Defeating NotPetya from your iLO4

Joffrey Czarny (Medallia) Alexandre Gazet (Airbus) Adrien Guinet (Quarkslab)
Fabien Perigaud (Synacktiv)

Introduction

NotPetya is a variant of the Petya ransomware that appeared in June 2017 in Ukraine [1].

These malwares have the particularity to rewrite the MBR of computers that are still using an old fashioned BIOS-based booting system. This MBR encrypts the Master File Table (MFT) of the underlying NTFS partition systems [2].

Previous work by Adrien Guinet [3] showed that the encryption key that is used to do this encryption stays in RAM after the encryption process. If you have the ability to read/write the memory of a running computer, chances are that you can decrypt this encryption stage.

One year later, Alexandre Gazet, Fabien Perigaud and Joffrey Czarny [4] found vulnerabilities in HP iLO's servers that allow them, among other things, to read and write into the physical memory of a running HP server.

This paper describes experiments where we can decrypt NotPetya's bootloader encryption using these vulnerabilities.

DMA access through iLO4

Exposed host memory DMA access

In our previous works on iLO4 and iLO5 BMC, we discovered that these chips take advantage of a hardware feature that allows them to perform DMA accesses to the host memory. Using a backdoored firmware, it is possible to expose this feature over the iLO's web interface.

From a backdoored system, one can read/write the host memory using simple HTTP queries. In the following example, we simply read `0x10000` bytes from the physical address `0x10000` and then dump the output in the `dmp.bin` file. The host system is a Windows 10.

```
$ wget --no-check-certificate -O dmp.bin 'https://192.168.42.78/backd00r.htm?act=
    dmp&hiaddr=0&loaddr=10000&count=10000'
--2019-01-25 17:29:03--  https://192.168.42.78/backd00r.htm?act=dmp&hiaddr=0&
    loaddr=10000&count=10000
Connecting to 192.168.42.78:443... connected.
WARNING: The certificate of ‘’192.168.42.78 is not trusted.
WARNING: The certificate of ‘’192.168.42.78 doesn't have a known issuer.
The certificate's owner does not match hostname ‘’192.168.42.78
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: ‘dmp.’bin

dmp.bin                          [ <=>                                   ]  64.00K
    --.-KB/s    in 0.05s

2019-01-25 17:29:04 (1.15 MB/s) - ‘dmp.’bin saved [65536]
```

For instance, we can recover the Windows bootloader:

```
$ 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  ...............'
[...]
```

Using this interface for PCILeech

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. It relies on several hardware or software interfaces to read and write physical memory.

Having the ability to do the same through HP iLO, we developed tooling to interface PCILeech with our backdoored iLO firmware [5].

This allowed us to make use of PCILeech using iLO as a DMA device:

```
$ ./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
```

Fixing the ransomware

In this section, we refer to addresses that are valid when the bootloader has been loaded and mapped in memory by the code present in the Master Boot Record. The provided script `remap_bootloader.py` simulates this and generates a bootloader.remap file that will contain the memory loaded at `0x7C00`. You can for instance load this file into IDA in x86 16-bit real mode, and rebase it at `0x7C00`.

The ransomware uses a modified (and broken [2]) implementation of Salsa20. Using a Miasm script to emulate NotPetya's bootloader [6], we can see that this key is still present in memory. It basically works by loading the key written by the malware in the sector 32 and search it in the Miasm VM memory. Here is an output example:

```
$ python ./emulate_mbr.py --hook=find_key ../disk.raw

Repairing file system on C:
[...]
CHKDSK is repairing sector 448 of 480 (93%)
[+] Looking for key 369
    f58e9b1b45a29553674c769c9e5506676208793dff5738bca2c5d3ed5ac46 in memory...
[+] Key found at address 0x674aL!
```

We can see that, after encryption, the Salsa20 key still resides in memory at the address `0x674A`. Indeed, at the end of the encryption step, the bootloader call the `INT 0x19` instruction, which is an interrupt that tells the BIOS to reload the MBR at the address `0x7C00` in memory and jump to it. It also states that the rest of the memory shouldn't be wiped. That's why this attack works!

