00 02 a0   KcHUAfmsBIh~....

bcee58e0  00 07 56 49 45 57 3d 33 0a 00 02 a0 00 02 29 0a   ..VIEW=3......).

bcee58f0  00 02 a0 00 01 0a fe ff ff ff ff ff ff ff ff ff   ................

bcee5900  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................

<SNIP>

============================================================================
NODE INFO
Filename:            mgrinfo.txt
Address:             0xbcee7ea0
Is Active?:          No
Next node address:   0xbcee7f20
Date:                0x00 0x00 0x00 0x63
Active flags:        0x00 0x00
Size [bytes]:        96
Data, first 32bytes are header, next bytes are data:
bcee7ea0  6d 67 72 69 6e 66 6f 2e 74 78 74 00 ff ff ff ff   mgrinfo.txt.....

bcee7eb0  00 00 00 60 00 00 00 63 bc ee 7f 20 00 00 ff ff   ...`...c..□ ....

bcee7ec0  00 00 00 01 46 4c 47 00 4d 41 52 4b 67 4f 74 73   ....FLG.MARKgOts

bcee7ed0  79 6a 52 73 62 4f 00 00 00 00 00 00 00 00 00 00   yjRsbO..........

bcee7ee0  00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff   ................

bcee7ef0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

bcee7f00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

bcee7f10  00 00 00 00 00 00 00 00 00 00 00 02 06 00 00 00   ................

============================================================================
```

Appendix G. Source Code for MPC862 mem controller decoder

Output listing 6-13: MPC862 memory controller configuration decoder in Java

```
 package termit;

/**
 * @author magnus at-sign stril.com
 * Class for decoding a Freescale MPC862 Base
 * Register and address mask in the Memory Controller Names and
 * nomenclature according to
 * "MPC862 PowerQUICC Integrated Communications Processor Family Reference Manual,
Rev. 3"
 *  chapter 15. The spec seems to have "bit0" as the MSB in the register
 *  which is a bit strange.
 *
 *
 */

public class MPC862memoryControl {
   public enum PORT_SIZE {
      BIT32, BIT8, BIT16, RESERVED
   };

   public enum MACHINE_SELECT {
      GPCM, RESERVED, UPMA, UPMB
   };

   private long register; // register number 0-7
   private long br; // BR field 16bits
   private long or; // OR field 16bits
   static MPC862memoryControl[] mems =
         new MPC862memoryControl[8];
   /**
    * @param args
    */
   public static void main(String[] args) {

      String[] br_strings = new String[] { "0xfff00401", "0x00000081",
            "0x040000c1", "0x050000c1", "0x02000081", "0x00000000",
            "0x60000901", "0x68000401" };

      String[] or_strings = new String[] { "0xfff005a6", "0x7e000600",
            "0x7f000600", "0x7f000600", "0x7e000600", "0x00000000",
            "0xfe000190", "0xfff001a8" };

      for (int i = 0; i < br_strings.length; i++) {
         MPC862memoryControl mem = new MPC862memoryControl(i,
               Long.decode(br_strings[i]), Long.decode(or_strings[i]));
         mems[i] = mem;
         System.out.println(mem);
      }
```

```
        // Below needs some further debugging to work
        /*
         * System.out.println("RAM 0x10000 is controlled by # " +
         * findController(0x10000).getRegisterNumber());
         * System.out.println("EEPROM 0x68000000 is controlled by # " +
         * findController(0x68000000).getRegisterNumber());
         * System.out.println("Flash 0x60000000 is controlled by # " +
         * findController(0x60000000).getRegisterNumber());
         */

    }

    public MPC862memoryControl(int register, long br, long or) {
        super();
        this.register = register;
        this.br = br;
        this.or = or;
    }

    public long getRegisterNumber() {
        return this.register;
    }

    public long getBa() {
        // MSB bits 0-16
        return extractBitValue(br, 15, 31);
    }

    public long getAt() {
        return extractBitValue(br, 12, 14);
    }

    public PORT_SIZE getPs() {
        long size = extractBitValue(br, 10, 11);

        switch ((int) size) {
        case 0:
            return PORT_SIZE.BIT32;
        case 1:
            return PORT_SIZE.BIT8;
        case 2:
            return PORT_SIZE.BIT16;
        case 3:
            return PORT_SIZE.RESERVED;

        default:
            System.out.println("Error: undefined PORT_SIZE: "
                    + String.valueOf(size));
            System.exit(1);
        }

        // should never get here...
        return PORT_SIZE.RESERVED;
    }

    public boolean getParity() {
        return extractBitValue(br, 9, 9) == 1;
    }

    public boolean getWp() {
        return extractBitValue(br, 8, 8) == 1;
    }

    public MACHINE_SELECT getMs() {
        long ms = extractBitValue(br, 6, 7);
        switch ((int) ms) {
        case 0:
            return MACHINE_SELECT.GPCM;
        case 1:
            return MACHINE_SELECT.RESERVED;
        case 2:
            return MACHINE_SELECT.UPMA;
        case 3:
            return MACHINE_SELECT.UPMB;

        default:
            System.out.println("Error: undefined MACHINE_SELECT: "
                    + String.valueOf(ms));
            System.exit(1);
        }

        // should never get here...
        System.exit(1);
        return MACHINE_SELECT.RESERVED;

    }

    public long getReserved() {
        return extractBitValue(br, 1, 5);
    }

    public boolean getValid() {
        return extractBitValue(br, 0, 0) == 1;
    }

    public long getBr() {
        return br;
    }

    // OR fields access methods
    public long getMA() {
        return extractBitValue(or, 15, 31);
    }

    public String toString() {
        String str = "****Memory Control Register # " + register
```

```
            + ": Base Register 0x" + Long.toHexString(br)
            + " Option Register 0x" + Long.toHexString(or) + "\n";

        str += "BA, Baseaddress:\t0x" + Long.toHexString(getBa()) + "\t"
            + Long.toBinaryString(getBa()) + "\n";
        str += "MA, Maskaddress:\t0x" + Long.toHexString(getMA()) + "\t"
            + Long.toBinaryString(getMA()) + "\n";
        str += "AT, Address Type:\t" + new Long(getAt()) + "\n";
        str += "PS, Portsize \t\t" + getPs() + "\n";
        str += "Parity:\t\t\t" + getParity() + "\n";
        str += "Write Protect:\t\t" + getWp() + "\n";
        str += "MS, Machine Select:\t" + getMs() + "\n";
        str += "Reserved (should be 0):\t" + new Long(getReserved()) + "\n";
        str += "Valid:\t\t\t" + getValid() + "\n";

        return str;
    }

    // true if this controller matches an address (32 bit)
    public boolean matchesAddress(long a) {
        a = a >>> 14; // remove the 14 lsb which are not used in the compare
        long mask = this.getMA();
        long x = this.getBa() & mask;
        long y = a & mask;
        boolean z = x == y;

        return ((this.getBa() & mask) == (a & mask));

    }

    // returns the controller for an address
    static public MPC862memoryControl findController(long address) {
        MPC862memoryControl m;
        for (int i = 0; i <= 7; i++) {
            m = mems[i];
            if (m.matchesAddress(address) && m.getValid()) {
                return mems[i];
            }
        }
        return null;
    }

    // extracts the value between and including startbit and end bit inside
    // data.
    // bits start counting at 0
    static long extractBitValue(long data, int startbit, int endbit) {
        // mask away everything above the endbit
        long mask = (2 << endbit) - 1;
        data = data & mask;

        // shift everything right so startbit becomes first bit
        data = data >>> startbit;
        return data;
    }

}
```

Appendix H. Sanitty source code

Output listing 6-14: term.h

```
/*
* Utilities for dealing with serial port terminal communication.
* Author and copyright Magnus Larsson, magnus [at] stril.com
* Written as part of master thesis project 2015 in Electrical Engineering from KTH.
* This code is HIGHLY experimental and not very robust in terms of input checks.
* No warranties given.
*/



#ifndef term_INCLUDED
#define term_INCLUDED

#include <stdio.h> //FILE

//Constants
//static const int READ_CHUNK = 4096;  //amount to try and read
static const int READ_DELAY = 50;  //delay in millisec between two reads
static const int OS_READ_BUFFER_SIZE = 4096;

//Buffer to store read data into while waiting for a string
#define term_READ_BUFFER_SIZE 1024*1024

//Error constants
#define term_err_NO_CMD_ECHO -1
#define term_err_BUFFER_OVERFLOW -2
#define term_err_TIMEOUT -3
#define term_err_NO_PROMPT_FOUND -4

 static const int TERM_ALLOWED_SPEEDS[] = { 110, 300, 600, 1200, 2400, 4800, 9600,
19200, 38400, 57600, 115200 };
 static const char* TERM_ALLOWED_SPEEDS_STR[] = { "110", "300", "600", "1200",
"2400", "4800", "9600", "19200", "38400", "57600", "115200" };
 static char* term_newline_str="\n\r";



 //get/set new line str;
 void term_setLineDelim(char*);
 char* term_getLineDelim();

//get/set the prompt to be used by cmdXXXX methods
 void term_setPrompt(char*);
 char* term_getPrompt();

// ===================================================
//@{
// Does the following
// 1. Empties the receive buffer.
// 2. Sends a command
// 3. Waits for the new prompt.
```

```
 // 4 Checks so the command itself is echoed
//
// @param cmd String to send. terminal_getNewLineStr() will be added automatically
to the end.
// @param prompt String to wait for. If NULL, the prompt set by
term_setPrompt(char*) will be used.
// @returns 0 for success. -1 of the command couldn't be found in the echo stream, -
2 for buffer overrun, -3 for timeout .
//@}
// ================================================
int term_sendCmd(char* cmd, char* prompt);



// ================================================
//@{
// Does the following
// 1. Empties the receive buffer.
// 2. Sends a command
// 3. Waits for the new prompt.
// 4 Returns data received between the end of the command and the beginning of the
new prompt following the result
//
// @param cmd String to send. terminal_getNewLineStr() will be added automatically
to the end.
// @param prompt String to wait for. If NULL, the prompt set by
term_setPrompt(char*) will be used.
// @param prompt res res should have a size of at least term_READ_BUFFER_SIZE
// @returns Number of chars written into res,-1 of the command couldn't be found in
the echo stream, -2 for buffer overrun, -3 for timeout, -4 Prompt could not be
recovered
//@}
// ================================================

int term_sendCmdGetRes(char* cmd, char* prompt, char* res );

// ================================================
//@{
// Read whatever is in the os receive buffer. Non blocking.
// buf mues have space for at least OS_READ_BUFFER_SIZE bytes.
// All other read methods must use this method if the data goes to inputLog.
//
// @param buf Pointer to buffer to write data
// @param nbytes Max number of bytes to read
// @returns -1 on error, or number of bytes read.
//@}
// ================================================
 int term_read(char* buf);

 //empties the read buffer.
 void term_emptyReadBuffer();

// ================================================
//@{
// Write a null terminated string. Blocks until all data is written
//
```

```
// @param string String to write
// @returns 1 on success. 0 on error.
//@}
// ================================================
int term_send(char* string);

//set timeout used for response waiting routines
void term_setTimeout(long ms);

//comport should be previously opened with RS232_OpenComport()...
//void term_setComPort(int comport);


//return 1 on fail
int term_open(int comport, int speed, char *mode);


void term_changeSpeed(int speed);
void term_close();


// ================================================
//@{
// get the current prompt line from the device.
// Sends a term_getLineDelim() waits 250ms and then returns
// a new malloc:ed string with the prompt line wihtout any surrounding
term_getLineDelim()

