Backdooring your server through its BMC: the HPE iLO4 case

Fabien Périgaud, Alexandre Gazet & Joffrey Czarny
Rennes, June 13-15, 2018
Outline

Introduction

Previous works

Firmware security

A firmware backdoor

Conclusion
• Baseboard Management Controller (BMC) embedded in most of HP servers for more than 10 years.
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 applicative image, running as soon as the server is powered.
iLO is directly connected to the PCI-Express bus.
Source: Managing HP servers through firewalls with Insight Software¹

Server
Outline

Introduction

Previous works

Firmware security

A firmware backdoor

Conclusion
I iz in yur computer
stealing yur dataz
Methodology

- 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 modules (equivalent of processes)
- Analysis of the web administration interface
- Total time of the study, approximately 5 man-months

Publication and tooling

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

One critical vulnerability identified

- CVE-2017-12542, CVSSv3 9.8
- **Authentication bypass** and **remote code execution**
- Fixed in iLO 4 version 2.53 (buggy) and 2.54

Full server compromise

- Arbitrary code execution in the context of the web server
- iLO to host attack
Vulnerability located in the web server

- Handling of HTTP line by line
- Many uses of C string handling manipulation functions:
  - strstr()
  - strcmp()
  - sscanf()
- Handling strings in C is complex and error-prone
How to properly use `sscanf()`?

```c
else if (!strnicmp(request, http_header, "Content-length:", 0xFu))
{
    content_length = 0;
    sscanf(http_header, "%*s %d", &content_length);
    state_set_content_length(global_struct_, content_length);
}
else if (!strnicmp(request, http_header, "Authorization:", 0xEu))
{
    sscanf(http_header, "%*s %15s %16383s", method, encoded_credentials);
    handle_authorization_credentials(method, encoded_credentials);
}
else if (!strnicmp(request, http_header, "Connection:", 0xBu))
{
    sscanf(http_header, "%*s %s", https_connection->connection);
}
```
The vulnerability allows to overflow the connection buffer of an `https_connection` object.

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

The vulnerability allows to overflow the connection buffer of an https_connection object.

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

Double cheese!

- Overwriting the boolean localConnection: **bypass of the REST API authentication**
  
curl -H "Connection: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" :)


Buffer overflow

The vulnerability allows to overflow the connection buffer of an `https_connection` object.

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

**Double cheese!**

- Overwriting the boolean `localConnection`: **bypass of the REST API authentication**
  
  ```bash
curl -H "Connection: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA:" ;)
  ```

- Overwriting the vtable pointer: **arbitrary code execution**
  - No NX, no ASLR
  - Web server working buffer at a fixed address
Analysis of a module: CHIF (*Channel Interface*)

- Ability to read WHEA information from the host OS
- Direct (read) access to the host memory

Feature analysis

- 16MB of the host memory can be mapped into the iLO memory using an unknown PCI register
- Writing to this mapped memory also impact the host memory
- Re-implement this mechanism in a shellcode executed in the context of the iLO WWW server
Introduction

Previous works

Firmware security

A firmware backdoor

Conclusion
Battle plan

Current status

- Full platform compromise
- Arbitrary code execution on the iLO and the host
- RW primitives to the host memory from the iLO

Our objective

- Persistent compromise
- Survive host re-installation
- Stealthiness

Idea
iLO firmware backdooring
Update mechanisms:
- Dedicated interface from the web administration panel
- From the host, using a dedicated binary

Firmware updates are signed

Integrity checked at two distinct times:
- Dynamically, during the update process, by the currently running iLO
- At boot-time, no hardware root of trust though
Bypass of the update mechanism

- Modules can expose services
- These services can be instantiated as object

**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
At this point, a rogue firmware is written in the flash.
System boot-time

1. check integrity
2. decompress
3. load

ILO4 bootchain

HW reset

1. check integrity
2. decompress
3. load

bootloader

kernel

userland
The up-coming compromise

Methodology

- Full extraction of the firmware update
Methodology

- Full extraction of the firmware update
- Patch of the bootloader
The up-coming compromise

Methodology

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

1. check integrity
2. decompress
3. load

hardware reset

iLO4

userland

kernel

bootloader
The up-coming compromise

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
WWW server

- Frequently exposed
- High-level network/HTTP communication primitives
- Ability to access the host memory through DMA (demonstrated)
- Large binary
How to insert the backdoor?

