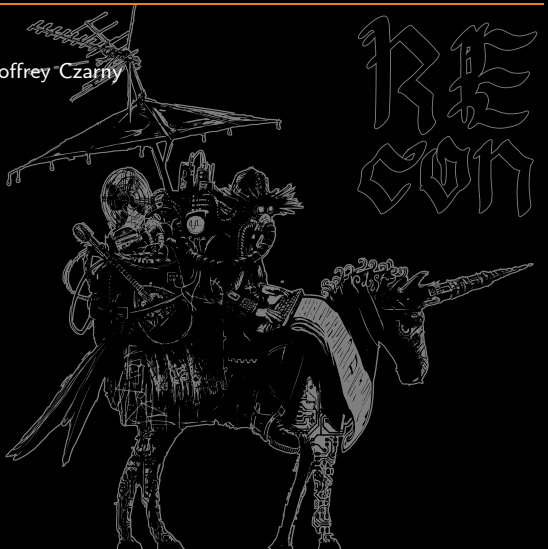
Subverting your server through its BMC: the HPE iLO4 case

Fabien Périgaud, Alexandre Gazet & Joffrey Czarny

Brussels, February 2-4, 2018



SYNACKTIV
DIGITAL SECURITY

AIRBUS

Outline

- Baseboard Management Controller (BMC) embedded in most of HP servers for more than 10 years
- Provides remote administration features:
  - Power Management
  - Remote system console
  - Remote CD/DVD image mounting
  - Several monitoring indicators

This talk will focus on iLO **version 4** (latest version until mid-2017) present in HP ProLiant Gen8 and Gen9.

Research has been more specifically performed on iLO4 versions **2.44** and **2.50**.

Figure 1: Directly integrated in the server's motherboard

Hardware level (2)

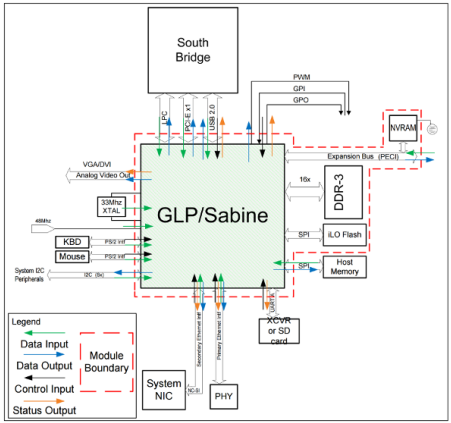
Standalone system:

- Dedicated ARM processor: `GLP/Sabine` architecture
- Firmware stored on a `NAND` flash chip
- Dedicated `RAM` chip
- Dedicated network interface

`iLO` runs even if the server is turned off.

There's a full operating system running in your server as soon as it has a connected power cord!

Privileged access to the server's hardware:



`iLO` is directly connected to the `PCI-Express` bus.

Summary of well known pentest tricks:

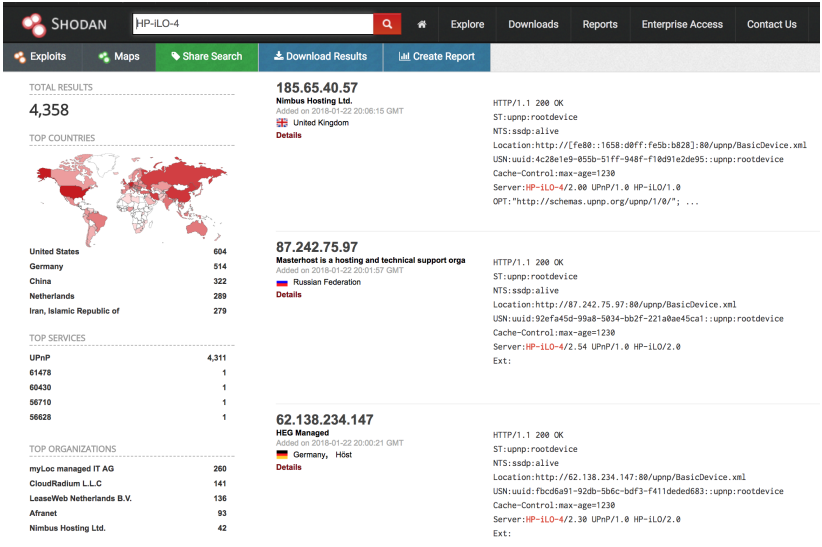
- `IPMI` Authentication Bypass via Cipher 0
- `IPMI 2.0 RAKP` Authentication Remote Password Hash Retrieval [1]

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/`

Internet presence of iLO4



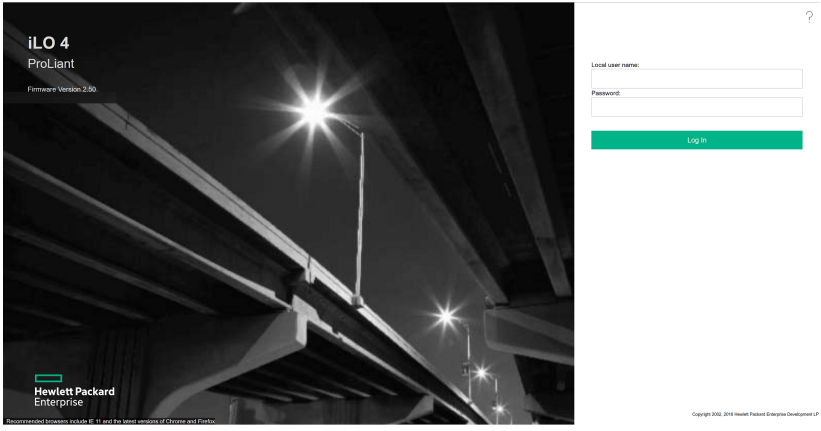
Around 3604 iLO interfaces version 4 exposed on Internet

```
   3 Server:HP-iLO-4/1.30 UPnP/1.0 HP-iLO/1.0
   1 Server:HP-iLO-4/1.51 UPnP/1.0 HP-iLO/1.0
 112 Server:HP-iLO-4/2.00 UPnP/1.0 HP-iLO/1.0
 140 Server:HP-iLO-4/2.02 UPnP/1.0 HP-iLO/1.0
 172 Server:HP-iLO-4/2.03 UPnP/1.0 HP-iLO/1.0
 230 Server:HP-iLO-4/2.10 UPnP/1.0 HP-iLO/2.0
 189 Server:HP-iLO-4/2.20 UPnP/1.0 HP-iLO/2.0
  29 Server:HP-iLO-4/2.22 UPnP/1.0 HP-iLO/2.0
 461 Server:HP-iLO-4/2.30 UPnP/1.0 HP-iLO/2.0
   4 Server:HP-iLO-4/2.31 UPnP/1.0 HP-iLO/2.0
 552 Server:HP-iLO-4/2.40 UPnP/1.0 HP-iLO/2.0
  14 Server:HP-iLO-4/2.42 UPnP/1.0 HP-iLO/2.0
 108 Server:HP-iLO-4/2.44 UPnP/1.0 HP-iLO/2.0
1050 Server:HP-iLO-4/2.50 UPnP/1.0 HP-iLO/2.0
 219 Server:HP-iLO-4/2.53 UPnP/1.0 HP-iLO/2.0
 320 Server:HP-iLO-4/2.54 UPnP/1.0 HP-iLO/2.0
```