// @returns a new malloc:ed string with the prompt line wihtout any surrounding
term_getNewlineStr()
// @returns NULL on failiure
//@}
// ================================================
char* term_getPromptFromDevice();




// ================================================
//@{
//waitfor a string and discard all data read during wait
//Returns pointer in buffer where the first char of str was found
//or NULL if timeout or read buffer out
// @param str String to look for
// @returns 0 if sucessful, -2 for buffer overrun, -3 for timeout .
//@}
// ================================================
int term_waitfor(char* str);

//waitfor a string and return all data read during wait.

void term_readUntil(char* str);
```

```
//methods for copying out the input to a file.  If set a copy of the input will go
to this file
void term_setTermLog(FILE *stream);
//term_removeInputLog(FILE *stream);


void term_test();


// ================================================
//@{
// Read n bytes
//
// Read n bytes or stop at timeout,
// @param buffer Memory to read to
// @param nbytes Number of bytes to read
// @returns number of bytes read. May be less than nbytes if timeouted
//@}
// ================================================
int term_readn(char* buffer, int nbytes);


//utility funcitons

// ================================================
//@{
// Extracts a line from a text buffer. getLineDelim() is used a line delimeter
//
// @param buf Text to extract line from
// @param buf_size, buffer siz in bytes
// @param linenum Line to extract. First line is 0
// @returns a new malloced string with the line or NULL if not found
//@}
// ================================================
char* term_util_extractLine(char* buf, int buf_size, int linenum);



// substring search utility function
// Stolen from https://github.com/rescrv/e/blob/master/memmem.h and modified
// Useful when searching for data when data or key are nut nullterminated.
// @param haystack pointer to seartch from
// @param haystack_len length of data to search
// @param needle pointer to needle
// @param needle_len needle length
// @returns pointer to beginning of needle if found in data. Or NULL if not found.
//@}
// ================================================
void *term_util_memmem(const void *haystack, size_t haystack_len, const void
*needle, size_t needle_len);
```

```
// ================================================
//@{
// Decodes a block of hexdump into binary: should be in the following format:
// bcee0000  2e 62 6f 6f 74 62 6c 6f 63 6b 00 ff ff ff ff ff   .bootblock......
// bcee0010  00 00 04 e0 ff ff ff ff bc ee 05 00 00 ff ff ff   ................
// bcee0020  ff ff ff ff ff ff ff ff 00 00 00 dc ff ff ff ff   ................

// @param str A hexdump format string, can be multiline. Must be null terminated.
// @param buf Memory to write converted bytes to (big enough).

// @returns Number of bytes successfully converted.
//@}
// ================================================
int term_util_hexdump2bytes(char* str, char* buf);



#endif /* term included */
```

Output listing 6-15: pcbench.h

```
/* Procurve bench mode utilities
* Contains utility functions for controlling a HP procurve switch over
* edomtset/bench terminal mode.
* Author and copyright Magnus Larsson, magnus [at] stril.com
* Written as part of master thesis project 2015 in Electrical Engineering from KTH.
* This code is HIGHLY experimental and not very robust in terms of input checks.
* No warranties given.
*/

#ifndef pcbench_INCLUDED
#define pcbench_INCLUDED

#include <inttypes.h>
#include <stdio.h>    /*FILE descriptor*/

#define pcbench_nvfs_node_HEADERSIZE 32 //size of the header structure

struct pcbench_nvfs_node    {
   /*The _nodeAddress is just a local field to remember where this node is on the
flash.*/
   unsigned long _nodeAddress;

   //Below are fields that exist on the actual node in the nv file system
   char *fname;                  //16 byte (incl terminating NULL) filename
   uint32_t datasize;            //size of data filed in bytes
   unsigned char date[4];
   uint32_t nextp;
   unsigned char activeflag[2];  //activeflag[1]=FF means active. 00 means inactive
   uint16_t flags;      //unknown use
   unsigned char* data;
};


//Public variables
static char* pcbench_partnrStr = NULL;

//Provurve model list
#define pcbench_MODEL_J9088A 1



//Procurve model infos
struct pcbench_model_info {
   char*     systemName;                //must match entry from "show system-
information"
   char*     name;
   char*     partnr;
   unsigned longflashBaseAddress;
   unsigned longflashSize;              //in bytes
   char*     flashChipName;

   /*below is the program sequence to unlock the flash chip
   address is relative flashBaseAddress. Eg the abolute address to use will be
   flashBaseAddress+flashProgramUnlockSeqAddr[x]
   */
   unsigned longflashProgramUnlockSeqAddr[64];
   unsigned charflashProgramUnlockSeqData[64];  // byte array
   int          flashProgramUnlockSeqLength;   //should be less than 64
};


// ===================================================
//@{
// Tries to enter bench mode using the following strategy:
// Send \n\n at maxspeed rate to trigger the procurves auto sense feature.
// wait for "Connected at" reply, or prompt
// repeat the above for about 1 minute in case the unit is rebooting.
// If maxspeed rate does not produce a reply, the unit may already be booted.
// Send \n for all speeds below maxrate and see which produces a prompt
// Enter username + password if supplied and asked for
// Disable timeout and terminal length
// @param maxspeed max baud rate for com port
// @param username username
// @param nbytes password
// @returns 0 if successful.
//@}
// ===================================================
int pcbench_enterBenchMode(int maxspeed);


// ===================================================
//@{
// returns the model info parametrs for a given model string.

// @param modelStr A model string equivalent to the pcbench_modelStr
// set during enterBenchMode
// @returns pointer to a info block, or NULL if not found.
//@}
// ===================================================
struct pcbench_model_info *pcbench_getModelInfo(char* modelStr);

// ===================================================
//@{
// Get all the device which is on the model database
// @returns a null terminated pointer array to the models handled.
//@}
// ===================================================
struct pcbench_model_info **pcbench_getModelDB();

// ===================================================
//@{
// Sets the model info for the device to use through out the session.
// This is needed for the flash programming to work.
// @param mod Pointer to the model info to use
//@}
// ===================================================
void pcbench_setCurrentModel(struct pcbench_model_info *mod);

// ================================================
//@{
// Gets the model info for the device currently set.
// This is needed for the flash programming to work.
// @returns Pointer to the model info struct currently used
//@}
// ================================================
struct pcbench_model_info *pcbench_getCurrentModel();


// ================================================
//@{
// Reads bytes from the switch memory.
//
// @param address Address to read from
// @param bytes Number of bytes to read
// @param Memory to store read bytes to
// @returns Number of successfully read bytes
//@}
// ================================================
int pcbench_memReadBytes(unsigned long address, size_t bytes, char* buf);

// ================================================
//@{
// Reads contents from memory and writes it to a file
//
// @param address Address to read from
// @param bytes Number of bytes to read
// @param fd file to write to
// @returns Number of successfully read+written bytes
//@}
// ================================================
int pcbench_memDumpToFile(unsigned long address, size_t bytes, FILE* fd);

// ================================================
//@{
// writes all contents of flash to a file
//
// @param file File to write to
// @returns 0 if successful
//@}
// ================================================
int pcbench_flashDumpToFile(char* file);


// ================================================
//@{
// Writes a byte to the flash memory without verification
//
// @param address Address to start writing to
// @returns 0 if succesful.
```

```
//@}
// ================================================
int pcbench_memWriteByteFlash(unsigned long address, unsigned char byte);


// ================================================
//@{
// Writes a byte to the memory without verification
//
// @param address Address to start writing to
// @returns 0 if succesful.
//@}
// ================================================
int pcbench_memWriteByteFlash(unsigned long address, unsigned char byte);


// ================================================
//@{
// Fetches a NVFS node from the switch
// The storage of the fetched node is malloc:ed so it has to be freed
// later.
//
// @returns Address to the fetched node. Null if unsuccessful.
//@}
// ================================================
struct pcbench_nvfs_node* pcbench_nvfs_fetchNode(unsigned long add);



// ================================================
//@{
// Returns the address of the first nvfs node (.bootblock file)
//
// @returns Address to first nvfs node
//@}
// ================================================
unsigned long pcbench_nvfs_getFirstNodeAddr();


// ================================================
//@{
// Check if a NVFSnode is active (by looking in the active flag)
//
// @param node Pointer to node
// @returns 1 if active, 0 if not active
//@}
// ================================================
int pcbench_nvfs_isActiveNode(struct pcbench_nvfs_node* node);

// ================================================
//@{
// Checks if a NVFSnode is is the last node
//
```

```
// @param node Pointer to node
// @returns 1 if last, ie nodes next pointer is 0xffffffff, 0 if not last
//@}
// ===================================================
int pcbench_nvfs_isLastNode(struct pcbench_nvfs_node* node);

// ===================================================
//@{
// Check if a NVFSnode is sanitized, eg all data portion is 0x00
// The check is done on the local node struct. NOT the switch memory itself.
// @param node Pointer to node
// @returns 1 if sanitized, 0 if not
//@}
// ===================================================
int pcbench_nvfs_isSanitized(struct pcbench_nvfs_node* node);


// ===================================================
//@{
// Sends the edomtset sm mode command to control how the read command
// displays memory and the width of the read access (byte, word, etc).
// The reason for putting this in a separate function is that if
// the current mode setting is equivalent to what we are about to send
// we can avoid sending the same comand again and thus speed up repetative
// large block reads of memory.
//
// @param smCmdStr The sm mode set command string.

// @returns 1 if successful, 0 if problem.
//@}
// ===================================================
pcbench_send_SMconfigSetup(char* cmd);



// ===================================================
//@{
// Overwrites the data portion of this nvfs node with 0x00.
// The writes takes place in the device flash.
// After the writes have been done in the flash the data is
// is syncronized. I.e data is read back fot verification and node
// will point to structure which corresponds to the flash node.
//
// @param node The node to sanitize.

// @returns 0 if successful, -1 if there is a problem.
//@}
// ===================================================
int pcbench_nvfs_sanitizeNode();

// ===================================================
//@{
// Overwrites the data portion of all inactive nodes.
// If the sanitizeActivetoo is passed active nodes except the
//.bootblock node are sanitized too. (I'm uncertain of its use.)
// The writes takes place in the device flash and is verified.
//
// @param sanitizeActivetoo:
// 0 will prevent active nodes to be sanitized
// !0 will also sanitize active nodes except the first .bootblock

// @returns 0 if successful, -1 if there is a problem.
//@}
// ===================================================

int pcbench_nvfs_sanitizeAll(int sanitizeActivetoo);


// ===================================================
//@{
// Checks if a nv fsnode is  with 0x00.
// And verifies.
//
// @param node The node to sanitize.

