A deeper UEFI dive into MoonBounce

Archived: 2026-04-05 21:21:26 UTC

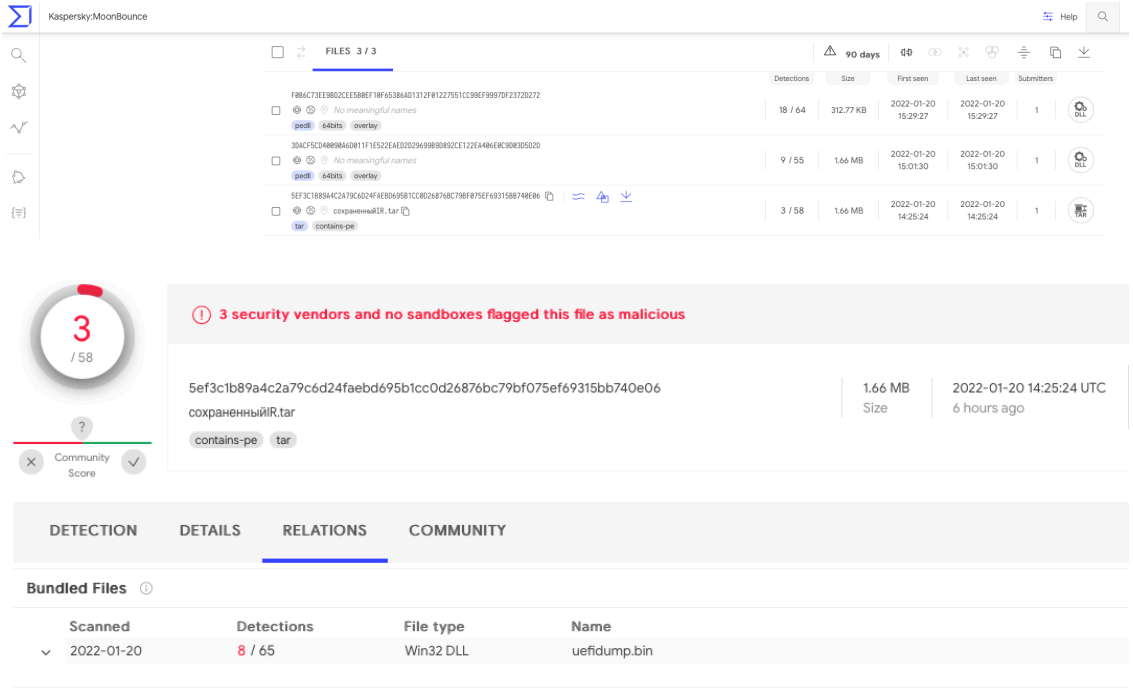
by Binarly Team

After uncovering FinSpy several months ago, an APT threat targeting UEFI bootloaders, in the morning of January 20th 2022, Kaspersky Lab has released a new report on their latest discovery, a very interesting UEFI firmware threat dubbed MoonBounce.

Last year, the ESET researchers discovered ESPecter another threat which also targets EFI bootloaders.

MoonBounce, FinSpy and ESPecter are examples of APT malware comprising components that target both UEFI and Legacy BIOS boot processes.