Around 3788 iLO interfaces version 4 exposed on Internet

```
  86 Server:HP-iLO-4/2.00 UPnP/1.0 HP-iLO/1.0
 117 Server:HP-iLO-4/2.02 UPnP/1.0 HP-iLO/1.0
 144 Server:HP-iLO-4/2.03 UPnP/1.0 HP-iLO/1.0
 173 Server:HP-iLO-4/2.10 UPnP/1.0 HP-iLO/2.0
 169 Server:HP-iLO-4/2.20 UPnP/1.0 HP-iLO/2.0
  26 Server:HP-iLO-4/2.22 UPnP/1.0 HP-iLO/2.0
 297 Server:HP-iLO-4/2.30 UPnP/1.0 HP-iLO/2.0
   2 Server:HP-iLO-4/2.31 UPnP/1.0 HP-iLO/2.0
 422 Server:HP-iLO-4/2.40 UPnP/1.0 HP-iLO/2.0
   9 Server:HP-iLO-4/2.42 UPnP/1.0 HP-iLO/2.0
  83 Server:HP-iLO-4/2.44 UPnP/1.0 HP-iLO/2.0
1020 Server:HP-iLO-4/2.50 UPnP/1.0 HP-iLO/2.0
 193 Server:HP-iLO-4/2.53 UPnP/1.0 HP-iLO/2.0
 571 Server:HP-iLO-4/2.54 UPnP/1.0 HP-iLO/2.0
 474 Server:HP-iLO-4/2.55 UPnP/1.0 HP-iLO/2.0
```

WWW login portal



iLO summary

- Critical technology: remote management for HP servers
- Frequently encountered during pentest engagements
- Wildely exposed (Internet or LAN)
- Known weaknesses in the authentication protocol
- Too rarely monitored

Objectives

- Evaluate the trust we can put in a solution/product
- Better understanding of the technology **and its internals**
- Better understanding of the exposed surface/risk

One critical vulnerability identified

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

Full server compromise

- iLO to Host attack
- iLO firmware backdooring

Responsible disclosure time-line

- **Feb 2017** - Vulnerability discovered
- **Feb 27 2017** - Vulnerability reported to HP PSIRT by Airbus CERT
- **Feb 28 2017** - HP acknowledges receiving the report
- **May 5 2017** - HP releases iLO 4 2.53, silently fixing the vulnerability
- **July 20 2017** - Airbus CERT contacts MITRE to request a CVE ID
- **July 28 2017** - HP PSIRT tells Airbus CERT that they are planning to release a security bulletin
- **August 24 2017** - HP releases security bulletin HPESBHF03769[4]

[4] https://support.hpe.com/hpsc/doc/public/display?docId=hpesbhf03769en_us

Outline

Where to get HP iLO firmware ?

- Hewlett Packard Enterprise Support Center (obviously)
- Great repository of archived versions:
  http://pingtool.org/latest-hp-ilo-firmwares/

Decap the first layers of CP030133.exe (iLO 4 firmware v2.44):

- Self-extracting/script based archive:
- ```
  total 17M
  -rwxr-xr-x 1 user None 198K Jul 21  2016 CP030133.xml*
  -rwxr-xr-x 1 user None 490K Apr  1  2016 flash_ilo4*
  -rwxr-xr-x 1 user None  17M Jul 21  2016 ilo4_244.bin*
  -rwxr-xr-x 1 user None 9.9K Jul 21  2016 Readme.txt*
  ```
  ```

- `flash_ilo4`: flashing tool, *x86* code
- `ilo4_244.bin`: actual firmware, concatenation of:
  - *HP Signed File* header

```
--=</Begin HP Signed File Fingerprint\>=--
Fingerprint Length: 000527
Key: label_HPBBatch
Hash: sha256
Signature: WtLLCUv/ergBGLM6fULxgUUvffHNPNblf5KQFUYOBKxYznzepQggzhF/UsuU2zlrdOD
+KHOYNOOdkycgVDKjilkD1nCgPrfLOyjZLI22AONZOuEle3uW+Gvkj3s178Zt1RJizAYLXU/vAG47G
OR1MjKmB8ca5tzJKxuRi1AxtRcfU7DaVtHPTPZ7ro5QL+JH7/EeBIZbi79CsHTgOkVdiPNaVlQ1eYb
uKjLwHptuTmOAmpvPnZ6oQi8FDmtHSeEIY4nCBl7GwBTYMYVUMwDcI8HQypuwnaOdAeUy4z2/xYcIu
kbwlZNREDt4QPHZzCP52clJIRhtwsjdD2SUwj3jGA== Fingerprint Length: 000527
--=</End HP Signed File Fingerprint\>=--
```

  - 3 HP certificates
  - HPIMAGE blob

- Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

```
00000000  48 50 49 4D 41 47 45 00 01 01 00 00 9D 7B 31 2F  HPIMAGE......{1/
00000010  E3 C9 76 4D BF F6 B9 D0 D0 85 A9 52 01 00 00 00  ..............
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
00000030  00 00 00 00 00 00 00 00 00 00 00 00 E0 07 07 13  ............Ãã...
00000040  32 2E 34 34 00 00 00 00 00 00 00 00 00 00 00 00  2.44............
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
00000060  69 4C 4F 20 34 00 00 00 00 00 00 00 00 00 00 00  iLO 4...........
```

- Magic: HPIMAGE
- Size: 0x4B8
- Version and GUID, not that much interesting info
- Strip the header, size of mapped firmware: 0x1000000 bytes

- `Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F`

```
00FFFFC0  76 20 30 2E 31 2E 37 39 2B 20 32 35 2D 4A 75 6E  v 0.1.79+ 25-Jun
00FFFFD0  2D 32 30 31 35 00 FF FF FF FF FF FF FF FF FF FF  -2015..........
00FFFFE0  FF FF FF FF 00 00 01 00 00 00 00 00 00 00 00 00  ................
00FFFFF0  6B 09 7C 77 B3 00 00 2B BC FB 00 00 69 4C 4F 34  ...........iLO4
```

- "mirrored" blob header: `iLO4` magic at the end
- `0xFBBC`: negative offset from the end of the file (`0x444`)
- Points to the cryptographic parameters, size `0x444`

```
class SignatureParams(LittleEndianStructure):

    _fields_ = [
        ("sig_size", c_uint),
        ("modulus", c_byte * 0x200),
        ("exponent", c_byte * 0x200)
    ]
```

- Used later for image signature verification

- 0x10000 from the end of the file
- `Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F`

```
00FEFFE0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00FEFFF0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
00FF0000  09 00 00 EA 7C 03 00 EA E1 07 00 EA D5 03 00 EA  ................
00FF0010  E7 03 00 EA FE FF FF EA 66 03 00 EA 0A 04 00 EA  ................
00FF0020  7C 0E FF FF A8 02 FF FF 10 80 00 D0 68 07 00 EB  ................
```

- **This is ARM code!**
- ARM processor bootstrap code
- Load (and check integrity) image from the HPIMAGE blob

- Concatenated to the HPIMAGE header, a set of IMG_HEADER:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  69 4C 4F 34 20 76 20 32 2E 34 34 2E 37 20 31 39  iLO4 v 2.44.7 19
00000010  2D 4A 75 6C 2D 32 30 31 36 1A 00 FF FF FF FF FF  -Jul-2016.......
00000020  08 00 00 10 F8 0A 00 00 57 5F 10 00 E0 02 68 01  ................
00000030  D3 EA D0 00 FF FF FF FF 00 00 00 00 FF FF FF FF  ................
00000040  68 3C 5A 2A E9 DF A1 6A C2 D6 96 43 85 54 4E D0  ................
[...]
```