// @returns 0 if successful, -1 if there is a problem.
//@}
// ===================================================
int pcbench_nvfs_sanitizeNode(struct pcbench_nvfs_node* node);


// ===================================================
//@{
// Writes the contents of the nv file chain to a file
// in hexdump style.
//
// @param file filename of output file

// @returns 0 if successful, -1 if there is a problem.
//@}
// ===================================================
int pcbench_nvfs_dumpAllToFile(char* file);


#endif /* pcbench_INCLUDED*/
```

Output listing 6-16: term.c

```
/*
* Utilities for dealing with serial port terminal communication.
* Author and copyright Magnus Larsson, magnus [at] stril.com
* Written as part of master thesis project 2015 in Electrical Engineering from KTH.
* This code is HIGHLY experimental and not very robust in terms of input checks.
* No warranties given.
*/



#include <time.h>
#include <stdio.h> //for inputLog file writing
#include <assert.h>
#include <string.h>
#include "term.h"
#include "log.h"

#include "rs232.h" // Teunis van Beelen's RS232 library


#if defined(__linux__) || defined(__FreeBSD__)
#include <unistd.h> //sleep();
#else
#include <windows.h> //sleep();
#endif

//routines for dealing with the terminal communication

//private variables
static int port=0;
static long timeout = 20000; //millisecs
static time_t  timeZero=0; //time at which timer was started. Initialize to avoid
compiler warning.
static FILE* inputLogFile;
static int maxspeed=115200;
static int speed=0;
static char* mode;
static char* prompt;

//private methods
static void timerReset();  //sets timer to 0;
static long getTimer();    //get time in millisec after last timeReset()

static unsigned char buf[term_READ_BUFFER_SIZE];

static void timerReset() {
   time(&timeZero);
}

static long getTimer() {
   time_t now;
   time(&now);
   return (long)(difftime(now, timeZero) * 1000);
}

void term_setTimeout(long ms) {
   timeout = ms;
   log_msg(LOG_DEBUG, "term_setTimeout: changed timeout");
}

void term_setNewlineStr(char* str) { term_newline_str = str; }
char* term_getLineDelim(){ return term_newline_str; }
void term_setLineDelim(char* str){ term_newline_str = str;  }

void term_setPrompt(char* p) { prompt = p; }
char* term_getPrompt(){ return prompt; }


//comport should be previously opened with RS232_OpenComport()...
/*void term_setComPort(int port) {
   comPort = port;
}*/


int term_open(int comport, int speedIn, char *modeIn) {
   port = comport;
   speed = speedIn;
   mode = modeIn;
   return RS232_OpenComport(port, speed, mode);
}

/*
void term_setMaxSpeed(int speed) {
   maxspeed = speed;
}
*/

void term_changeSpeed(int newSpeed) {
   if (speed != newSpeed) {
      term_close();
      speed = newSpeed;
      term_open(port, speed, mode);
   }
}
void term_close() {
   RS232_CloseComport(port);
}



void term_setTermLog(FILE* f){
   inputLogFile = f;
}

int term_read(char* buf) {
   int i;
```

```
    i = RS232_PollComport(port, buf, OS_READ_BUFFER_SIZE);

    //printf("received %i bytes\n", i);
    if (inputLogFile) {
        for (int j = 0; j < i; j++) {
            fputc(buf[j], inputLogFile);
        }
    }

    fflush(inputLogFile);


    /*If the buffer is filled we might have lost input*/
    if (i == OS_READ_BUFFER_SIZE) {
        log_error("RS232 input buffer filled. Possbile input data loss");
        return -1;
    }
return i;
}

void term_emptyReadBuffer(){
    RS232_PollComport(port, buf, OS_READ_BUFFER_SIZE);
}

//waitfor a string and discard all data read during wait.
//Some data received after the found string may also be discarded from the read
buffer.

int term_waitfor(char* str){
    unsigned char* bufCur=buf;
    char* ptr;
    int i;

    timerReset();

    while (1) {
        if (getTimer() > timeout) {
            log_msgn_NULLTERM(LOG_DEBUG, "Timout waiting for: ", str, NULL);
            return term_err_TIMEOUT;
        }

        /*if another read could overrun our read buffer size*/
        if (bufCur + OS_READ_BUFFER_SIZE >= buf + term_READ_BUFFER_SIZE) {
            log_msgn_NULLTERM(LOG_DEBUG, "Buffer overflow waiting for: ", str, NULL);
            return term_err_BUFFER_OVERFLOW;
        }

        i= term_read(bufCur);
        bufCur += i;


        //Todo optimization: Dont research all buffer from beginning every time
        if (ptr= term_util_memmem(buf, bufCur - buf, str, strlen(str)))
```

```
         return 0; //success

      Sleep(READ_DELAY);
   }

   //should never get here
   assert(0);
   return -100;

}

int term_sendCmdGetRes(char* cmd, char* prompt, char* res){
   char* start;
   char* end;
   int len;
   int ret;
   if (!prompt)
      prompt = term_getPrompt();

   term_emptyReadBuffer();

   term_send(cmd);
   term_send(term_getLineDelim());
   ret = term_waitfor(prompt); //puts received data in buf variable
   if (ret)
      return ret;


   end = term_util_memmem(buf, term_READ_BUFFER_SIZE, prompt, strlen(prompt));
   if (!end) {
      log_msgn_NULLTERM(LOG_DEBUG, "term_sendCmd: Command <", cmd, "> sent. But 
prompt <", prompt, "> never found.", NULL);
      return term_err_NO_PROMPT_FOUND;
   }
   start = term_util_memmem(buf, OS_READ_BUFFER_SIZE, cmd, strlen(cmd));
   if (!start) {
      log_msgn_NULLTERM(LOG_DEBUG, "term_sendCmd: Command <", cmd, "> sent. But echo 
never read.", NULL);
      return term_err_NO_CMD_ECHO;
   }

   start += strlen(cmd); //skip the command itself
   //skip new line after command
   if (!strncmp(start, term_getLineDelim(), strlen(term_getLineDelim()))){
      start += strlen(term_getLineDelim());
   }

   len = end - start;
   memcpy(res, start, len);
   res[len] = 0; //null terminate to make a string
   return len;
}
```

```
int term_sendCmd(char* cmd, char* prompt){
   if (!prompt)
      prompt = term_getPrompt();

   term_emptyReadBuffer();

   term_send(cmd);
   term_send(term_getLineDelim());


      if (term_waitfor(prompt)) {
      log_msgn_NULLTERM(LOG_DEBUG, "term_sendCmd: Command <", cmd, "> sent. But 
prompt <", prompt, "> never found.",  NULL);
      return term_err_NO_PROMPT_FOUND;
   }
      if (!term_util_memmem(buf, OS_READ_BUFFER_SIZE, cmd, strlen(cmd))) {
      log_msgn_NULLTERM(LOG_DEBUG, "term_sendCmd: Command <", cmd, "> sent. But echo 
never read.",  NULL);
      return term_err_NO_CMD_ECHO;
   }
   return 0;
}

//blocking
//return 0 on fail
int term_send(char* string) {

   int l = strlen(string);
   if (RS232_SendBuf(port, string, strlen(string)) != l) {
      return 0;
   }
   return 1;
}



void term_test(){
   log_debug("TimerTest");
   timerReset();

   while (TRUE) {
      printf("%u\n", getTimer());
      Sleep(1000);

   }

}


char* term_getPromptFromDevice(){
   int len;
   //extract the full line prompt
   term_emptyReadBuffer();
   term_send(term_getLineDelim());
   Sleep(250);
   len=term_read(buf);

   //line 0 is new line cho. line 1 is prompt.
   return term_util_extractLine(buf, len, 1);
}


char* term_util_extractLine(char* buf, int buf_size, int line){
   char* end;
   char* lineStr;
   int len;

   end = term_util_memmem(buf, buf_size, term_getLineDelim(),
strlen(term_getLineDelim()));

   if (line==0) {
      if (!end) {
         end = buf + buf_size;  //set end of line at end of buffer
      }
      len = end - buf;
      lineStr = malloc(len + 1);
      assert(lineStr);
      strncpy(lineStr, buf, len);
      lineStr[len] = '\0';
      return lineStr;
   }

   //line >1 but no more rows
   if (!end) {
      return 0;
   }

   // for lines > 1 continue  recursively with the rest of the lines

   return term_util_extractLine(
      end + strlen(term_getLineDelim()),               //advance to beginning of
next line
      buf_size - (end - buf + strlen(term_getLineDelim())),   //remove this line
from text size
      --line);
}



/* Substring search utility function.
* Stolen from https://github.com/rescrv/e/blob/master/memmem.h and modified
* XXX: Partially adapted from code which contianed this
*      copyright:
* Byte-wise substring search, using the Two-Way algorithm.
* Copyright (C) 2008, 2010 Eric Blake
* Permission to use, copy, modify, and distribute this software
* is freely granted, provided that this notice is preserved.
*/



void *term_util_memmem(const void *haystack, size_t haystack_len,
   const void *needle, size_t needle_len)
{
   const char *begin = (const char*)haystack;
   const char *last_possible = begin + haystack_len - needle_len;
   const char *tail = (const char*)needle;
   char point;

   /*
   * The first occurrence of the empty string is deemed to occur at
   * the beginning of the string.
   */
   if (needle_len == 0)
      return (void *)begin;

   /*
   * Sanity check, otherwise the loop might search through the whole
   * memory.
   */
   if (haystack_len < needle_len)
      return NULL;

   point = *tail++;
   for (; begin <= last_possible; begin++) {
      if (*begin == point && !memcmp(begin + 1, tail, needle_len - 1))
         return (void *)begin;
   }

   return NULL;
}




// ==================================================
//@{
// Decodes a block of hexdump into binary: should be in the following format:
// bcee0000  2e 62 6f 6f 74 62 6c 6f 63 6b 00 ff ff ff ff ff   .bootblock......
// bcee0010  00 00 04 e0 ff ff ff ff bc ee 05 00 00 ff ff ff   ................
// bcee0020  ff ff ff ff ff ff ff ff 00 00 00 dc ff ff ff ff   ................

// @param str A hexdump format string, can be multiline. Must be null terminated.
// @param buf Memory to write converted bytes to (big enough).

// @returns Number of bytes successfully converted.
//@}
// ==================================================
int term_util_hexdump2bytes(char* str, char* destbuf){
   char* lineStr;
   char hexbyteStr[3];
   char* tok;
   int line = 0;
   char* delim = " ";
   size_t bytesDecoded=0;
   int bytesPerLine = 16;

   while (lineStr = term_util_extractLine(str, strlen(str), line)) {
      //todo check address continuity
      tok = strstr(lineStr, "  ");
      if (!tok)
         return bytesDecoded; //this row has no double space so it is not a hexdump
data row
      tok += 2;

      //tok now points to the first byte in the line to decode.
      //write all pairs until we hit double space again
      while (!(tok[0] == ' ' && tok[1] == ' ')) {

         if (tok[0] == ' ') //skip SINGLE space between bytes
            tok++;

         hexbyteStr[0] = *tok;
         hexbyteStr[1] = *(tok + 1);
         hexbyteStr[2] = '\0';
         *(destbuf++) = (char)strtoul(hexbyteStr, NULL, 16);
         bytesDecoded++;
         tok += 2;
      }
      free(lineStr);
      line++;

   }
         return bytesDecoded;
}
```

Output listing 6-17: pcbench.h

```
/* Procurve bench mode utilities
* Contains utility functions for controlling a HP procurve switch over
* edomtset/bench terminal mode.
* Author and copyright Magnus Larsson, magnus [at] stril.com
* Written as part of master thesis project 2015 in Electrical Engineering from KTH.
* This code is HIGHLY experimental and not very robust in terms of input checks.
* No warranties given.
*/

#ifndef pcbench_INCLUDED
#define pcbench_INCLUDED

#include <inttypes.h>
#include <stdio.h>    /*FILE descriptor*/

#define pcbench_nvfs_node_HEADERSIZE 32 //size of the header structure

struct pcbench_nvfs_node    {
   /*The _nodeAddress is just a local field to remember where this node is on the
flash.*/
   unsigned long _nodeAddress;