The WWW server handles many pages, like:

- `/html/help.html`
- `/debug.html`
- `/html/info_blade.html`
- `/html/admin_manage.html`

Internally represented by structures; a dedicated pointer for each supported HTTP method (GET, POST, PUT, DELETE, HEAD).
How to insert the backdoor? (2)

- Insert code in an unused space of the WWW server binary
- Highjack pointers (GET et POST) from a page handler to point to our code
We want a bidirectional channel between the iLO and the Linux host, through the DMA link.
Code injection

- Overwrite the GET request handler
- Insert code in unused space of the binary: content of a downloadable PE file

Features

- R/W primitive in the host physical memory
- Re-use web server functions to parse/handle request
Linux kernel implant

Specifications

• Create a new kernel thread
• Allocate physical memory for the communication channel
• Retrieve and execute commands
• Retrieve commands output
Specifications

- Create a new kernel thread
- Allocate physical memory for the communication channel
- Retrieve and execute commands
- Retrieve commands output

Kernel API

- Create a new kernel thread: `kthread_create_on_node() / wake_up_process()`
- Physical memory allocation: `kmalloc() / virt_to_phys()`
- Run commands: `call_usermodehelper()`
- Retrieve their output: redirection into a temp file, then `kernel_read_file_from_path()`
Simple structure in a shared physical memory page

- Buffer to store shell command sent by the iLO
- Buffer to store the command output, later grabbed by the iLO
- Booleans to signal the availability of data

```c
struct channel {
    int available_input;
    int input_len;
    char input[4096];
    int available_output;
    int output_len;
    char output[];
}
```
Attacker side: client in Python

- Check for the presence of implants
- Installation and removal of the Linux implant
- Send arbitrary commands

Problem: received data are sometimes slightly corrupted
Root cause seems to be in the encoding of specific characters...
We need to patch this bug as well

```c
if ( v13 == '%' )
{
    if ( v11 < 2 || sscanf(v5, "%d", &v19) != 1 || v19 > 0xFF )
        return 0;
    v12 = v19;
    v5 += 2;
    v11 -= 2;
    goto LABEL_21;
}
```

# Patch query string decoding bug...
# "%d" => addrof("%02x")

```c
PATCH5 = {"offset": 0x5D534, "size": 4, "prev_data": "25640000", "patch": "A8CE0400", "decode": "hex"}
PATCHES.append(PATCH5)
# ADR R1, "%d" => LDR R1, addrof("%02x")
PATCH6 = {"offset": 0x5D1A4, "size": 4, "prev_data": "E21F8FE2", "patch": "88139FE5", "decode": "hex"}
PATCHES.append(PATCH6)
```
How to detect the compromise of an iLO host?

- Retrieve current firmware using a shellcode that reads the content of the flash memory
- Compare to a list of known “good” images
- [https://github.com/airbus-seclab/ilo4_toolbox](https://github.com/airbus-seclab/ilo4_toolbox)
- Smart kid: what about a backdoor that alters the read data on the fly?
Outline

Introduction

Previous works

Firmware security

A firmware backdoor

Conclusion
iLO4 key takeaways

- No hardware root of trust\(^2\), combined to the bypass of some of the integrity check mechanism: **persistence achievable and demonstrated**
- DMA access to the host memory re-purposed as a dual-way communication channel
- The proof-of-concepts require the exploitation of a vulnerability and execution of arbitrary code on the iLO system
- Vulnerability reported to the vendor and fixed (in May 2017), **please patch!**
- iLO4, critical remote administration tool:
  - Fully disabled if not actively used
  - Network isolation

Thanks for your 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