- iLO4 magic
- Images are signed (RSA sig)
- 3 images for this firmware: userland, kernel (*main*, *recovery*)
- Possibly compressed (LZ-*like* algorithm found in the bootstrap code)

```
class ImgHeader(LittleEndianStructure):

    _fields_ = [
        ("il0_magic", c_byte * 4),
        ("build_version", c_char * 0x1C),
        ("type", c_ushort),
        ("compression_type", c_ushort),
        ("field_24", c_uint),
        ("field_28", c_uint),
        ("decompressed_size", c_uint),
        ("raw_size", c_uint),
        ("load_address", c_uint),
        ("signature", c_byte * 0x200),
        ("padding", c_byte * 0x200)
    ]
```

Extracting image

- [+] iLO Header 0: iLO4 v 2.44.7 19-Jul-2016
  - > magic : iLO4
  - > build_version : v 2.44.7 19-Jul-2016
  - > type : 0x08
  - > compression_type : 0x1000
  - > field_24 : 0xaf8
  - > field_28 : 0x105f57
  - > decompressed_size : 0x16802e0
  - > raw_size : 0xd0ead3
  - > load_address : 0xffffffff
  - > field_38 : 0x0
  - > field_3C : 0xffffffff
  - > signature

```
0000  68 3c 5a 2a e9 df a1 6a c2 d6 96 43 85 54 4e d0   h<Z*...j...C.TN.
0010  c3 a4 e1 6f cb 2d 0f b6 0c 28 cd 31 88 db 07 6c   ...o.-...(.1...l
```

- Once extracted and decompressed: an **ELF**!

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00  .ELF............
00000010  02 00 28 00 01 00 00 00 00 00 00 00 34 00 00 00  ..(.........4...
00000020  A0 A2 67 01 00 0C 17 00 34 00 20 00 66 02 28 00  ........4. .f.(.
00000030  68 02 67 02 01 00 00 00 F4 4C 00 00 00 00 00 00  h.g.....L......
```

- Understanding of the `HPIMAGE` firmware format
- `HPIMAGE` blob is signed, verified by the `x86` code (flashing tool)
- Collection of images
- Each of them is signed, verified by the `ARM` bootstrap code
- The main image is a 23Mo `ELF` file!

Outline

ELF image

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          23569056 (bytes into file)
  Flags:                             0x170c00, GNU EABI, VFP, Maverick FP, <unknown>
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         614
  Size of section headers:           40 (bytes)
  Number of section headers:         616
  Section header string table index: 615
```

Large number of sections

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] MemRegion0        NOBITS          00000000 004cf4 036000 00 WAX  0   0  0
  [ 2] MemRegion1        NOBITS          00036000 004cf4 013000 00 WAX  0   0  0
  ...
  [49] .dvi.elf.text     PROGBITS        009a3000 004cf4 035468 00  AX  0   0  0
  [50] .dvi.elf.data     PROGBITS        009d9000 03a15c 00077c 00 WAX  0   0  0
  [51] .libINTEGRITY.so. PROGBITS        009ea000 03a8d8 0047ec 00  AX  0   0  0
  [52] .libINTEGRITY.so. PROGBITS        009ef000 03f0c4 000014 00 WAX  0   0  0
  [53] .libc.so.text     PROGBITS        009f0000 03f0d8 033b84 00  AX  0   0  0
  [54] .libc.so.data     PROGBITS        00a24000 072c5c 0007fc 00 WAX  0   0  0
  [55] .libc.so.bss      NOBITS          00a25000 073458 002000 00 WAX  0   0  0
  [56] .libevlog.so.text PROGBITS        00a27000 073458 03c024 00  AX  0   0  0
  [57] .libevlog.so.data PROGBITS        00a64000 0af47c 0022f4 00 WAX  0   0  0
  ...
  [613] .boottable       PROGBITS        03da5000 15ed530 0841d4 00  WA  0   0  0
  [614] .secinfo         PROGBITS        03e291d4 1671704 0060f0 00   A  0   0  0
```

Lots of build paths, root:

```
/home/delphyne2/autobuild/intbuild112210-000317-47/
```

File names:

```
rtos-i10.0/build/binary/rtos/modules/ghs/posix/pthread_create.c
rtos-i10.0/build/binary/rtos/modules/ghs/posix/posix_server.c
rtos-i10.0/build/binary/rtos/modules/ghs/posix/ptls.c
rtos-i10.0/build/binary/rtos/modules/ghs/posix/posix_env.c
rtos-i10.0/build/binary/rtos/modules/ghs/posix/ghsposix.c
rtos-i10.0/build/binary/rtos/modules/ghs/posix/pthread_sched.c
rtos-i10.0/build/binary/rtos/INTEGRITY-libs/util/getopt.c
```

Integrity

The ELF binary is an applicative image (userland) packaged by **HP**, running on Integrity real-time OS, developped by **Green Hills Software**[5]

[5] http://www.ghs.com/products/rtos/integrity.html

- Two ELF sections are critical to understand the memory layout: `.secinfo` & `.boottable`.
- `.secinfo`: linked list of `SECTION_INFO` structures:

```
struct SECTION_INFO
{
  SECTION_INFO *next;
  void *name;
  int section_offset;
  int section_size;
  int rights;
  int field_14;
};
```

```
-----------------[ Sections List ]-----------------

> name: .secinfo, 0x3e291d4, 0x60f0 bytes
> 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
> 0x0007 -      .libevlog.so.text at 0x00a27000, size 0x0003c024 flags 0x1,0x0
> 0x0008 -      .libevlog.so.data at 0x00a64000, size 0x000022f4 flags 0x9,0x0
> 0x0009 -      .libevlog.so.data at 0x00a67000, size 0x0000a000 flags 0xc,0x0
[...]
```

`section_offset` is an offset into the main ELF file. `rights` can take three values:

- `0x1` for executable section
- `0x9` for statically defined data
- `0xC` for uninitialized data (`stack`, `heap`, `bss`)

Parsing the `.boottable` section

The `.boottable` section:

- Is more complex
- Relies upon the `.secinfo` section
- Contains all the information needed to rebuild the virtual memory mapping of the `Integrity` image.

```
.boottable:03DA5000                     AREA .boottable, DATA, ALIGN=0
.boottable:03DA5000 dword_3DA5000       DCD 0x31
.boottable:03DA5004                     DCD TASK_CONF
.boottable:03DA5008                     DCD 0x30
.boottable:03DA500C                     DCD LIST_INITITIAL
.boottable:03DA5010                     DCD 0xFFFFFFFF
.boottable:03DA5014                     DCD 1
.boottable:03DA5018                     DCD 0
.boottable:03DA501C                     DCD 0
.boottable:03DA5020                     DCD 0
.boottable:03DA5024                     DCD LIST_INITITIAL
.boottable:03DA5028                     DCD 0
.boottable:03DA502C                     DCD 6
.boottable:03DA5030                     DCD 6
.boottable:03DA5034                     DCD LIST_CORE
```

List of shared modules (CORE_ENTRY):

```
struct CORE_ENTRY
{
  int name;
  int mod_size;
  int mod_index;
  int field_C;
  int field_10;
};
```