   //Below are fields that exist on the actual node in the nv file system
   char *fname;                   //16 byte (incl terminating NULL) filename
   uint32_t datasize;             //size of data filed in bytes
   unsigned char date[4];
   uint32_t nextp;
   unsigned char activeflag[2];   //activeflag[1]=FF means active. 00 means inactive
   uint16_t flags;      //unknown use
   unsigned char* data;
};


//Public variables
static char* pcbench_partnrStr = NULL;

//Provurve model list
#define pcbench_MODEL_J9088A 1



//Procurve model infos
struct pcbench_model_info {
   char*      systemName;                //must match entry from "show system-
information"
   char*      name;
   char*      partnr;
   unsigned longflashBaseAddress;
   unsigned longflashSize;               //in bytes
   char*      flashChipName;

   /*below is the program sequence to unlock the flash chip
   address is relative flashBaseAddress. Eg the abolute address to use will be
   flashBaseAddress+flashProgramUnlockSeqAddr[x]
   */
   unsigned longflashProgramUnlockSeqAddr[64];
   unsigned charflashProgramUnlockSeqData[64];  // byte array
   int          flashProgramUnlockSeqLength;   //should be less than 64
};




// ================================================
//@{
// Tries to enter bench mode using the following strategy:
// Send \n\n at maxspeed rate to trigger the procurves auto sense feature.
// wait for "Connected at" reply, or prompt
// repeat the above for about 1 minute in case the unit is rebooting.
// If maxspeed rate does not produce a reply, the unit may already be booted.
// Send \n for all speeds below maxrate and see which produces a prompt
// Enter username + password if supplied and asked for
// Disable timeout and terminal length
// @param maxspeed max baud rate for com port
// @param username username
// @param nbytes password
// @returns 0 if successful.
//@}
// ================================================
int pcbench_enterBenchMode(int maxspeed);



// ================================================
//@{
// returns the model info parametrs for a given model string.

// @param modelStr A model string equivalent to the pcbench_modelStr
// set during enterBenchMode
// @returns pointer to a info block, or NULL if not found.
//@}
// ================================================
struct pcbench_model_info *pcbench_getModelInfo(char* modelStr);

// ================================================
//@{
// Get all the device which is on the model database
// @returns a null terminated pointer array to the models handled.
//@}
// ================================================
struct pcbench_model_info **pcbench_getModelDB();

// ================================================
//@{
// Sets the model info for the device to use through out the session.
// This is needed for the flash programming to work.
// @param mod Pointer to the model info to use
//@}
// ================================================
void pcbench_setCurrentModel(struct pcbench_model_info *mod);

// ================================================
//@{
// Gets the model info for the device currently set.
// This is needed for the flash programming to work.
// @returns Pointer to the model info struct currently used
//@}
// ================================================
struct pcbench_model_info *pcbench_getCurrentModel();


// ================================================
//@{
// Reads bytes from the switch memory.
//
// @param address Address to read from
// @param bytes Number of bytes to read
// @param Memory to store read bytes to
// @returns Number of successfully read bytes
//@}
// ================================================
int pcbench_memReadBytes(unsigned long address, size_t bytes, char* buf);

// ================================================
//@{
// Reads contents from memory and writes it to a file
//
// @param address Address to read from
// @param bytes Number of bytes to read
// @param fd file to write to
// @returns Number of successfully read+written bytes
//@}
// ================================================
int pcbench_memDumpToFile(unsigned long address, size_t bytes, FILE* fd);

// ================================================
//@{
// writes all contents of flash to a file
//
// @param file File to write to
// @returns 0 if successful
//@}
// ================================================
int pcbench_flashDumpToFile(char* file);


// ================================================
//@{
// Writes a byte to the flash memory without verification
```

```
//
// @param address Address to start writing to
// @returns 0 if succesful.
//@}
// ================================================
int pcbench_memWriteByteFlash(unsigned long address, unsigned char byte);


// ================================================
//@{
// Writes a byte to the memory without verification
//
// @param address Address to start writing to
// @returns 0 if succesful.
//@}
// ================================================
int pcbench_memWriteByteFlash(unsigned long address, unsigned char byte);


// ================================================
//@{
// Fetches a NVFS node from the switch
// The storage of the fetched node is malloc:ed so it has to be freed
// later.
//
// @returns Address to the fetched node. Null if unsuccessful.
//@}
// ================================================
struct pcbench_nvfs_node* pcbench_nvfs_fetchNode(unsigned long add);



// ================================================
//@{
// Returns the address of the first nvfs node (.bootblock file)
//
// @returns Address to first nvfs node
//@}
// ================================================
unsigned long pcbench_nvfs_getFirstNodeAddr();


// ================================================
//@{
// Check if a NVFSnode is active (by looking in the active flag)
//
// @param node Pointer to node
// @returns 1 if active, 0 if not active
//@}
// ================================================
int pcbench_nvfs_isActiveNode(struct pcbench_nvfs_node* node);

// ================================================
//@{
// Checks if a NVFSnode is is the last node
//
// @param node Pointer to node
// @returns 1 if last, ie nodes next pointer is 0xffffffff, 0 if not last
//@}
// ================================================
int pcbench_nvfs_isLastNode(struct pcbench_nvfs_node* node);

// ================================================
//@{
// Check if a NVFSnode is sanitized, eg all data portion is 0x00
// The check is done on the local node struct. NOT the switch memory itself.
// @param node Pointer to node
// @returns 1 if sanitized, 0 if not
//@}
// ================================================
int pcbench_nvfs_isSanitized(struct pcbench_nvfs_node* node);


// ================================================
//@{
// Sends the edomtset sm mode command to control how the read command
// displays memory and the width of the read access (byte, word, etc).
// The reason for putting this in a separate function is that if
// the current mode setting is equivalent to what we are about to send
// we can avoid sending the same comand again and thus speed up repetative
// large block reads of memory.
//
// @param smCmdStr The sm mode set command string.

// @returns 1 if successful, 0 if problem.
//@}
// ================================================
pcbench_send_SMconfigSetup(char* cmd);




// ================================================
//@{
// Overwrites the data portion of this nvfs node with 0x00.
// The writes takes place in the device flash.
// After the writes have been done in the flash the data is
// is syncronized. I.e data is read back fot verification and node
// will point to structure which corresponds to the flash node.
//
// @param node The node to sanitize.

// @returns 0 if successful, -1 if there is a problem.
//@}
// ================================================
int pcbench_nvfs_sanitizeNode();
```

```
// ================================================
//@{
// Overwrites the data portion of all inactive nodes.
// If the sanitizeActivetoo is passed active nodes except the
//.bootblock node are sanitized too. (I'm uncertain of its use.)
// The writes takes place in the device flash and is verified.
//
// @param sanitizeActivetoo:
// 0 will prevent active nodes to be sanitized
// !0 will also sanitize active nodes except the first .bootblock

// @returns 0 if successful, -1 if there is a problem.
//@}
// ================================================

int pcbench_nvfs_sanitizeAll(int sanitizeActivetoo);



// ================================================
//@{
// Checks if a nv fsnode is  with 0x00.
// And verifies.
//
// @param node The node to sanitize.

// @returns 0 if successful, -1 if there is a problem.
//@}
// ================================================
int pcbench_nvfs_sanitizeNode(struct pcbench_nvfs_node* node);



// ================================================
//@{
// Writes the contents of the nv file chain to a file
// in hexdump style.
//
// @param file filename of output file

// @returns 0 if successful, -1 if there is a problem.
//@}
// ================================================
int pcbench_nvfs_dumpAllToFile(char* file);



#endif /* pcbench_INCLUDED*/
```

Output listing 6-18: pcbench.c

```
/* Procurve bench mode utilities
 * Contains utility functions for controlling a HP procurve switch over
 * edomtset/bench terminal mode.
 * Author and copyright Magnus Larsson, magnus [at] stril.com
 * Written as part of master thesis project 2015 in Electrical Engineering from KTH.
 * This code is HIGHLY experimental and not very robust in terms of input checks.
 * No warranties given.
*/


#include <string.h> /* strstr */
#include <assert.h>
#include <stdlib.h> /* malloc, free*/
#include <time.h>
#include "pcbench.h"
#include "term.h"
#include "log.h"


#if defined(__linux__) || defined(__FreeBSD__)
#include <unistd.h> //sleep();
#else
#include <windows.h> //sleep();
#endif

//Public variables
static struct pcbench_model_info *pcbench_modelsDB[128] = { NULL };

//private variables
static unsigned char buf[term_READ_BUFFER_SIZE];
static struct pcbench_model_info *pcbench_currentModel=NULL;
static char pcbench_lastSMreadConfigCommand[256] = { '\0' }; //last edomtset sm mode
command sent

//private helper functions
void makeModelInfoDB();

int pcbench_enterBenchMode(int maxspeed) {
   int i;
   char* str;
   int len;


   //try to connect at max speed for 6 seconds
   term_changeSpeed(maxspeed);

   //find the maxspeed index
   for (i = 0; TERM_ALLOWED_SPEEDS[i] < maxspeed; i++);

   /*
   2 \r trigger baud sense
   wait 2 sec
   1 \r to get by intro text

   */
   do {
   log_msgn_NULLTERM(LOG_INFO, "Trying to connect with speed ",
TERM_ALLOWED_SPEEDS_STR[i], NULL);
      term_changeSpeed(TERM_ALLOWED_SPEEDS[i]);
      term_send("\r\r");
      Sleep(2000);
      term_send("\r");
      len=term_read(buf);

      /*below code decodes the part# from the entry screen but it is unreliable
        so we get the part# by cmd option for now.

      char *needle = "ProCurve J";
      if (str = term_util_memmem(buf, OS_READ_BUFFER_SIZE, needle, strlen(needle)))
{
         str += strlen(needle) - 1;
         //found a model#
         if (strEnd = memchr(str, ' ', 32)){//search for end of part# space. Max 32
chars

            //the string is in buf memory which is not permanent. Need to copy it and
store in pcbench_modelStr.
            len = strEnd - str;
            pcbench_partnrStr = malloc(len+1);
            assert(pcbench_partnrStr);
            memcpy(pcbench_partnrStr, str, len);
            pcbench_partnrStr[len] = '\0';

         }
      }

      */

      str = "Please Enter Login Name:";
      if (term_util_memmem(buf, OS_READ_BUFFER_SIZE, str, strlen(str))) {
         log_error("Unit has password set. Password entry is not handled by this
tool."
            " Either remove passwords or do a factory reset");
         exit(EXIT_FAILURE);

      }

      if (memchr(buf, '#', len)) {
         goto connected; //enable mode
      }
      if (memchr(buf, '$', len)) {
         goto inBenchMode;  //edomtset mode
      }
      if (memchr(buf, '=', len)) {
         goto inBenchMode;  //bench jumper mode
      }
   } while (--i > 0);

   log_error("Unable to connect to switch.");
   exit(EXIT_FAILURE);


connected:

   log_msgn_NULLTERM(LOG_INFO, "Connected with speed ", TERM_ALLOWED_SPEEDS_STR[i],
NULL);
   term_send("edomtset\r");
   term_send("edomtset\r");


      inBenchMode:
         log_info("Is now in bench mode");
         term_send("setterm ascii\r"); //use ascii to get rid of annoying vt100 codes
         //term_waitfor("$"); //could either be edomtset $ or bench mode jumper =>
promt
         Sleep(500);
         //term_sendCmd("no page", "$"); //terminal length 0
         term_send("no page\r");
         Sleep(500);

         /*Using single char prompts #, $, = is not robust in case they appear in
data.
         From now on we use the full (longer) edomtset prompt to verify command
completion.
         Ie, "ProCurve Switch 2610-48$" instead of just "$"
         */
         str = term_getPromptFromDevice();
         if (!str){
            log_error("could not decode prompt from device");
               exit(EXIT_FAILURE);
         }

         term_setPrompt(str);
         log_msg2(LOG_DEBUG, "Prompt set to: ", str);


         /*As far as I know the only way to get the part# and model# is entry banner
screen
         * Current edomtset point is the only stable point in the above process.
         * But we might be at this point when the switch was connected already in
edomtset mode and then we never passed the
         * login screen. We would eed to back out, active the banner screen, parse
the model# and then
         * simply redo the edomtset entry.