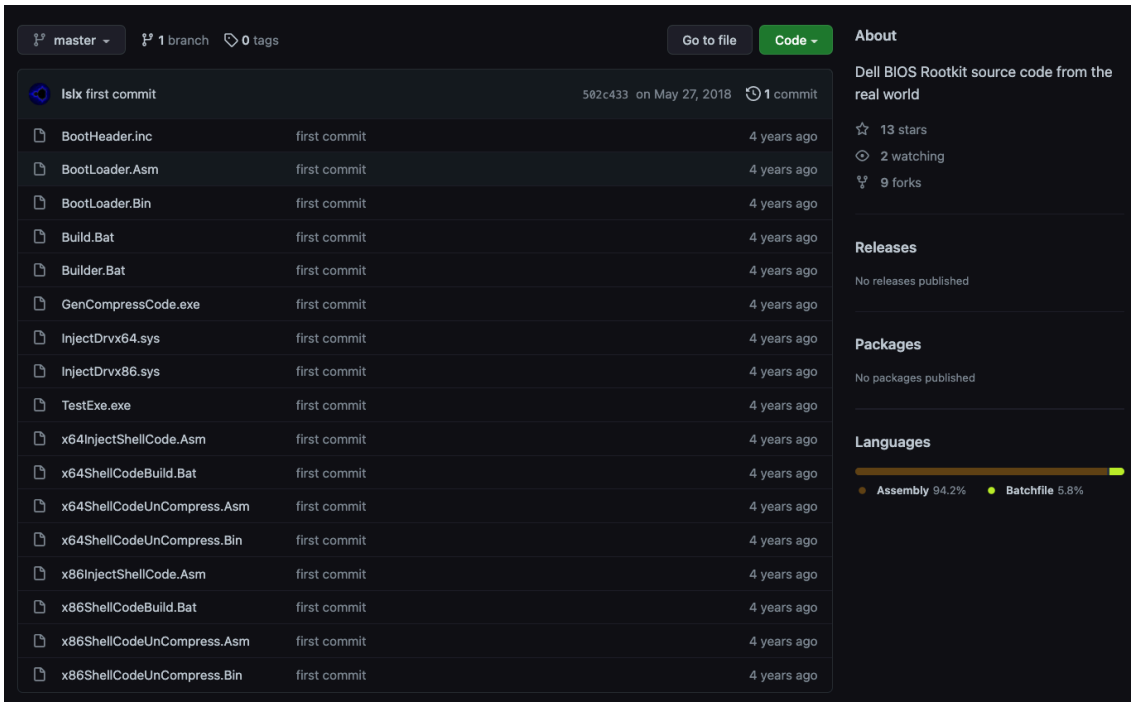
To kickstart our investigation, we leveraged VirusTotal Intelligence and discovered an archive exhibiting the detections mentioned in the Kaspersky Lab's MoonBounce report. Recently, a user uploaded the related samples, which surprised us greatly since we did not expect to find them that quickly by just querying using the detection name.



Let's dive into the dark waters of the MoonBounce firmware implant. In this blog we want to discuss some of the facts which weren't covered by the original report and are interesting to share.

Kaspersky Lab's original report emphasizes that the aforementioned malware consists of a number of known malware components and frameworks: Microcin, Mimikat SSP, xTalker, etc.

Binarly Research Team has analyzed the samples found in VirusTotal and discovered that the UEFI component (the first stage in the malware boot process) is quite old too. It was compiled with borrowed code from an unknown old malware project, most likely previously discovered in the wild on some Dell systems back in 2018 (according to GitHub repository information) and reconstructed in detail in the GitHub repository called "BootLoader".