```
  -----------------[ Shared modules ]-----------------

> mod 0x00 -   libINTEGRITY.so size 0x000047ec, id 0x001e, flags 0xffffffff,0xffffffff
> mod 0x01 -           libc.so size 0x00033b84, id 0x0020, flags 0xffffffff,0xffffffff
> mod 0x02 -       libevlog.so size 0x00032cf0, id 0x0023, flags 0xffffffff,0xffffffff
> mod 0x03 - VComCShared_RM.so size 0x00005ba8, id 0x0033, flags 0xffffffff,0xffffffff
> mod 0x04 -        libsslc.so size 0x0009d010, id 0x0070, flags 0xffffffff,0xffffffff
```

List of task descriptors (TASK_INFO):

```
struct TASK_INFO
{
  int type;
  int vmem_space_count;
  void *vmem_space_array;
  int field_C;
  int var_count;
  void *var_array;
  int field_18;
  int field_1C;
  int field_20;
  int field_24;
  int field_28;
  int field_2C;
  int field_30;
  int field_34;
  int field_38;
  int field_3C;
};
```

The type field seems to take two values: 0x0 for the kernel task, 0x21 otherwise.

vmem_space_array: a pointer to an array of MEM_RANGE descriptors, used to fully describe the virtual memory space of the task:

```
struct MEM_RANGE
{
  int present;
  int rights;
  int range_base;
  int range_size;
  int id;
  int field_14;
  int field_18;
  int field_1C;
};
```

- present: 1 if the range is loaded from the content of a section from the ELF image (see the id field)
- rights: 5 for executable range, 7 for data (no distinction between .data, .stack or .heap).
- id: **shifted index of the ELF section**. The shift is equal to the number of modules.

Full memory mapping

```
> task 0x27 (webserv.elf) - 0x00000049 entries
  range: dw1 0x00 - dw2 0x007 - base 0x0001000 - size 0x00F000 - id 0xffffffff
  range: dw1 0x01 - dw2 0x005 - base 0x0010000 - size 0x2B6000 - id 0x000001e2 - .webserv.elf.text
  range: dw1 0x00 - dw2 0x007 - base 0x02c6000 - size 0x002000 - id 0xffffffff
  range: dw1 0x01 - dw2 0x007 - base 0x02c8000 - size 0x112000 - id 0x000001e3 - .webserv.elf.data
  range: dw1 0x01 - dw2 0x007 - base 0x03db000 - size 0x002000 - id 0x000001ef -
.webserv.Initial.stack
  range: dw1 0x01 - dw2 0x007 - base 0x03dd000 - size 0x190000 - id 0x000001f0 - .webserv.heap
  range: dw1 0x00 - dw2 0x107 - base 0x056d000 - size 0x004000 - id 0xffffffff
  range: dw1 0x00 - dw2 0x007 - base 0x0571000 - size 0x98F000 - id 0xffffffff
  range: dw1 0x00 - dw2 0x007 - base 0x0f00000 - size 0x100000 - id 0xffffffff
  range: dw1 0x00 - dw2 0x007 - base 0x1000000 - size 0x680000 - id 0xffffffff
  range: dw1 0x01 - dw2 0x005 - base 0x1680000 - size 0x033000 - id 0x000001ab - .libscxx.so.text
  range: dw1 0x00 - dw2 0x007 - base 0x16b3000 - size 0x001000 - id 0xffffffff
  range: dw1 0x01 - dw2 0x007 - base 0x16b4000 - size 0x001000 - id 0x000001e5 - .libscxx.so.data
  range: dw1 0x00 - dw2 0x007 - base 0x16b5000 - size 0x003000 - id 0xffffffff
  range: dw1 0x01 - dw2 0x007 - base 0x16b8000 - size 0x001000 - id 0x000001e6 - .libscxx.so.bss
  [...]
```

- Extensive understanding of the `Integrity` applicative image format
- Tasks (or processes) are isolated into their own virtual memory space
- Their base address is always `0x10000` (main *.text*)
- Automatization: pack all this stuff into a script that generates a complete `IDB`
- Let's check the kernel!

Outline

- No technical documentation available to us
- Only commercial information:

Figure 2: `https://www.ghs.com/products/rtos/integrity.html`

- All the information presented further is based on our analysis; it may be wrong/incomplete.

Micro-kernel architecture build upon a small set of core modules:

```
ROM:200BCBF0 BOOT_MODULES    BOOT_MOD <0x200BCC58, 0x200BCC7C, 0x200C0E6C,
                                       0x200BFC6C, 0x200C10C4, 0x200C1264,
                                       0x200C1288, 0x200C12AC, 0>
```

Each entry is a pointer to a BOOT_MOD structure:

```
struct BOOT_MOD
{
  void *constructor;
  void *destructor;
  char *name;
  void *dependencies;
  MOD_STATUS mod_status;
};
```

Parsing loader's components list:

```
> component "Resource Manager", constructor 0x20009aa4, destructor 0x00000000

> component "Dynamic Loader", constructor 0x20009bd8, destructor 0x20009c94
   > depedencies (0x200bcca0):
        Resource Manager (0x200b70fc)

> component "TCP/IP", constructor 0x200448bc, destructor 0x00000000
   > depedencies (0x200c0e90):
        Resource Manager (0x200b8a38)
        System Resources (0x200b8a4c)
        UMAC (0x200b8a60)

> component "UMAC", constructor 0x20030854, destructor 0x00000000
   > depedencies (0x200c0dd4):
        Resource Manager (0x200b7dec)
        System Resources (0x200b7e00)
[...]
```

They all depends upon the "`Resource Manager`" component!

Example of object instantiation:

```
1 rsrc_mgr_reglink_msg(OBJ_LoaderConnection,
2                      "__LoaderConnection",
3                      "!systempassword")
```

The code above setup and sends a request made of:

- A verb: "`reglink`"
- The name of the object: "`__LoaderConnection`"
- A password: "`!systempassword`"

List of supported verbs:

- "`procure`"
- "`delete`"
- "`register`"
- "`reglink`"
- "`procure-delete`"

There is an equivalent of the ELF `.secinfo` section in the kernel:

```
> parsing .secinfo entries:

       .romstart - base 0xf8000000 - size 0x00000000 - access : 0x4
         .romend - base 0xf8000000 - size 0x00000000 - access : 0x4
      .ramstart - base 0x20001000 - size 0x00000000 - access : 0x4
       .picbase - base 0x20001000 - size 0x00000000 - access : 0x4
        .vector - base 0x20001000 - size 0x000000a8 - access : 0x1
   .probe_agent - base 0x200010a8 - size 0x00000038 - access : 0xc
      .fasttext - base 0x20004000 - size 0x00001f6c - access : 0x9
        .rtext - base 0x20005f6c - size 0x00000000 - access : 0x4
         .text - base 0x20005f6c - size 0x000b0a9c - access : 0x1
    .loadertext - base 0x200b6a08 - size 0x00000000 - access : 0x4
       .syscall - base 0x200b6a08 - size 0x00000008 - access : 0x1
     .intercall - base 0x200b6a10 - size 0x00000000 - access : 0x4
     .interfunc - base 0x200b6a10 - size 0x00000006 - access : 0x1
       .fixaddr - base 0x200b6a16 - size 0x00000000 - access : 0x4
       .fixtype - base 0x200b6a16 - size 0x00000000 - access : 0x4
       .secinfo - base 0x200b6a18 - size 0x0000067c - access : 0x2
    [...]
```

- Early initialization phase
- The kernel parses some kind of memory region descriptors
- Informative error message:

```
errlog("Failed to build memory map\n");
raise("F:\\bspwd\\int1002\\modules\\ghs\\bspsrc\\support\\bsp_common.c", 130);
```

- Each of these descriptors is then registered as resource (exposed)