So, once this "reboot" is done, the ransomware message appears and wait for the user to input the "decryption key" he potentially bought. What we will do is patch the bootloader in memory using the HP iLO of our server to add code that will use the Salsa20 key still in memory and jump on the decryption code that is present in the bootloader.

This happens by writing this small "shellcode" (compilable with `nasm`) at the address `0x82A8` (which is basically where the bootloader ends-up after the user presses the `enter` key):

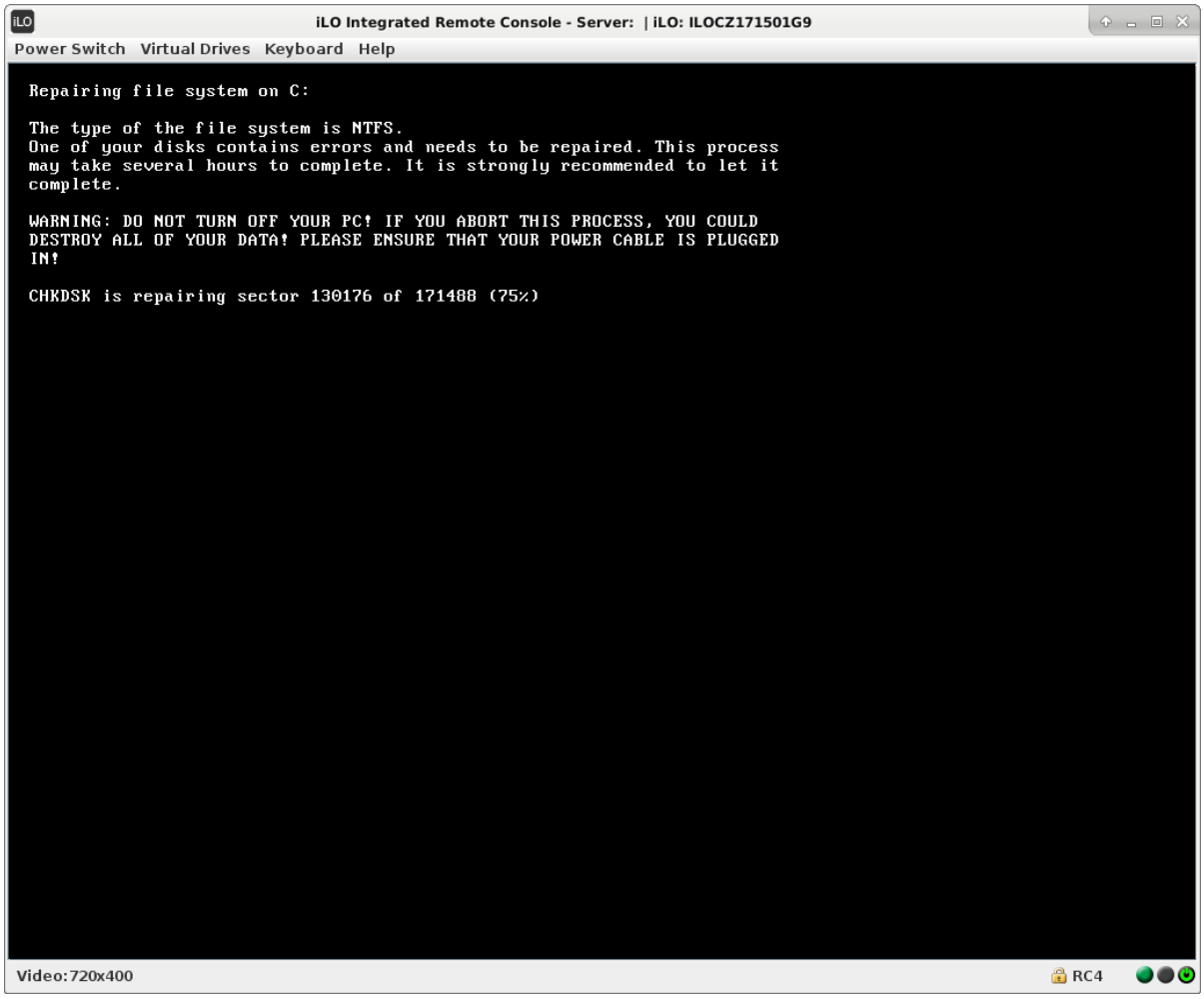
```
bits 16
org 0x82A8

start:
pusha

lea di, [bp-0x44]
lea bx, [0x674A]
xor cx,cx

loop:
mov eax, dword [bx]
mov dword [di], eax
add di, 4
add bx, 4
inc cx
cmp cx,8
jnz loop

popa
```