         * Todo: Never got the below to work properly, so the part# will be passed as
a cmd line option instead for now.
         */

         /*
```

```
        if (!pcbench_partnrStr) {
           term_send("logout\n");
           Sleep(1000);
           // Do you want to log out[y / n] ?
           term_send("y");
           term_send(term_newline_str);
           Sleep(1000);
           //Do you want to save current configuration[y / n] ?
           term_send("n");
           term_send(term_newline_str);
           Sleep(1000);
           term_emptyReadBuffer();
           log_debug("reconnecting to get partnr string");
           pcbench_enterBenchMode(maxspeed); // do it all over again to catch the
main screen

        }

        pcbench_currentModel = pcbench_getModelInfo(pcbench_partnrStr);
        if (!pcbench_currentModel)
        {
           log_msgn_NULLTERM(LOG_ERROR, "pcbench_enterBenchMode: Unable to find
device ", pcbench_partnrStr, " in device database", NULL);
           return -1;
        }
        */

        return 0; //success
     }



int pcbench_memReadBytes(unsigned long address, size_t bytes, char* bindest) {

   int readchunk = 0;
   char cmd[128];

   int len;
   size_t readcount=0;

   while (bytes) {

     if (bytes > 256)
        readchunk = 256;
     else
        readchunk = bytes;


     if (sprintf(cmd, "sm -l%u -ab -db", readchunk) < 0) {  //setup of the hex dump
output format including the read length
        log_msgn_NULLTERM(LOG_ERROR, "pcbench_memReadBytes: Could not generate smode
string: ", cmd, NULL);
        return -1;
     }
```

```
        pcbench_send_SMconfigSetup(cmd);

        if (sprintf(cmd, "read 0x%08x", address) < 0) {
            log_msgn_NULLTERM(LOG_ERROR, "pcbench_memReadBytes: Could not generate read
address string: ", cmd, NULL);
            return -1;
        }

        len = term_sendCmdGetRes(cmd, NULL, buf);
        buf[len] = '\0'; // null terminate
            if (term_util_hexdump2bytes(buf, bindest) != readchunk) {
                log_error("pcbench_memReadBytes read error");
                return -1;
            }
        readcount += readchunk;
        bindest += readchunk;
        bytes -= readchunk;
        address += readchunk;

        if (!(readcount % (256 * 256)))
            log_debug("64kB block fetched.");
    }
    return readcount;
}


int pcbench_memDumpToFile(unsigned long address, size_t bytes, FILE* fd) {
    const unsigned  int READBLOCK = 10 * 1024; //10kB readblocks
    char *rbuf = malloc(READBLOCK);
    size_t left = bytes;
    size_t read = 0;
    int ret;
    char msg[256];
    int toRead;

    time_t tStart;
    time_t tEnd;
    double durSec;

    while (left>0) {
        if (left > READBLOCK)
            toRead = READBLOCK;
        else
            toRead = left;

        tStart= time(NULL);

        ret = pcbench_memReadBytes(address, toRead, rbuf);
        if (ret != toRead) {
            free(rbuf);
            return -1;
        }
        left -= ret;
```

```
      read += ret;
      address += ret;
      ret = fwrite(rbuf, sizeof(char), toRead, fd);
      if (ret != toRead) {
         free(rbuf);
         return -1;
      }

      tEnd=time(NULL);
      durSec = difftime(tEnd, tStart);

      printf("\r%u%% complete. Transfer rate: %.0f bytes/sec", read * 100 / bytes,
((double)toRead)/durSec);
   }

   free(rbuf);
   return 0;
}


int pcbench_memWriteByte(unsigned long address, unsigned char b) {
   char cmd[128];

   //setup byte access
   pcbench_send_SMconfigSetup("sm -b");

   if (sprintf(cmd, "wr 0x%08x 0x%02x", address, b) < 0) {
      log_msgn_NULLTERM(LOG_ERROR, "pcbench_memWriteByte: Could not generate wr
command string: ", cmd, NULL);
      return -1;
   }

   if (term_sendCmd(cmd, NULL))
      return -1;

   return 0; //sucess
}


int pcbench_memWriteByteFlash(unsigned long address, unsigned char byte) {

   /*
   if (address >= pcbench_currentModel->flashBaseAddress &&
   address < pcbench_currentModel->flashBaseAddress + pcbench_currentModel-
>flashSize)
   _*/

   //writes must be prepended by the flash unlock sequence
   for (int i = 0; i < pcbench_currentModel->flashProgramUnlockSeqLength; i++) {
      unsigned long a = pcbench_currentModel->flashProgramUnlockSeqAddr[i] +
pcbench_currentModel->flashBaseAddress;
      unsigned char b = pcbench_currentModel->flashProgramUnlockSeqData[i];
      if (pcbench_memWriteByte(a, b)) {
         return -1;
```

```
        }
    }

    /*
    The flash should now be unlocked and can be written. We are writing a byte but it
seems some flash chips will write a whole word.
    As such we might overwrite the next byte too by 0x00 accidently.
    */

    if (pcbench_memWriteByte(address, byte)) {
        return -1;
    }

    return 0; // success
}




unsigned long pcbench_nvfs_getFirstNodeAddr() {
    char* addr;
    char* addr_end;

    switch (term_sendCmdGetRes("fs nvfswalk", NULL, buf)){
        case term_err_NO_CMD_ECHO:
        log_error("pcbench_nvfs_getFirstNodeAddr(): fs nvfswalk command not found in
echo stream");
        return 0;
        case term_err_TIMEOUT:
        log_error("pcbench_nvfs_getFirstNodeAddr(): fs nvfswalk command generated
timout");
        return 0;
        case term_err_BUFFER_OVERFLOW:
            log_error("pcbench_nvfs_getFirstNodeAddr(): fs nvfswalk command generated
buffer overrrun");
            return 0;
    }

    buf[term_READ_BUFFER_SIZE-1] = '\0'; // null terminate to make string searches
safe

    //the first address should be the .bootblock
    addr = strstr(buf, "0x");
    if (!addr)
        return 0;
    addr_end=strchr(addr, ' ');
    if (!addr_end)
        return 0;
    *addr_end= '\0'; //null terminate to make addr a string
    return strtoul(addr, NULL, 16);  //convert from hex to unsigned long
}



struct pcbench_nvfs_node* pcbench_nvfs_fetchNode(unsigned long addr) {
    unsigned char* headmem;
```

```
   struct pcbench_nvfs_node* node;
   int i;

   /*Mapping the struct directly could be risky. Even thoug pragma pack could be
used
   different endian CPU settings could mess upp the integers. Safer to decode them
manually.
   pcbench_nvfs_fetchNode() returnes a malloc:ed data so we can reference inside it
permanently.
   */

   headmem = malloc(pcbench_nvfs_node_HEADERSIZE);
   assert(headmem);

   //fetch the header (32bytes)
   if (pcbench_memReadBytes(addr, pcbench_nvfs_node_HEADERSIZE, headmem) !=
pcbench_nvfs_node_HEADERSIZE)
      return NULL;

   /*
   The first 16 bytes should contain a null terminated filename string.
   */
   for (i = 0; i <= 16 && headmem[i] != '\0'; i++);

   if (i == 16) {
      //no NULL in the first 16 bytes
      free (headmem);
      return NULL; //fail
   }

   node = malloc(sizeof(struct pcbench_nvfs_node));
   assert(node);

   //remember the address of this node
   node->_nodeAddress = addr;

   //keep null terminated string
   node->fname = headmem;

   //datasize is 4 bytes mapped least significant byte first
   node->datasize = (headmem[16] << 24) + (headmem[17] << 16) + (headmem[18] << 8) +
headmem[19];

   //I don't know the meaning of the 4byte date field so just keep it byte ordered
for now
   node->date[0] = headmem[20];
   node->date[1] = headmem[21];
   node->date[2] = headmem[22];
   node->date[3] = headmem[23];

   //4 byte nextpointer mapped least significant byte first
   node->nextp = (headmem[24] << 24) + (headmem[25] << 16) + (headmem[26] << 8) +
headmem[27];
```

```
   node->activeflag[0] = headmem[28];
   node->activeflag[1] = headmem[29];

   //unknown meaning, just copy as is
   node->flags = (headmem[30]<<8) + headmem[31];



   //from the header we can get the data size and malloc enough RAM for the whole
node
   //But first make some sanity checks.

   if (node->datasize > 10000000) {
      log_error("pcbench_nvfs_fetchNode: Sanity check failed. Data size to big");
      return NULL;
   }

   node->data = malloc(node->datasize);
   assert(node->data);

   //copy the data from the switch
   if (pcbench_memReadBytes(addr+32, node->datasize, node->data) != node->datasize)
      return NULL;

   return node;

}

int pcbench_flashDumpToFile(char* file){
   FILE *f;

   //check if file already exists
   if (f = fopen(file, "r")) {
      fclose(f);
      log_msgn_NULLTERM(LOG_ERROR, "File already exists. Could not open ", file ,
NULL);
      return -1;
   }

   if (!(f = fopen(file, "wb"))) {  //b for binary to prevent EOL conversion
   log_msgn_NULLTERM(LOG_ERROR, "Could not open file with r/w access: ",file, NULL);
      return -1;
   }

   if (!pcbench_currentModel)
      return -1;

   if (pcbench_memDumpToFile(pcbench_currentModel->flashBaseAddress,
pcbench_currentModel->flashSize, f))
      return -1;

   fclose(f);
   return 0;
}
```

```
int pcbench_nvfs_isActiveNode(struct pcbench_nvfs_node *node){
   assert(node);
   return (node->activeflag[1] & 0xff);
}

int pcbench_nvfs_isLastNode(struct pcbench_nvfs_node* node) {
   return (node->nextp == 0xffffffff);
}

int pcbench_nvfs_isSanitized(struct pcbench_nvfs_node* node){
   unsigned long i;


   for (i=0; i < node->datasize; i++) {
      if (node->data[i] != 0x00){
         return 0;
      }
   }

   return 1;
}

struct pcbench_model_info *pcbench_getModelInfo(char* modelStr){
   if (!pcbench_modelsDB[0])
      makeModelInfoDB();

   for (int i = 0; pcbench_modelsDB[i]; i++)
      if (!strcmp(pcbench_modelsDB[i]->partnr, modelStr))
         return pcbench_modelsDB[i];

   return NULL;
}

void pcbench_setCurrentModel(struct pcbench_model_info *mod) {
   pcbench_currentModel = mod;
}
struct pcbench_model_info *pcbench_getCurrentModel() {
   return pcbench_currentModel;
}

struct pcbench_model_info **pcbench_getModelDB() {
   if (!pcbench_modelsDB[0])
      makeModelInfoDB();