```
rsrc_mgr_reglink_msg(memory_object, entry->name, "!systempassword") )
```

```
struct  MEMORY_REGION
{
  int id;
  int field_4;
  int low;
  int high;
  int field_10;
  int field_14;
  int field_18;
  int field_1C;
  const char *mr_name;
  int field_24;
  MEMORY_REGION *next;
  MEMORY_REGION *self;
  int field_30;
};
```

```
>> id: 0x0507 - reg: MRSRAM_ASM - low 0xd0000000 / high 0xd0007fff
>> id: 0x0507 - reg: MRSRAM_CFG - low 0xd000a000 / high 0xd003ffff
>> id: 0x0507 - reg:    MRC0008 - low 0xc0008000 / high 0xc0008fff
>> id: 0x0507 - reg: MRDOWNLOAD - low 0x207ba000 / high 0x221ba000
>> id: 0x0507 - reg: MRVIDEORAM - low 0x00000000 / high 0x00000fff
[...]
>> id: 0x0507 - reg:    MR80108 - low 0x80108000 / high 0x8010bfff
>> id: 0x0507 - reg:    MR801F0 - low 0x801f0000 / high 0x801f0fff
>> id: 0x0507 - reg:    MR80200 - low 0x80200000 / high 0x80200fff
[...]
>> id: 0x0507 - reg:    MRC0000 - low 0xc0000000 / high 0xc0000fff
>> id: 0x0507 - reg:    MRC0001 - low 0xc0001000 / high 0xc0001fff
>> id: 0x0507 - reg:    MRC0002 - low 0xc0002000 / high 0xc0002fff
[...]
>> id: 0x0507 - reg:    MRD1000 - low 0xd1000000 / high 0xd1000fff
>> id: 0x0507 - reg: __ghs_bcncsi_buffers - low 0x00000000 / high 0x000005ff
>> id: 0x0407 - reg: __ghs_umac_descriptors - low 0x00000000 / high 0x000007ff
>> id: 0x0507 - reg: __ghs_umacdev_mirror_buffers - low 0x00000000 / high 0x00005fff
>> id: 0x0507 - reg: __ghs_tcpip_buffers - low 0x00000000 / high 0x00017fff
>> id: 0x0507 - reg: __ghs_ktstncsi_phy_buffers - low 0x00000000 / high 0x000005ff
```

Outline

HP provides access to iLO from the host operating system

- iLO is seen as a PCI device
- Driver is provided in the Linux kernel

Handled on iLO side by the CHIF task

- Various command handlers
- Gives access to Redfish API
- CHIF can also pass some commands to other tasks, such as embmedia

Commands are passed through shared memory

- Linux driver implements 8 to 24 channels in the device shared memory
- Commands are sent in a FIFO

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

iLO access from host is privileged

- No authentication required
- Ability to add new users/administrators
- Ability to access the Redfish API

Example: access the default settings stored in EEPROM (CHIF command 0x70):

```
>>> f=open("/dev/hpilo/d0ccb1", "wb+")
>>> data = "MFGDiag\x00" + pack("<L", 1) # Read eeprom, offset 0
>>> 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 01 00 00 00   ....p...........
0010  43 5a 31 37 31 35 30 31 47 39 20 20 20 20 20   CZ171501G9
0020  00 00 00 00 00 00 00 00 02 00 00 00 ff ff ff   ................
0030  ff ff ff ff 41 64 6d 69 6e 69 73 74 72 61 74   ....Administrato
0040  72 00 00 00 00 00 00 00 00 00 00 00 47 xx xx   r...........G***
0050  36 4e 4a 37 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   ................
0070  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+......
```

- Network scan:

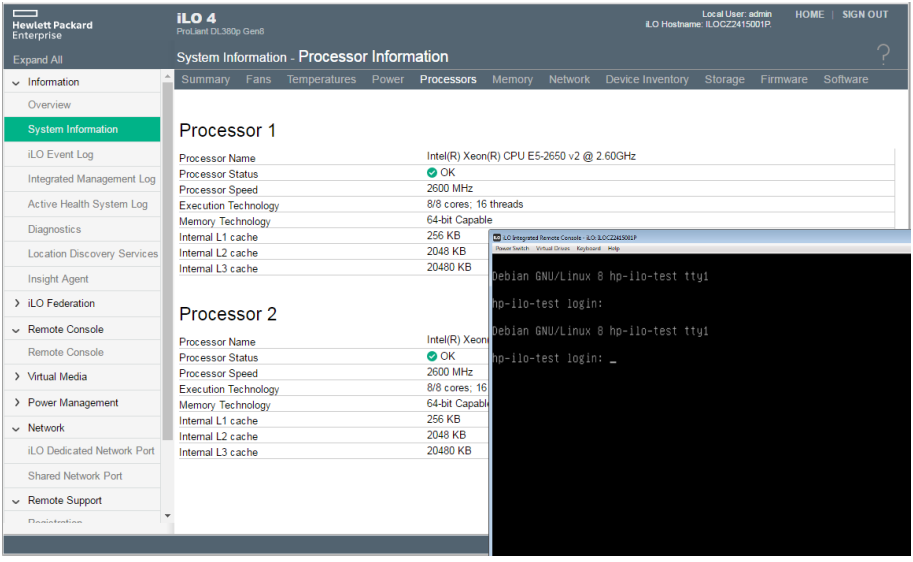
```
Host is up (0.015s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE  VERSION
22/tcp    open  ssh      HP Integrated Lights-Out mpSSH 0.2.1 (protocol 2.0)
80/tcp    open  http     HP Integrated Lights-Out web interface 1.30
443/tcp   open  ssl/http HP Integrated Lights-Out web interface 1.30
17988/tcp open  ilo-vm   HP Integrated Lights-Out Virtual Media
Service Info: CPE: cpe:/h:hp:integrated_lights-out, cpe:/h:hp:integrated_lights-out:1.30
```

Summary of exposed endpoints:

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

Outline

Webserver


Hewlett Packard Enterprise
iLO 4
ProLiant DL380p Gen8
Local User: admin
iLO Hostname: ILOCZ2415001P
HOME | SIGN OUT
Expand All
System Information - Processor Information
Information
Overview
System Information
iLO Event Log
Integrated Management Log
Active Health System Log
Diagnostics
Location Discovery Services
Insight Agent
iLO Federation
Remote Console
Remote Console
Virtual Media
Power Management
Network
iLO Dedicated Network Port
Shared Network Port
Remote Support
Summary Fans Temperatures Power Processors Memory Network Device Inventory Storage Firmware Software
Processor 1
Processor Name Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
Processor Status OK
Processor Speed 2600 MHz
Execution Technology 8/8 cores; 16 threads
Memory Technology 64-bit Capable
Internal L1 cache 256 KB
Internal L2 cache 2048 KB
Internal L3 cache 20480 KB
Processor 2
Processor Name Intel(R) Xeon
Processor Status OK
Processor Speed 2600 MHz
Execution Technology 8/8 cores; 16
Memory Technology 64-bit Capable
Internal L1 cache 256 KB
Internal L2 cache 2048 KB
Internal L3 cache 20480 KB
iLO Integrated Remote Console - iLO: ILOCZ2415001P
Power Switch Virtual Drives Keyboard Help
Debian GNU/Linux 8 hp-ilo-test tty1
hp-ilo-test login:
Debian GNU/Linux 8 hp-ilo-test tty1
hp-ilo-test login:

Web server provides:

- A shiny web interface
- An `XML` API
- A Redfish API
- A remote console through a Java applet

GOOD
Access to almost all pages is authenticated.

BAD
An `/xmldata` endpoint can be accessed without authentication.

Getting information: /xmldata?item=all

Getting information: /xmldata?item=all