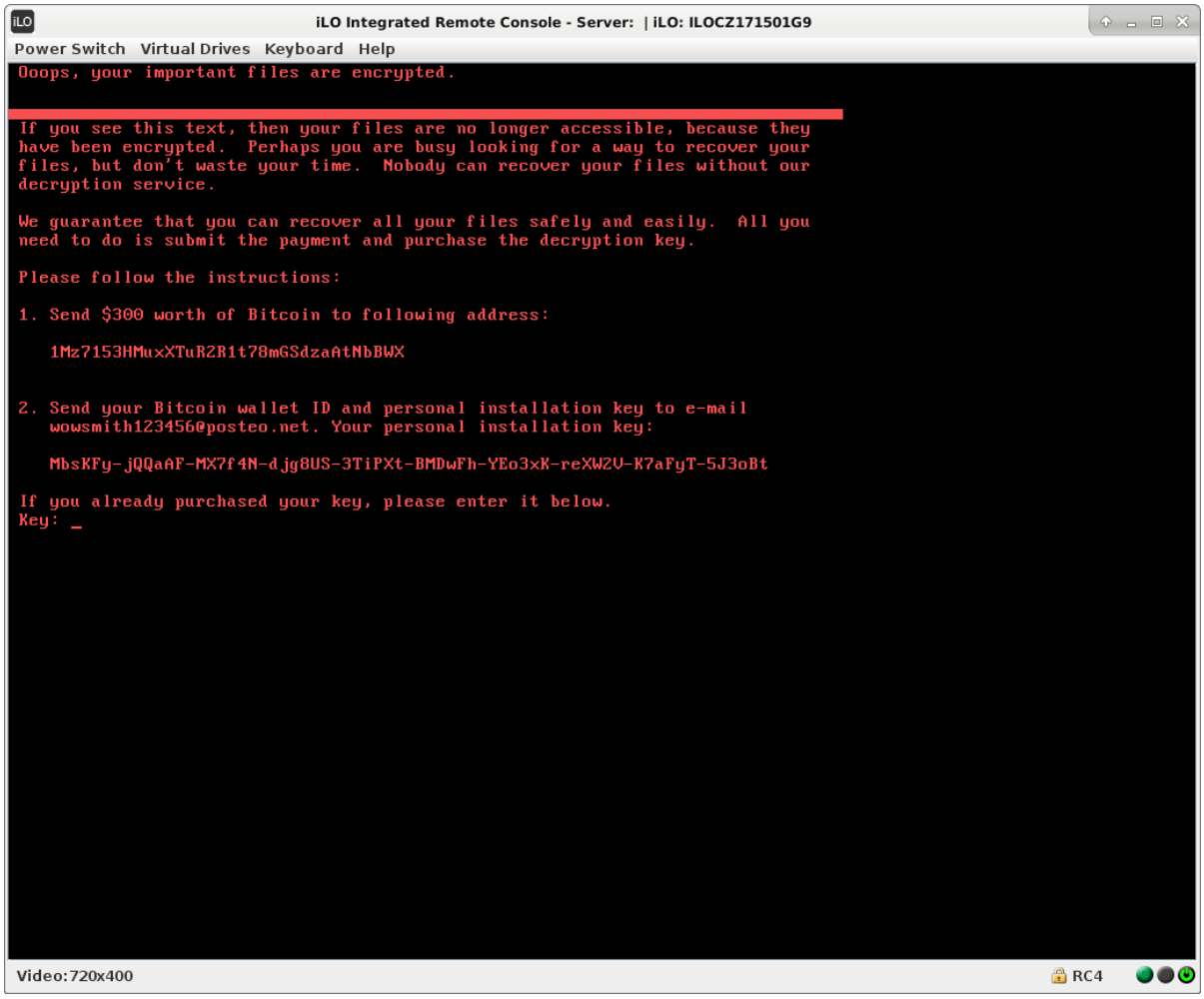
Once the memory is patched, we just press `enter` and the decryption arises!