   return pcbench_modelsDB;
}

void makeModelInfoDB(){

   static struct pcbench_model_info J9088A = {
      .systemName = "ProCurve Switch 2610 - 48",
      .name = "ProCurve Switch 2610 - 48",
      .partnr = "J9088A",
      .flashBaseAddress = (unsigned long) 0xbc000000,
      .flashSize = 0x1000000, //16MB
      .flashChipName = "S29GL128P",
      .flashProgramUnlockSeqAddr = { (unsigned long)0xaaa, (unsigned long)0x555, 
(unsigned long)0xaaa },
      .flashProgramUnlockSeqData = { (unsigned char)0xaa, (unsigned char)0x55, 
(unsigned char)0xA0 },
      .flashProgramUnlockSeqLength = 3
   };


   static struct pcbench_model_info J4900A = {
   .systemName = "ProCurve Switch 2626",
   .name = "ProCurve Switch 2626",
   .partnr = "J4900A",
   .flashBaseAddress = (unsigned long)0x7c800000,
   .flashSize = 0x00800000, //8MB
   .flashChipName = "AM29LV065D",
   //any address in this chip can be written to for the unlock sequence. Only data 
sequence matters.
   //Let's use the base address.
   .flashProgramUnlockSeqAddr = { (unsigned long)0x0, (unsigned long)0x0, (unsigned 
long)0x0 },
   .flashProgramUnlockSeqData = { (unsigned char)0xaa, (unsigned char)0x55, 
(unsigned char)0xA0 },
   .flashProgramUnlockSeqLength = 3
   };

   pcbench_modelsDB[0] = &J9088A;
   pcbench_modelsDB[1] = &J4900A;
   pcbench_modelsDB[2] = NULL; //null stopper
}


pcbench_send_SMconfigSetup(char* cmd) {
   if (strcmp(cmd, pcbench_lastSMreadConfigCommand)) {
      term_sendCmd(cmd, NULL);
      strncpy(pcbench_lastSMreadConfigCommand, cmd, 
sizeof(pcbench_lastSMreadConfigCommand));
   }
}


int pcbench_nvfs_sanitizeNode(struct pcbench_nvfs_node* node) {
   struct pcbench_nvfs_node* node2;
   char msg[256];
   char *ghostStr="";

   if (!pcbench_nvfs_isActiveNode(node))
      ghostStr = "ghost";

   sprintf(msg, "Sanitizing: %snode at 0x%x: %s, %u bytes", ghostStr, node-
>_nodeAddress, node->fname, node->datasize);
   log_info(msg);
   time_t tStart = time(&tStart);
   time_t tEnd;
   double durSec;

   if (pcbench_nvfs_isSanitized(node)){
      //already santizied (as long as the local node is synced to the node on flash
)
      log_info(" Already sanitized");
      return 0;
   }

   unsigned long a = node->_nodeAddress + pcbench_nvfs_node_HEADERSIZE;

   for (unsigned int i = 0; i < node->datasize; i++) {
      if (pcbench_memWriteByteFlash(a+i, (unsigned char) 0x00)) {
         return -1;
      }
      printf("\r%u%% complete", i * 100 / node->datasize);
   }
   printf("\r100%% complete\nVerification... ");

   node2 = pcbench_nvfs_fetchNode(node->_nodeAddress); //refetch the (hopefully)
sanitized node
   //todo: free node and fields
   node = node2;
   if (pcbench_nvfs_isSanitized(node)) {
      printf(" OK\n");
      time(&tEnd);
      durSec = difftime(tEnd, tStart);
      sprintf(msg, "Sanitized Node at 0x%x took %5.1f seconds, %5.1fbytes/sec",
node->_nodeAddress, durSec, node->datasize/durSec);
      log_info(msg);

      return 0; //sucesss
}
   else{
      printf(" FAILED!!!");
      return 1; //failed
   }
}



int pcbench_nvfs_sanitizeAll(int sanitizeActivetoo) {
   unsigned long addr;
   struct pcbench_nvfs_node *node;

   addr = pcbench_nvfs_getFirstNodeAddr();
   node = pcbench_nvfs_fetchNode(addr);        //get the first .bootblock  node

   while (node->nextp != 0xffffffff) {
      node = pcbench_nvfs_fetchNode(node->nextp); // fetch next
      if (node == NULL) return -1;
      if (pcbench_nvfs_isActiveNode(node) && !sanitizeActivetoo) {
         continue;
      }

      if (pcbench_nvfs_sanitizeNode(node))
         return -1;
   }

   return 0;
}


int pcbench_nvfs_dumpAllToFile(char* file){
   FILE *f;
   char cmd[128];
   int len;
   char *str;
   long leftToRead = 0;
   long toRead;
   unsigned long addr;
   struct pcbench_nvfs_node *node;
   char *DELIM =
"============================================================================\n";


   //check if file already exists
   if (f = fopen(file, "r")) {
      fclose(f);
      log_msgn_NULLTERM(LOG_ERROR, "File already exists. Could not open ", file,
NULL);
      return -1;
   }

   if (!(f = fopen(file, "w"))) {
      log_msgn_NULLTERM(LOG_ERROR, "Could not open file with r/w access: ", file,
NULL);
      return -1;
   }

   addr = pcbench_nvfs_getFirstNodeAddr();
   node = pcbench_nvfs_fetchNode(addr);        //get the first .bootblock  node
   if (node == NULL) {
      log_error("pcbench_nvfs_dumpAllToFile(): Could not fetch boot block");
      return -1;
   }

   while(1){
      fprintf(f, DELIM);
      fprintf(f, "NODE INFO\n");
      fprintf(f, "Filename:           %s\n", node->fname);
      fprintf(f, "Address:            0x%08x\n", node->_nodeAddress);
      if (pcbench_nvfs_isActiveNode(node))
         str = "Yes";
```

```
     else
        str = "No";
     fprintf(f, "Is Active?:         %s\n", str);
     fprintf(f, "Next node address:  0x%08x\n", node->nextp);
     fprintf(f, "Date:               0x%02x 0x%02x 0x%02x 0x%02x\n", node->date[0],
node->date[1], node->date[2], node->date[3]);
     fprintf(f, "Active flags:       0x%02x 0x%02x\n", node->activeflag[0], node-
>activeflag[1]);
     fprintf(f, "Size [bytes]:       %u\n", node->datasize);
     fprintf(f, "Data, first 32bytes are header, next bytes are data:\n");


     //write the data portion
     leftToRead = node->datasize + 32; //header + data
     addr = node->_nodeAddress;

     while (leftToRead>0){

        /*Setup the switch  to read/display 256 or 16bytes (one row) hexdump style.
        It is practical since the nv filesystem is 16byte aligned.
        pcbench_nvfs_fetchNode modifie s this so it has to be called before
        each node data is dumped.
        */

        if (leftToRead >= 256) {
           toRead = 256;
           pcbench_send_SMconfigSetup("sm -l256 -ab -db");
        }
        else{
           toRead = 16;
           pcbench_send_SMconfigSetup("sm -l16 -ab -db");
        }


        if (sprintf(cmd, "read 0x%08x", addr) < 0) {
           log_msgn_NULLTERM(LOG_ERROR, "pcbench_nvfs_dumpAllToFile: Could not
generate read address string: ", cmd, NULL);
           return -1;
        }

        len = term_sendCmdGetRes(cmd, NULL, buf);
        buf[len] = '\0'; // null terminate
        if (len <= 0) {
           log_error("pcbench_nvfs_dumpAllToFile: read error");
           return -1;
        }

        if (len != fwrite(buf, sizeof(char), len, f)){
           log_error("pcbench_nvfs_dumpAllToFile: write error");
           return -1;
        }

        leftToRead -= toRead;
        addr += toRead;

     } //end for each hexdump line

     if (pcbench_nvfs_isActiveNode(node))
        log_msgn_NULLTERM(LOG_INFO, "Node successfully fetched: ", node->fname,
NULL);
     else
        log_msgn_NULLTERM(LOG_INFO, "Ghost Node successfully fetched: ", node-
>fname, NULL);


     if (pcbench_nvfs_isLastNode(node))
        break;  //break out of the for-each-node loop

     node = pcbench_nvfs_fetchNode(node->nextp); // fetch next node
     if (node == NULL) return -1;

  }// end for each node
     fprintf(f, DELIM);
     return 0;
}
```

Output listing 6-19: sanitty_pc.c

```
/*
* Command line application for inspecting and sanitizing HP Procurve flash.
* Especially the NV filesystem storing configurations.
* Author and copyright Magnus Larsson, magnus [at] stril.com
* Written as part of master thesis project 2015 in Electrical Engineering from KTH.
* This code is HIGHLY experimental and not very robust in terms of input checks.
* No warranties given.
*/



#include <stdio.h>
#include <stdlib.h> //atoi, atexit
#include <string.h> //strchr
#include "log.h"
#include "term.h"
#include "rs232.h"
#include "pcbench.h"

struct option {
   char *name;
   char *strValue;
};

void static printHelp();
void static processOptions(int argc, char *argv[]);
struct option parseOption(char* str);
static void processCommand();
void exitCleanup(void);
void tests();
void test_rx();


//optionVariables settable form the command line
int comPortOpt = 0;
int maxSpeedOpt=115200;
char modeOpt[] = { '8', 'N', '1', 0 };
FILE* termlogFile;


static char** cmd;    //commands to execute
unsigned char buf[term_READ_BUFFER_SIZE];


int main(int argc, char *argv[])
{

   atexit(exitCleanup); //register the cleanup routine
   log_setFilterLevel(LOG_DEBUG);
   log_debug("Sanitty started");

   processOptions(argc, argv);
   //handle unknown model error
   if (!pcbench_getCurrentModel()) {
      //display which options are supported.

      printf("ERROR: Missing mandatory option: -device = { part# } \nThese part# are
currently supported:\n");
      struct pcbench_model_info **modDB = pcbench_getModelDB();
      int i = 0;
      while (modDB[i]){
         printf(modDB[i]->partnr);
         printf("\n");
         i++;
      }
      printHelp();
      exit(EXIT_FAILURE);
   }

   /*
   * It seems optimal lineDelim should be \r on commamnds sent to the switch and
\r\n for data returned.
   * But term_xxx doesn't make difference of the rx and tx new line strings so we
will go for the deafult for now.
   */
   //term_setLineDelim("\r");
   term_setTimeout(10000);

   if(term_open(comPortOpt, maxSpeedOpt, modeOpt))
   {
      log_error("Can not open comport");
      exit(EXIT_FAILURE);
   }

   if (pcbench_enterBenchMode(maxSpeedOpt)) {
      log_error("Could not enter bench mode on the switch.");
      exit(EXIT_FAILURE);
      }


// tests();
   processCommand();

   printf("Press [Enter] to exit . . .");
   fflush(stdout);
   getchar();
   return 0;

}


void printHelp() {
   printf("Usage: sanitty_pc --option1=value --option2 command \n\n"
      "Options: (may be prepended by - or --) \n"
```

```
     " -loglevel= {debug, info, error} default:info\n"
     " -port= { 0, 1, 2 ... } com port to use, default: 0\n" //windows
     " -maxspeed= {110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 
115200} default: 115200\n"
     " -termlog=[path to RS232 input log file]\n"
     "Mandatory option:\n"
     " -device={part#} example -device=J9088A\n"
     "\nCommands:\n"
     " sanitize              Writes 0x00 to all unactive nv file records\n"
     " dumpflash [filename]  Dumps the contents of the flash chip\n"
     " dumpnvfs [filename]   Writes a list of the nv file records with contents to 
a file\n"
     "\n***\n"
     "IMPORTANT: This tool is highly experimental and may make your switch unusable 
and invalidate your warranty.\n"
     "It is supplied AS IS without any warranties.  Copyright Magnus Larsson 2015. 
magnus"
     "@stril.com\n"
     "***\n"
     "\n");