```
<PN>Integrated Lights-Out 4 (iLO 4)</PN>
<RIMP>
<HSI>
<SBSN>CZ24370KKL </SBSN>
<SPN>ProLiant DL380p Gen8</SPN>
<UUID>653200CZ24370KKL</UUID>
<SP>1</SP>
<FWRI>2.55</FWRI>
<BBLK>03/05/2013</BBLK>
<HWRI>ASIC: 12</HWRI>
<SN>ILOCZ24370KKL </SN>
<UUID>ILO653200CZ24370KKL</UUID>
<IPM>1</IPM>
<SSO>0</SSO>
<PWRM>3.3.0</PWRM>
```

Getting external **and internal** network interfaces information? `/xmldata?item=all`

Getting external **and internal** network interfaces information? `/xmldata?item=all`

```
<NIC>
<PORT>1</PORT>
<DESCRIPTION>iLO 4</DESCRIPTION>
<LOCATION>Embedded</LOCATION>
<MACADDR>9c:b6:52:b1:2a:62</MACADDR>
<IPADDR>10.123.48.19</IPADDR>
<STATUS>OK</STATUS>
</NIC>
<NIC>
<PORT>2</PORT>
<DESCRIPTION>iLO 4</DESCRIPTION>
<LOCATION>Embedded</LOCATION>
<MACADDR>9c:b6:52:b1:2a:63</MACADDR>
<IPADDR>192.168.1.152</IPADDR>
<STATUS>OK</STATUS>
</NIC>
```

Getting a **free Advanced License**? /xmldata?item=CpqKey

Getting a **free Advanced License**? /xmldata?item=CpqKey

```
<PROLIANTKEY>
<LNAME>iLO Advanced</LNAME>
<IDATE>Thu Nov 19 03:02:39 2015</IDATE>
<KEY>35DQ5-*****-8YTD2-*****-X5W7R</KEY>
</PROLIANTKEY>
```

Teasing for later

After authentication, a hidden debug page can be accessed.

- pci_info :
  - 0.
    - Fn0 PCI-E Status Reg CSMPCISR : 0x0010
    - Fn2 PCI-E Status Reg CHIFPCISR : 0x0010
    - Fn3 PCI-E Status Reg WDGPCISR : 0x0010
    - Fn4 PCI-E Status Reg UHCIPCISR : 0x0010
    - Fn5 PCI-E Status Reg VSPPCISR : 0x0010
    - Fn6 PCI-E Status Reg IPMIPCISR : 0x0010
    - Fn0 PCI-E Device Status Reg : 0x0008
    - Fn2 PCI-E Device Status Reg : 0x0008
    - Fn3 PCI-E Device Status Reg : 0x0000
    - Fn4 PCI-E Device Status Reg : 0x0000
    - Fn5 PCI-E Device Status Reg : 0x0000
    - Fn6 PCI-E Device Status Reg : 0x0000
    - PCI-E Err Stat Reg PERSTAT : 0xf0000400
    - Sys Flt Stat Reg SYSFAULT : 0x0000
    - CSM NMI Stat Reg (Fn0) NMISTAT : 0x00
    - SIRQ config Reg : 0x00
    - Fn0 PCI-E I/O BAR : 0x00003001
    - Fn2 PCI-E I/O BAR : 0x00003801
    - Fn5 PCI-E I/O BAR : 0x00000001
    - Fn6 PCI-E Memory BAR : 0x00000000

Information about various `PCI` registers

What about the web server architecture?

- Web server has 4 concurrent threads
- Each thread handles a connection
- Supports `HTTP` and `HTTPS`

What about the web server architecture?

- Web server has 4 concurrent threads
- Each thread handles a connection
- Supports `HTTP` and `HTTPS`

New connection handling:

- Each request is parsed line by line

What about the web server architecture?

- Web server has 4 concurrent threads
- Each thread handles a connection
- Supports `HTTP` and `HTTPS`

New connection handling:

- Each request is parsed line by line
- Various `strcmp()`, `strstr()` and `sscanf()` calls

What about the web server architecture?

- Web server has 4 concurrent threads
- Each thread handles a connection
- Supports `HTTP` and `HTTPS`

New connection handling:

- Each request is parsed line by line
- Various `strcmp()`, `strstr()` and `sscanf()` calls
- **What could possibly go wrong?**

```
ADD             R1, R6, #8
ADD             R0, R6, #4
STMEA           SP, {R0,R1}
MOV             R3, R8
ADD             R2, SP, #0x27C+var_30
ADR             R1, a9s1023sHttpd_D ; "%9s %1023s HTTP/%d.%d"
MOV             R0, R5
BL              sscanf
```

OK

sscanf() is hard!

```
ADD             R1, R6, #8
ADD             R0, R6, #4
STMEA           SP, {R0,R1}
MOV             R3, R8
ADD             R2, SP, #0x27C+var_30
ADR             R1, a9s1023sHttpd_D ; "%9s %1023s HTTP/%d.%d"
MOV             R0, R5
BL              sscanf
```

OK

```
ADR             R2, aAuthorization ; "Authorization:"
MOV             R1, R5
MOV             R0, R7
MOV             R3, #0xE
BL              strncmp
CMP             R0, #0
BNE             loc_14F9C
MOV             R0, R6
BL              sub_17B6C
ADD             R2, SP, #0x27C+var_40
MOV             R3, R0
ADR             R1, aS15s16383s ; "%*s %15s %16383s"
MOV             R0, R5
BL              sscanf
```

OK

```
ADD             R1, R6, #8
ADD             R0, R6, #4
STMEA           SP, {R0,R1}
MOV             R3, R8
ADD             R2, SP, #0x27C+var_30
ADR             R1, a9s1023sHttpD_D ; "%9s %1023s HTTP/%d.%d"
MOV             R0, R5
BL              sscanf
```

OK

```
ADR             R2, aAuthorization ; "Authorization:"
MOV             R1, R5
MOV             R0, R7
MOV             R3, #0xE
BL              strncmp
CMP             R0, #0
BNE             loc_14F9C
MOV             R0, R6
BL              sub_17B6C
ADD             R2, SP, #0x27C+var_40
MOV             R3, R0
ADR             R1, aS15s16383s ; "%*s %15s %16383s"
MOV             R0, R5
BL              sscanf
```

OK

```
ADR             R2, aContentLength ; "Content-length:"
MOV             R1, R5
MOV             R0, R7
MOV             R3, #0xF
BL              strncmp
CMP             R0, #0
BNE             loc_14FDC
STR             R0, [SP,#0x27C+var_26C]
ADD             R2, SP, #0x27C+var_26C
ADR             R1, aSD ; "%*s %d"
MOV             R0, R5
BL              sscanf
```

OK

sscanf() is hard!