One last detail to fix is to get a proper reboot of the computer. This is done by patching the memory and add the now famous `INT 0x19` instruction at the address `0x829F` (which is were the bootloader ends-up after the decryption is finished).

Experiment with Windows 10

The first experiment we did was with an up-to-date Windows 10, that we infected by hand. After the reboot of the server, the bootloader launches the encryption of the hard drive, and displayed the ransom message.

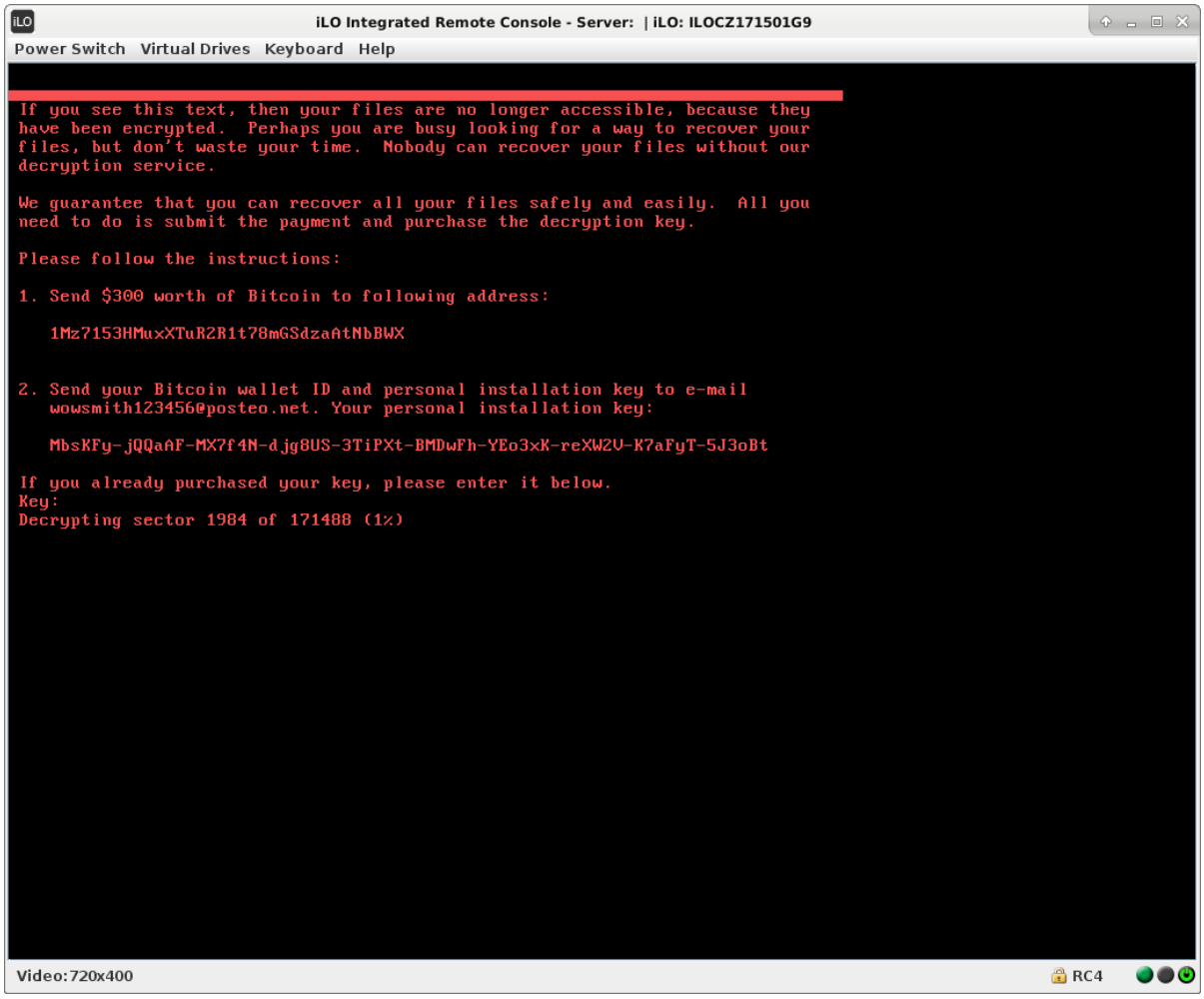

iLO Integrated Remote Console - Server: | iLO: ILOCZ171501G9
Power Switch Virtual Drives Keyboard Help
Repairing file system on C:
The type of the file system is NTFS.
One of your disks contains errors and needs to be repaired. This process
may take several hours to complete. It is strongly recommended to let it
complete.
WARNING: DO NOT TURN OFF YOUR PC! IF YOU ABORT THIS PROCESS, YOU COULD
DESTROY ALL OF YOUR DATA! PLEASE ENSURE THAT YOUR POWER CABLE IS PLUGGED
IN!
CHKDSK is repairing sector 130176 of 171488 (75%)
Video: 720x400
RC4



At this point, we used the provided script [4] to gather the Salsa20 key and patch the bootloader:

```
$ python patch_mem_ilo.py 192.168.42.78
[*] Asked dump of 00000020 bytes at 000000000000674a
[+] Dump OK in 0.284821033478
Key found in memory: 369
    f58e9b1b45a29553674c769c9e5506676208793dff5738bca2c5d3ed5ac46
Do you want to patch the memory? [Y/n] Y
[*] Asked write of 00000020 bytes at 00000000000082a8
[+] Write done in 0.231099128723