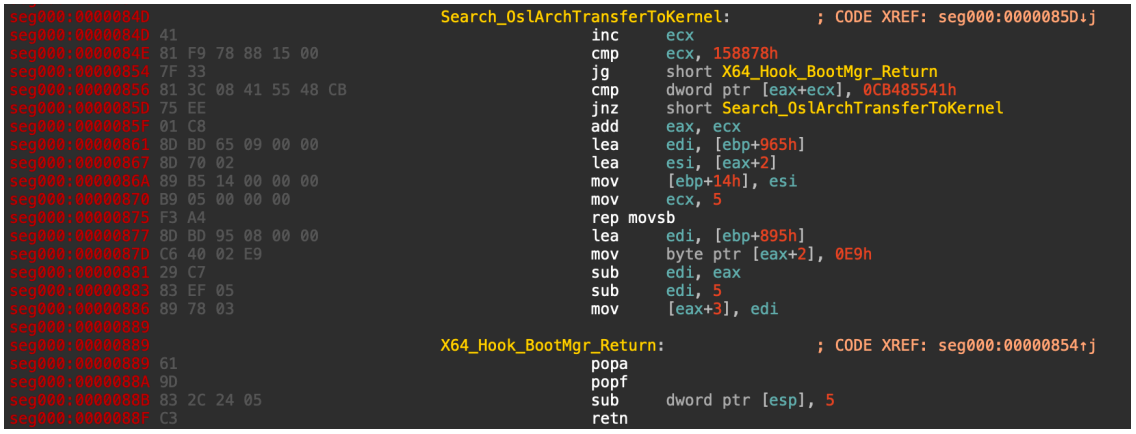


Binarly investigation focused on the firmware research of the UEFI component to provide additional technical information to the already detailed original report on MoonBounce threat.

Our deep dive uncovered similarities between the MoonBounce UEFI component and the binaries available in the BootLoader GitHub repository. Diving into the BootLoader code, visually the hooking routine *Search_OslArchTransferToKernel()* piqued our interest, as it is almost identical with the textual disassembly of the MoonBounce's *CORE_DXE* firmware dump component:

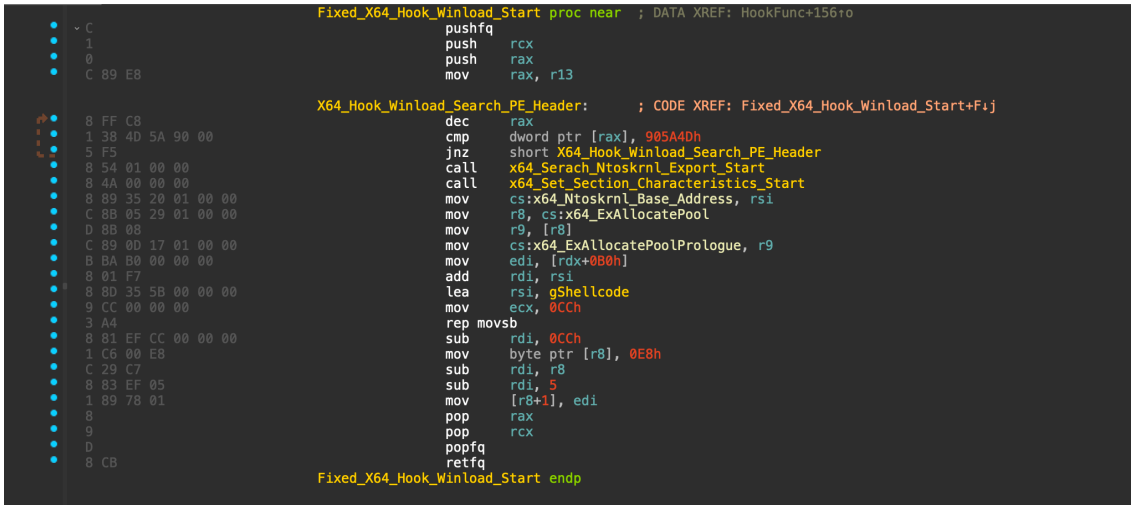
```
seg005:0000000180157837                         Search_OslArchTransferToKernel:         ; CODE XREF: HookFunc+149↓j
seg005:0000000180157837 FF C1                                   inc     ecx
seg005:0000000180157839 81 F9 78 88 15 00                       cmp     ecx, 158878h
seg005:000000018015783F 7F 32                                   jg      short X64_Hook_BootMgr_Return
seg005:0000000180157841 67 81 3C 08 41 55 48 CB                 cmp     dword ptr [eax+ecx], 0CB485541h
seg005:0000000180157849 75 EC                                   jnz     short Search_OslArchTransferToKernel
seg005:000000018015784B 01 C8                                   add     eax, ecx
seg005:000000018015784D 83 C0 02                                add     eax, 2
seg005:0000000180157850 BF 00 80 09 00                          mov     edi, 98000h     ; Fixed_X64_Hook_Winload_Start
seg005:0000000180157855 57                                      push    rdi
seg005:0000000180157856 48 8D 35 21 00 00 00                    lea     rsi, Fixed_X64_Hook_Winload_Start
seg005:000000018015785D B9 29 02 00 00                          mov     ecx, 229h
seg005:0000000180157862 FC                                      cld
seg005:0000000180157863 F3 A4                                   rep movsb
seg005:0000000180157865 5F                                      pop     rdi
seg005:0000000180157866 67 C6 00 E9                             mov     byte ptr [eax], 0E9h
seg005:000000018015786A 29 C7                                   sub     edi, eax
seg005:000000018015786C 83 EF 05                                sub     edi, 5
seg005:000000018015786F 67 89 78 01                             mov     [eax+1], edi
seg005:0000000180157873
seg005:0000000180157873                         X64_Hook_BootMgr_Return:                ; CODE XREF: HookFunc+13F↑j
seg005:0000000180157873 5F                                      pop     rdi
seg005:0000000180157874 5E                                      pop     rsi
seg005:0000000180157875 41 59                                   pop     r9
seg005:0000000180157877 41 58                                   pop     r8
seg005:0000000180157879 5A                                      pop     rdx
seg005:000000018015787A 59                                      pop     rcx
seg005:000000018015787B 5B                                      pop     rbx
seg005:000000018015787C 9D                                      popfq
seg005:000000018015787D C3                                      retn
seg005:000000018015787D                         HookFunc        endp
```