```
ADD     R1, R6, #8
ADD     R0, R6, #4
STMEA   SP, {R0,R1}
MOV     R3, R8
ADD     R2, SP, #0x27C+var_30
ADR     R1, a9s1023sHttpd_D ; "%9s %1023s HTTP/%d.%d"
MOV     R0, R5
BL      sscanf
```

OK

```
ADR     R2, aAuthorization ; "Authorization:"
MOV     R1, R5
MOV     R0, R7
MOV     R3, #0xE
BL      strncmp
CMP     R0, #0
BNE     loc_14F9C
MOV     R0, R6
BL      sub_17B6C
ADD     R2, SP, #0x27C+var_40
MOV     R3, R0
ADR     R1, aS15s16383s ; "%*s %15s %16383s"
MOV     R0, R5
BL      sscanf
```

OK

```
ADR     R2, aContentLength ; "Content-length:"
MOV     R1, R5
MOV     R0, R7
MOV     R3, #0xF
BL      strncmp
CMP     R0, #0
BNE     loc_14FDC
STR     R0, [SP,#0x27C+var_26C]
ADD     R2, SP, #0x27C+var_26C
ADR     R1, aSD ; "%*s %d"
MOV     R0, R5
BL      sscanf
```

OK

```
ADR     R2, aConnection ; "Connection:"
MOV     R1, R5
MOV     R0, R7
MOV     R3, #0xB
BL      strncmp
CMP     R0, #0
ADDEQ   R2, R7, #0xC
ADREQ   R1, aSS_0 ; "%*s %s"
MOVEQ   R0, R5
BLEQ    sscanf
```

FAIL

The web server parses the HTTP headers from user's request:

```
1  else if ( !strnicmp(request, http_header, "Content-length:", 0xFu) )
2  {
3    content_length = 0;
4    sscanf(http_header, "%*s %d", &content_length);
5    state_set_content_length(global_struct_, content_length);
6  }
7  else if ( !strnicmp(request, http_header, "Cookie:", 7u) )
8  {
9    cookie_buffer = state_get_cookie_buffer(global_struct_);
10   parse_cookie(request, http_header, cookie_buffer);
11 }
12 else if ( !strnicmp(request, http_header, "Connection:", 0xBu) )
13 {
14   sscanf(http_header, "%*s %s", https_connection->connection);
15 }
```

- String parsing function:

```
int sscanf ( const char * s, const char * format, ...);
```

- Definition: *Read formatted data from string. Reads data from s and stores them according to parameter format into the locations given by the additional arguments.*

Parsing is hard

```
sscanf(http_header, "%*s %s", https_connection->connection);
```

- Basically says "*copy whatever you can into* `connection`" buffer
- No sanitization/bound checking
- Destination buffer is only 16 bytes ⇒ **buffer overflow**

R7 points to a `https_connection` object
R7+0xC points to a fixed-size buffer

```
struct https_connection {
        ...
        0x0C: char connection[0x10];
        ...
        0x28: char localConnection;
        ...
}
```

Host system can talk to iLO through shared memory

- Redfish API can be requested from host
- Authentication is not required!
- Web server uses the `localConnection` boolean to check if the connection comes from the host

Host system can talk to iLO through shared memory

- Redfish API can be requested from host
- Authentication is not required!
- Web server uses the `localConnection` boolean to check if the connection comes from the host

Redfish API allows unauthenticated access when `localConnection` is not `zero`

Host system can talk to iLO through shared memory

- Redfish API can be requested from host
- Authentication is not required!
- Web server uses the `localConnection` boolean to check if the connection comes from the host

Redfish API allows unauthenticated access when `localConnection` is not `zero`

What about a nice cup of `AAAAAAAAAAAAAAAAAAAAAAAAAAAA`?

DEMO

- Pre-auth buffer overflow in web server `HTTP` headers handling
- Vulnerability was reported to `HP` on February 2017
- Vulnerability can be exploited for authentication bypass
  - Ability to add a new administrator account
  - Access to the remote console
  - Ability to restart the server on an arbitrary `ISO`
- We need to go deeper!

Web server objects have their *vtable* pointer at the end of the structure:

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

Web server objects have their *vtable* pointer at the end of the structure:

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

There is **no** ASLR.

Web server objects have their *vtable* pointer at the end of the structure:

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

There is **no** ASLR.

There is **no** NX.

Web server objects have their *vtable* pointer at the end of the structure:

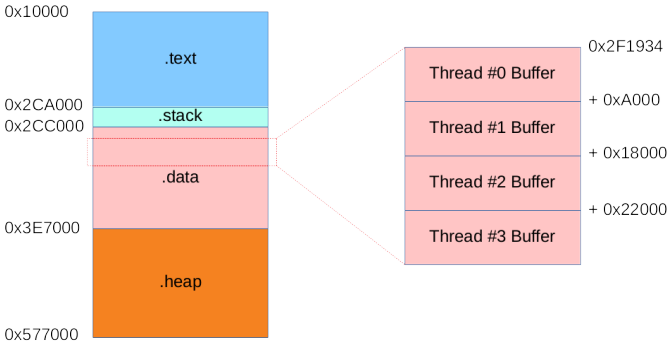
```
struct https_connection {
        ...
        0x0C: char connection[0x10];
        ...
        0x28: char localConnection;
        ...
        0xB8: void *vtable;
}
```

There is **no** ASLR.

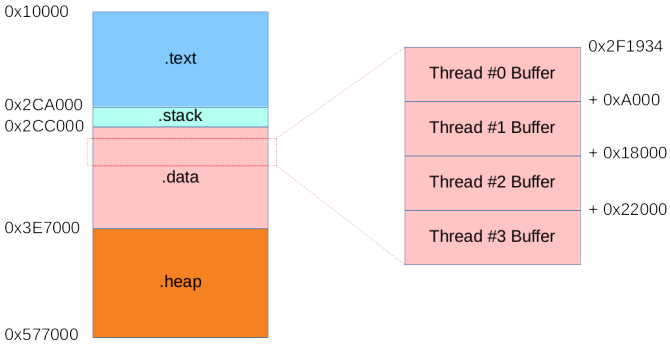
There is **no** NX.

Each web server thread has a *working buffer* at a fixed address.

Working buffer



Working buffers receive each line during `HTTP` request parsing.



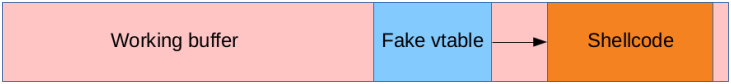
Working buffers receive each line during `HTTP` request parsing.

Working buffers can be populated with controlled data!

Write shellcode in working buffer

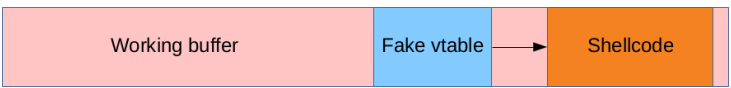
Write shellcode in working buffer

Write fake *vtable* in working buffer



Write shellcode in working buffer

Write fake *vtable* in working buffer



Trigger the buffer overflow to overwrite the object's *vtable* pointer!

We can craft an arbitrary shellcode which:

- Reads contents of the password file on the `iLO` filesystem
  - Passwords are stored in cleartext
- Sends back results through the `HTTPS` socket

DEMO

- Buffer overflow can be turned into a reliable RCE
- We can now execute an arbitrary shellcode in the webserver context
- We need to go deeper! `iLO` should be able to access the main physical memory.

Outline

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
```

Keywords: SMBIOS type_229, $WHE, HostRAM

Type 229 is OEM defined. Let's dump it from our Linux!

```
0000   24 44 46 43 00 50 fe f1 00 00 00 00 00 04 00 00   $DFC.P..........
0010   24 43 52 50 00 50 f9 f1 00 00 00 00 00 00 05 00   $CRP.P..........
0020   24 48 44 44 00 30 f9 f1 00 00 00 00 00 20 00 00   $HDD.0....... ..
0030   24 4f 43 53 00 f0 f8 f1 00 00 00 00 00 40 00 00   $OCS.........@..
0040   24 4f 43 42 00 f0 f7 f1 00 00 00 00 00 00 01 00   $OCB............
0050   24 53 41 45 00 e0 f7 f1 00 00 00 00 00 10 00 00   $SAE............
```

Looks like structures:

```
struct entry229 {
  char tag[4];
  void *pointer64;
  int flags;
}
```

Let's check one of these pointers in physical memory...