}


void processOptions(int argc, char *argv[]){


   ////debug
   //for (int i = 0; i < argc; i++)
   //log_debug(argv[i]);

   for (int i = 1; i < argc; i++) {
      struct option opt;
      char* str;

      opt = parseOption(argv[i]);


      if (!opt.name) {
         //end of options. the rest are commands.
         cmd = &argv[i];
         break;
      }

      str = "loglevel";
      if (!strcmp(str, opt.name)) {
         if (!strcmp(opt.strValue, "debug"))
            log_setFilterLevel(LOG_DEBUG);
         else if (!strcmp(opt.strValue, "info"))
            log_setFilterLevel(LOG_INFO);
         else if (!strcmp(opt.strValue, "error"))
            log_setFilterLevel(LOG_ERROR);
         else {
            log_msg2(LOG_ERROR, "Unknown loglevel option value: ", opt.strValue);
            printHelp();
            exit(EXIT_FAILURE);
         }
         continue;
      }

      str = "maxspeed";
      if (!strcmp(str, opt.name)) {
         maxSpeedOpt = atoi(opt.strValue);
         continue;
      }

      str = "port";
      if (!strcmp(str, opt.name)) {
         comPortOpt = atoi(opt.strValue);
         continue;
      }

      //this will be a cmd line option until a safe way of autodetec from the CLI is 
found.
      str = "device";
      if (!strcmp(str, opt.name)) {
         //a device option is passed. Try to get its infoDB
         struct pcbench_model_info* modelInfo = pcbench_getModelInfo(opt.strValue);
         if (modelInfo) {
            pcbench_setCurrentModel(modelInfo);

         }
      }



      str = "termlog";
      if (!strcmp(str, opt.name)) {
         //check if file already exists
         FILE *f;
         if (f=fopen(opt.strValue, "r")) {
            fclose(f);
            log_msgn_NULLTERM(LOG_ERROR, "Could not open termLog. File already 
exists: ", opt.strValue, NULL);
            exit(EXIT_FAILURE);
         }

         if (! (f = fopen(opt.strValue, "w"))) {
               log_msgn_NULLTERM(LOG_ERROR, "Could not open term with r/w access: ", 
opt.strValue, NULL);
               exit(EXIT_FAILURE);
            }

               log_msgn_NULLTERM(LOG_INFO, "termlog opened: ", opt.strValue, NULL);
               termlogFile = f;
               term_setTermLog(f);

      } // option inlog
```

```
   }  // for each option

}


// ================================================
//@{
// Parses an option. Eg -param=value returns a struct of {param, value}
//
// @param str Option string to parse
// @returns optionstruct on success. Option struct with NULL fields on error.
//@}
// ================================================

struct option parseOption(char* str) {
   log_msgn_NULLTERM(LOG_DEBUG, "parsing command: ", str, NULL);


   struct option opt = { 0, 0 };

   if (str[0] != '-') {
      return opt; //not an option, return opt with null fields
   }
   str++;
   if (str[0] == '-')
      str++; //skip second -

   opt.name = str;
   if (opt.strValue = strchr(str, '=')) {
      opt.strValue[0] = 0;
      opt.strValue++;
   }

   log_msgn_NULLTERM(LOG_DEBUG, "Option ", opt.name, " found, value: ",
opt.strValue, NULL);

   return opt;
}


static void processCommand(){
   char *str;
   char* filename;

   str = "sanitize";
   if (!strcmp(str, cmd[0])) {
      int sanitizeActivetoo = 0;
      if (pcbench_nvfs_sanitizeAll(sanitizeActivetoo)) {
         log_error("Failed to sanitize NV file system.");
         exit(EXIT_FAILURE);
      }
      else {
         log_info("Successfully sanitized NV file system.");
         return;
      }
   }

   str = "dumpflash";
   if (!strcmp(str, cmd[0])) {
      filename = cmd[1];
      if (!filename){
         log_msgn_NULLTERM(LOG_ERROR, "Output file missing", filename, NULL);
         exit(EXIT_FAILURE);
      }
      log_msgn_NULLTERM(LOG_INFO, "Dumping flash to file: ", filename, NULL);
      if (pcbench_flashDumpToFile(filename)) {
         log_msgn_NULLTERM(LOG_ERROR, "\nFailed to dump flash to file: ", filename,
NULL);
         exit(EXIT_FAILURE);
      }
      else {
         log_msgn_NULLTERM(LOG_INFO, "\nSuccessfully dumped flash to file: ",
filename, NULL);
         return;
      }
   }

   str = "dumpnvfs";
   if (!strcmp(str, cmd[0])) {
      filename = cmd[1];
      if (!filename){
         log_msgn_NULLTERM(LOG_ERROR, "Output file missing", filename, NULL);
         exit(EXIT_FAILURE);
      }
      log_msgn_NULLTERM(LOG_INFO, "Writing NV fs contents to file: ", filename,
NULL);
      if (pcbench_nvfs_dumpAllToFile(filename)) {
         log_msgn_NULLTERM(LOG_ERROR, "Failed to write NV fs contents to file: ",
filename, NULL);
         exit(EXIT_FAILURE);
      }
      else {
         log_msgn_NULLTERM(LOG_INFO, "Successfully wrote NV fs contents to file: ",
filename, NULL);
         return;
      }
   }

   if (cmd[0])
      log_msgn_NULLTERM(LOG_ERROR, "Unknown command:", cmd[0], NULL);
   else
      log_msgn_NULLTERM(LOG_ERROR, "Command missing.", NULL);

   printHelp();


}
```

```
void exitCleanup(void) {
   //close any open file descriptors
   if (termlogFile)
      fclose(termlogFile);

}

void tests() {
   //some test cases
   log_msgn_NULLTERM(LOG_ERROR, "!!!!!!!!!!!!!!!!!!!TESTS!!!!!!!!!!!!!!!!!!!!",
NULL);
   //test a write
   unsigned long addr;

   addr = 0x7cf20ee4;
   pcbench_memReadBytes(addr, 4, buf);
   pcbench_memWriteByteFlash(addr, 0x00);

   pcbench_nvfs_dumpAllToFile("c:\\tmp\\nvfsdump.txt");
   pcbench_flashDumpToFile("c:\\tmp\\flash.bin");
   pcbench_nvfs_sanitizeAll(0);

   addr=pcbench_nvfs_getFirstNodeAddr();
   struct pcbench_nvfs_node *node = pcbench_nvfs_fetchNode(addr);



   int read =  pcbench_memReadBytes(0xbcee0000, 1000000, buf);
}
```

Output listing 6-20: log.h

```
/*
* Contains logging utility system.
* Author and copyright Magnus Larsson, magnus [at] stril.com
* Written as part of master thesis project 2015 in Electrical Engineering from KTH.
* This code is HIGHLY experimental and not very robust in terms of input checks.
* No warranties given.
*/


#ifndef LOG_H
#define LOG_H

#include <stdio.h>  // FILE descriptor

//Severity levels
#define LOG_DEBUG 1
#define LOG_INFO  2
#define LOG_ERROR 3


//Set the level at which messages are currently displayed. Less severe messages are
discarded.
void log_setFilterLevel(int level);

void log_msg(int severity, char* message);

void log_msg2(int severity, char* m1, char* m2);
void log_msg3(int severity, char* m1, char* m2, char* m3);
void log_msgarray(int severity, char* msgArr[]);

void log_msgn_NULLTERM(int severity, ...);


//log with LOG_INFO severity
void log_info(char* message);

//log with LOG_debug severity
void log_debug(char* message);

//log with LOG_error severity
void log_error(char* message);

// Filedescriptor to write log entries to. Default is 2 ( stderr )
void setOutputFile(FILE* fd);

#endif /* LOG_H */
```

Output listing 6-21: log.c

```
/*
* Contains logging utility system.
* Author and copyright Magnus Larsson, magnus [at] stril.com
* Written as part of master thesis project 2015 in Electrical Engineering from KTH.
* This code is HIGHLY experimental and not very robust in terms of input checks.
* No warranties given.
*/


#include "log.h"
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h> //for log_msgn variable arguments
#include <assert.h>
#include <stdlib.h> //malloc

//file desriptor to write to. stderr by default.
static FILE *file;

//Level at which messages are currently displayed. Less severe messages are
discarded.
static int filterLevel = LOG_INFO;

//include time
static int includeTime = 1;

void log_msgn_NULLTERM(int severity, ...) {

   int len;
   va_list ap;
   char* msg;
   char* str;

   va_start(ap, severity);

   msg = malloc(1024);
   assert(msg);
   msg[0] = '\0'; //empty string

   str = va_arg(ap, char*);

   while (str ) {
   if (str[0] == '\n')
      break;
   msg = realloc(msg, strlen(msg) + strlen(str) + 1);
   assert(msg);
   sprintf(msg, "%s%s", msg, str); //concatenate
   str = va_arg(ap, char*);

   }
   va_end(ap);
   log_msg(severity, msg);
}



void log_msg(int severity, char* message){
   if (severity < filterLevel)
      return;

   FILE *f=file;
   if (!f)
      f = stderr;  //log to stderr by default

   char* sevPrefix="?";
   switch (severity) {
   case LOG_DEBUG: sevPrefix = "Debug:"; break;
   case LOG_INFO: sevPrefix = "Info: "; break;
   case LOG_ERROR: sevPrefix = "Error:"; break;
   }

   fprintf(f, "%s ", sevPrefix);


   if (includeTime) {
      time_t now;
      char* strTime;

      time(&now);
      strTime = ctime(&now);  // www dd hh:mm:ss yyyy
      //just select the  hh:mm:ss
      strTime += 11;
      strTime[8] = 0;

      fprintf(f, "%s ", strTime);
   }

   fprintf(f, "%s\n", message);

}


void log_msg2(int severity, char* m1, char* m2){
   char* str;
   str = malloc(strlen(m1) + strlen(m2) + 1);
   assert(str);
   sprintf(str, "%s%s", m1, m2); //concatenate
   log_msg(severity, str);
   free(str);
}

void log_msg3(int severity, char* m1, char* m2, char* m3){
   char* str;
   str = malloc(strlen(m1) + strlen(m2) + strlen(m3) + 1);
   assert(str);
   sprintf(str, "%s%s%s", m1, m2, m3); //concatenate
   log_msg(severity, str);
   free(str);
}

void log_msgarray(int severity, char* msgArr[]){
//	char* str;
   //int len=0;
 //
   //str = malloc(strlen(m1) + strlen(m2) + strlen(m3) + 1);
   //assert(str);
   //sprintf(str, "%s%s%s", m1, m2, m3); //concatenate
   //log_msg(severity, str);
   //free(str);


}

//log with LOG_DEBUG severity
void log_debug(char* message) {
   log_msg(LOG_DEBUG, message);
}

//log with LOG_INFO severity
void log_info(char* message) {
   log_msg(LOG_INFO, message);
}

//log with LOG_ERROR severity
void log_error(char* message) {
   log_msg(LOG_ERROR, message);
}

// Filedescriptor to write log entries to. Default is 2 ( stderr )
void setOutputFile(FILE* fd) {
   file = fd;
}

//Set the new filter level
void log_setFilterLevel(int newlevel){
   filterLevel = newlevel;
}