[*] Asked write of 00000002 bytes at 000000000000829f
[+] Write done in 0.259984016418
Memory has been patched!
```

Pressing the `enter` key launches the decryption process:



The same operation could also have been done directly with PCILeech, using the `write` command.

It's possible that the Volume Boot Record of the NTFS partition has been erased by the malware. You would see this error:

```
Invalid operating system.
Press Ctrl+Alt+Suppr to reboot
```

In this case, use a Windows 10 install CD to launch a recovery console a fix the boot record using bootrec:

```
bootrec /FixBoot
```

As a final note, the system will boot but files encrypted by the malware when it was running under Windows are still encrypted!

Conclusion

We demonstrated a way to use vulnerabilities in iLO to decrypt NotPetya's bootloader encryption. Scripts are provided if you want to try it yourself!

The processus isn't fully automated, as the user still needs to press `enter` to launch the decryption process. Further work could use iLO to simulate this keystroke.

References

[1] https://en.wikipedia.org/wiki/2017_cyberattacks_on_Ukraine

[2] https://www.crowdstrike.com/blog/petrwrap-ransomware-technical-analysis-triple-threat-file-encryption-mft-encryption-credential-theft/

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

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

[5] https://www.synacktiv.com/posts/exploit/using-your-bmc-as-a-dma-device-plugging-pcileech-to-hpe-ilo-4.html

[6] https://github.com/aguinet/miasm-bootloader/