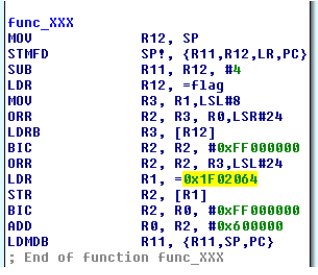
```
root@ilo-server-ubuntu:~# xxd -s $((0xf1f95000)) /dev/mem|head -n 8

f1f95000: 2452 4253 0000 0000 0001 0069 0813 0400  $RBS.......i....
f1f95010: 0113 0400 0101 6f00 0000 0001 6752 4f4d  ......o.....gROM
f1f95020: 2d42 6173 6564 2053 6574 7570 2055 7469  -Based Setup Uti
f1f95030: 6c69 7479 2c20 5665 7273 696f 6e20 332e  lity, Version 3.
f1f95040: 3030 0d0a 436f 7079 7269 6768 7420 3139  00..Copyright 19
f1f95050: 3832 2c20 3230 3135 2048 6577 6c65 7474  82, 2015 Hewlett
f1f95060: 2d50 6163 6b61 7264 2044 6576 656c 6f70  -Packard Develop
f1f95070: 6d65 6e74 2043 6f6d 7061 6e79 2c20 4c2e  ment Company, L.
```

Looks great!

```
char whea_header[0x18];
int *ptr_entry = find_in_smbios_229("$WHE");
if (ptr_entry) {
  int phy_ptr_low = ptr_entry[1];
  int phy_ptr_high = ptr_entry[2];

  void *whea_ptr = func_XXX(phy_ptr_low, phy_ptr_high);
  sawbase_memcpy_s(whea_header, whea_ptr, 0x18);
  [...]
}
```



```
void *func_XXX(void *ptr_low, void *ptr_hi) {
  int magic = (flag<<24) |
              (((ptr_hi << 8) | (ptr_low >> 24)) & 0x00ffffff);
  *(0x1f02064) = magic;
  return (ptr_low & 0x00ffffff) | 0x600000;
}
```

- Passed 64-bits pointer is truncated to a 16MB boundary
- Flag is set to 2
- What is mapped at 0x1F02064?
- What is mapped at 0x600000?

- Range of iLO **physical memory**
- Mapped in a userland task virtual memory
- Requested as a resource from userland
  - ```
    sprintf(mr_name, "MR%X", mr_physical >> 12);
    RequestResource(&mr_object, mr_name, "!systempassword");
    ```
  - RequestResource setup and sends a request to the kernel, made of:
    - A verb: "procure"
    - The name of the object, ex: "MR80200"
    - A password: "!systempassword"

Each task has a list of Memory Region which can be mapped in its virtual memory, by calling memmap().
    ```

CHIF task maps the following Memory Region:

Physical	Virtual	Size	
0x80000000	0x1F00000	0x1000	MR80000
0x800F0000	0x1F01000	0x1000	MR800F0
0x80200000	0x1F02000	0x1000	MR80200
0x802F0000	0x1F03000	0x1000	MR802F0
0x804F0000	0x1F07000	0x1000	MR804F0
0x82000000	0x600000	0x1000000	MR82000
0xC0000000	0x1F10000	0x1000	MRC0000
0xD1000000	0x1F14000	0x1000	MRD1000

We now have our virtual $\Leftrightarrow$ physical mapping:

- 0x1F02064 is the mapping of 0x80200064
- 0x600000 is the mapping of 0x82000000

Remember the `dbug.html` page?

Reversing it gives information about the `0x1F00000` memory range.

```
1f01006              Fn0 PCI-E Status Reg  CSMPCISR
1f01010              Fn0 PCI-E I/O BAR
1f010ca              Fn0 PCI-E Device Status Reg
1f02078              PCI-E Err Stat Reg PERSTAT
1f020b4              Sys Flt Stat Reg SYSFAULT
1f03006              Fn2 PCI-E Status Reg CHIFPCISR
1f03010              Fn2 PCI-E I/O BAR
1f030ca              Fn2 PCI-E Device Status Reg
1f05006              Fn3 PCI-E Status Reg  WDGPCISR
1f050ca              Fn3 PCI-E Device Status Reg
1f07006              Fn4 PCI-E Status Reg UHCIPCISR
1f070ca              Fn4 PCI-E Device Status Reg
1f09006              Fn5 PCI-E Status Reg  VSPPCISR
1f09010              Fn5 PCI-E I/O BAR
1f090ca              Fn5 PCI-E Device Status Reg
1f0b006              Fn6 PCI-E Status Reg IPMIPCISR
1f0b010              Fn6 PCI-E Memory BAR
1f0b0ca              Fn6 PCI-E Device Status Reg
```

`0x1F02064` should be an unknown PCI register.

- Reversing CHIF told us how to access host memory
- Let's implement this technique in the webserver task through a shellcode

Shellcode will fill a 16MB `Memory Region` with host memory.

- Take a **host** physical memory address
- Shift it right by 24
- Add flag
- Write the value in register `0x1F02064`
- ???
- Profit by accessing `MR82000`!

After testing, we can access almost any memory address in the host physical memory, with **read and write** access.

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

We have fully compromised the platform, how to remain persistent accross reboot without modifying the file system of the host?

Let's investigate!

Outline

Many advantages:

- Survives an OS reinstallation
- Extremely stealth: who can do forensics on iLO?

Not so simple (theoretically):

- Updates are signed
- Boot-time integrity check

Firmware updates can be performed through the web interface, or directly from the host.

Update mechanism analysis shows two important points:

- Downgrading is possible! Long life to vulnerabilities!
- Signature is correctly checked

However, if we get RCE, **we can bypass this signature check and write our own firmware on the flash chip**.

According to HP:

> At boot time, each piece has its signature validated before it is allowed to execute. Subsequent pieces are checked by the previous ones until iLO is fully booted.

However... **who checks the bootloader signature?**

Gaining persistence in 6 steps:

- Patch bootloader signature check of the kernel
- Patch kernel signature check of the userland
- Inject a malicious payload in a userland task
- Patch the firmware update userland task to reinject the backdoor in case of firmware update
- Rebuild a correct image (`ELF` structure, compression)
- Flash the new image using a custom shellcode

Outline

- Documentation of the Integrity image internals
- Huge exposed attack surface
- IPMI v2 authentication **weak by design**
- One critical vulnerability discovered, fixed by the vendor[6]
- Unpatched server ⇒ **trivial remote authentication bypass** and **full compromise** (code execution) of the iLO system
- **New DMA based exploitation technique** ⇒ full compromise of the host system from the iLO system
- No hardware root of trust: open doors to compromised firmware[7]
- Remote administration services should be isolated in a dedicated administration VLAN

[6] CVE-2017-12542, CVSSv3 9.8

[7] Fix proposed in iLO5, released mid-2017, see "silicon root of trust", https://support.hpe.com/hpsc/doc/public/display?docId=a00018320en_us

Thank you 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