//Set includeTime, 0=disable, 1=enable
void setIncludeTime(int b){
   includeTime = b;
}
```

Appendix I. Independence of data when searching for markers

Does the probability of finding a random marker in a data string by accident depend on how the data looks? In the previous Section 3.2 I stated: *“A random marker of a single byte has the probability 2^{-8} to match any other byte regardless of probability distribution”*

Is this really true? Should not the probability of finding a marker by accident depend on the data it is injected into? In this section we will see that if the markers are created from *random* symbols, it is possible to calculate an upper limit for the risk of finding them in any data, regardless what the data may look like.

Let us study this in steps by looking at a generic single data and marker symbol and investigate the probability of them matching.

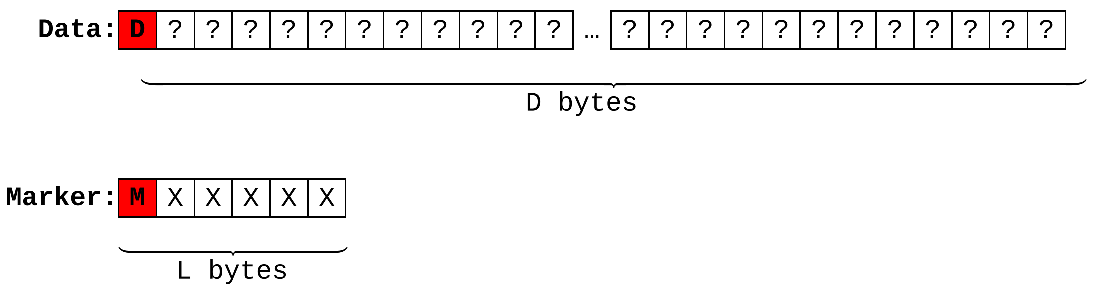


Figure 6-2: Data and ma

Let **D** be a symbol in the data string having any of the **d** symbol values from the alphabet $\{d_1, d_2, \ldots, d_d\}$.

Let **M** be a symbol in the marker string having any of the **m** symbol values from the alphabet $\{m_1, m_2, \ldots, m_m\}$

Let M’s alphabet be a subset of D’s alphabet. Or differently said: Every symbol value in M’s alphabet exists in D’s alphabet. But a symbol value of D’s alphabet may not necessarily exist in M’s alphabet.

Example of the above notation: character marker in full byte data

D can be any of the 255 values of byte. d=255.

M can be any of the 52 values of the byte which are characters. m=52

Every character M can have will also be possible to find in D

But every value D can have will not be in M’s alphabet (i.e. the non characters)

D and M can be considered as discrete random (stochastic) variables. Since we generate each marker’s symbol value random between the values in its alphabet we can write the probability distribution as:

Equation 6-1

$$Pr(M = d_x) = \begin{cases} \dfrac{1}{m}\ when\ \ d_x\ is\ a\ character\ in\ the\ markers\ alphabet\ (having\ length\ m). \\ 0\ otherwise \end{cases}$$

We are now interested to write the probability of a symbol match between M and D:

$Probablity(M = D)$

We can build a complete probability matrix for all combinations of M and D and their probability “mass”. It will be a square [d x d] matrix and the diagonal is the interesting part as those cells represent probability “mass” of a match. Example: The first cell contains the probability of both M and D to both have the value d_1 and is Prob(M= d_1 & D= d_1).

Figure 6-3: Full state table over the probabilities of a marker and data symbol match

	$M=d_1$	$M=d_2$	…	$M=d_d$
$D=d_1$	**$p(M=d_1$ & $D=d_1)$**			
$D=d_2$		**$p(M=d_2$ & $D=d_2)$**		
…			**…**	
$D=d_d$				**$p(M=d_d$ & $D=d_d)$**

Since our marker symbols are generated random and thus are independent from the data symbols we can write the probability as the product of the separate independent probabilities [68p. 413] = Prob($M=d_x$ & $D=d_x$)=Prob($M= d_x$) * Prob($D= d_x$)

We can now express the probability-mass in the diagonal as:

Equation 6-2

$$Probablity\ of\ match\ between\ symbol\ M\ and\ D = Pr(M=D) = \sum_{x=1}^{d} Pr(M=d_x) * Pr(D=d_x)$$

The first factor in the above equation, $Pr(M = d_x)$, is the probability distribution of our marker symbols which we know (Equation 6-1). As it is $1/m$ for all d_x belonging to the alphabet of M and zero otherwise we only need to perform the sum over the alphabet of M:

$$Pr(M=D) = \sum_{x=1}^{m} Pr(M=d_x) * Pr(D=d_x) \quad = \frac{1}{m}\sum_{x=1}^{m} Pr(D=d_x) \quad =$$

The probability distribution of the data symbols is unknown but we know that if the marker and symbol alphabets are equal (m = d , such as for MAC address markers) we are performing the sum over the whole probability space which is equal to 1:

Equation 6-3

$$\begin{array}{c} Probablity\ of\ a\ marker\ symbol\ (M)\ match\ a\ data\ symbol\ (D). \\ M\ is\ uniformly\ random\ over\ the\ value\ range\ of\ D. \\ d{=}the\ number\ of\ different\ values\ M\ and\ D\ can\ take \end{array} \quad = \frac{1}{d}\underbrace{\sum_{x=1}^{d} Pr(D=d_x)}_{=1} = \boldsymbol{\frac{1}{d}}$$

If marker and data symbols do *not* share alphabet such as the example with character markers in generic byte data we can not calculate the sum *exactly* because we don't know the probability distribution of the data symbol values that happen to match the marker symbols. However we know that the sum over a *subset* of the probability mass function of D can never be more than 1. Therefore we can write:

Equation 6-4

$$\begin{array}{c} Probablity\ of\ marker\ (M)\ match\ in\ data\ (D) \\ Where\ M\ can\ take\ m\ values\ of\ D\ and\ the\ probablities \\ of\ these\ m\ values\ are\ uniform\ random\ in\ M. \end{array} \quad = \frac{1}{m}\underbrace{\sum_{x=1}^{m} Pr(D=m_x)}_{\leq 1} \leq \frac{1}{m}$$

This is good enough because we do not actually need to calculate the exact risk of a marker should appear by random in the data. All we need is an upper limit. We want to be able to state something like “the probability of finding this password string in unknown data by accident, is **LESS** than or equal to 0.001%”. If it is less, it is just good because it makes the marker stronger and the results more significant.

As such, we have shown that the ***upper limit*** of the probability of a marker symbol appearing in the data by chance ***is fully independent of the data we are looking in***. It only depends on the number of symbols a marker can have, as long as they are all uniformly distributed (= equally probable) and a subset of the data symbols.

Example: character marker in full byte data

Each marker symbol can be any of the 52 letter characters. If they are generated randomly uniform the probability of a match against any unknown byte in some data would be **less** than 1/52.

Appendix J. Discussion on rejection of ugly markers

Sections 3.2 and *Appendix I* showed how long a strong marker should be and that its minimum strength is independent of the data searched under the assumption that the marker is random. One question is whether certain markers should be eliminated. If a MAC address marker comes out as FF:FF:FF:FF:FF:FF from the random number generator it will match an completely erased flash. Intuitively it seems like a good idea to reject certain markers and generate a new marker. However, in practice it is not needed and it will cause the math surrounding the false positive calculations to be invalid.

Let us investigate this by an example. The probability of the number generator making an all FF_{16} or 00_{16} MAC address is 2^{-47} which is so unlikely that this will not propose a problem. According to Table 3-2 on page 24 the risk of accidental match of a MAC address in 1 GB of data is 0.00038%. If we are prepared to accept the risk of a false positive at 0.00038% it does not make sense to take precautions to avoid the *much* smaller risk of 2^{-47}. It is like traveling at 200km/hour on a small, icy mountain road with a motorcycle and worrying about being struck by lightning.

Also if we start to reject markers based on assumption of the data, there will be dependence between marker and data and the relation $Prob(M= d_1 \,\&\, D= d_1)=Prob(M= d_1\,) * Prob(D= d_1)$ is false and we cannot arrive at Equation 6-2. Thus we can safely trust random markers in *any* data as long as the markers are of sufficient length and have enough large alphabet (symbol value range).

Appendix K. Alternative: Maximum contrasting markers?

Section 3.2 proposed a method to generate random markers to be injected into the configurations. As they are uniformly random we can think of them as *white noise markers*. In section Appendix I I proved that we can quantify an upper limit of a false positive independent of the data we will search in.

An alternative marker generation method is to consider is the contrasting *marker*. The idea is to try and create markers which stand out as much as possible from the data it is to be injected in. For example, in an English text data with symbols a-z we could construct markers with the Swedish unique characters symbols å, ä, and ö. From a mathematical standpoint, I believe it is equivalent to trying and estimating the probability distribution of D and creating the M symbols so they differs as much as possible. For example, construct an "orthogonal" M which would minimize the sum in the expression below with notations taken from Section Appendix I:

$$Probablity\ of\ match\ between\ symbol\ M\ and\ D = \ Pr(M = D) = \sum_{x=1}^{d} Pr(M = d_x \ \&\ D = d_x)$$

At first thought this may sound easy, and intuitive: "In a world of green, markers should be red". And if we can minimize the sum above we would in fact also minimize the risk of a false positive for a given marker length.

However, there are many practical problems that make this method difficult compared to the white noise marker idea. I will address some of the difficulties I see:

- A scan of the data to estimate the probability distribution will only be an approximation. If the whole memory is pre-scanned and we find, let us says, not a single 0x13 byte. Can we then conclude the probability of that byte is 0 and consider it safe to use a single byte 0x61 (ASCII 'a') marker? I do not think so. Just because the (complex) system did not write that byte in the memory is no guarantee that it will not. For an error free estimation of the probability distribution we would need an infinite number of samples. Or complete understanding of the complex internal mechanisms generating the data that is written to the memory. That is not doable in practice.
- If we inject multiple markers in the same trials, then we are changing the distribution of the data which has to be considered.
- Since the individual symbols making up the marker are not independent, I do not think we can use the calculations in Section 3.2.
- Since the individual symbols making up the marker are constructed from and thus dependent on the data we cannot arrive at equation Appendix I because: Prob($M=d_x$ & $D=d_x$) ***is not equal to*** Prob($M= d_x$) * Prob($D= d_x$)
- If we do the data distribution pre-scan estimate, how long is it valid? Can we reboot?
- The config write and erase will change the data distribution. Should we also do a data distribution post-scan estimate? What if they differ by much (even if we remove the marker from the post scan to cancel its effect)? Should we use the post scan, pre-scan, or some average?
- If the post-scan estimation concludes the marker is no longer optimal, should we reject the test?
- Do we also need an estimation scan after the configuration with the marker saved, but before the erase?
- What if the pre-scan is hard to perform? For example, in the case where a chip is soldered and has to be removed for reading. Is a pre-scan estimation on a similar device enough or has it to be on the same device?

There are a lot of questions I cannot answer before I use this technique with confidence. Most importantly I have no clue as to how to quantify the risk of finding a contrasting marker by accident (e.g. a false positive). As I see it, the only advantage of the contrasting marker compared to the white noise marker would be the former *may* offer less probability of accidental occurrence in a data string. However, since I do not know how to calculate that risk I cannot use it. Moreover, if we are concerned about the risk we can increase the white noise marker length (at least for passwords, hostnames, or other variable length configuration strings).

I conclude that the possible small benefits of the contrasting marker are outweighed by the simplicity of the white noise marker. Hence, I will use white noise markers for my investigations.

TRITA-ICT-EX-2015:35