Code from CORE_DXE of MoonBounce firmware dump



Disassembled 32-bit code of BootLoader binary

During early runtime of the DXE phase, it is a known practice employed by malicious actors to modify or hook DXE Services to intercept a boot flow inside the firmware. From a forensics perspective, such modifications are very visible and can be detected using common integrity firmware monitoring approaches.



Disassembly code from CORE_DXE of MoonBounce firmware dump

This code from MoonBounce component is pretty similar with code flow from BootLoader.asm. These observations lead to the conclusion that MoonBounce's authors are the same or use similar code techniques or frameworks to embed their modification into the firmware and Windows kernel.

Additional details can be found in the original Kaspersky report "Technical details of MoonBounce's implementation" (p. 6, "Code that set up a hook in the ExAllocatePool function within ntoskrnl.exe").

Another technical detail we'd like to highlight here relates to the multiple infection delivery paths. The original report explains the usage of *CoreCreateEventInternal()* hook - as support for both UEFI boot and Legacy boot mode (assuming it's targeting deprecated Compatibility Support Module (CSM)).

Compatibility Support Module (CSM) - this module emulates the legacy BIOS in UEFI systems and was developed by Intel to ease the transition to UEFI world. It's pretty common on hardware released before 2020. For newer enterprise hardware is usually disabled by default.

The reason for using *CoreExitBootServices()* is to hook a call from Windows loader (Winload) and prepare the next steps for the MoonBounce boot interception process. What is the point of hooking *InternalAllocatePool()* at the beginning? The possible motivation for this could be to avoid storing the shellcode in a virtual address space along the Windows kernel, by placing it in a physical memory and then executing it directly from there. Such a technique can be used for fileless in memory code execution which would be orchestrated directly from the firmware. During incident response, this will cause complications from a forensic perspective.

Upon further analysis, we found that CORE_DXE contains the target firmware name as a string constant "E7846IMS.M30".

```
rdata:00000001800489E0                  text "UTF-16LE", 'PCI IDE Mass Storage Device',0
rdata:0000000180048A18 aAmiIdeBusDrive:                 ; DATA XREF: .data:off_18004FE20↓o
rdata:0000000180048A18                  text "UTF-16LE", 'AMI IDE BUS Driver',0
rdata:0000000180048A3E                  align 20h
rdata:0000000180048A40 aAmigopoutputdp:                 ; DATA XREF: sub_180040AE0+20↑o
rdata:0000000180048A40                  text "UTF-16LE", 'AmiGopOutputDp',0
rdata:0000000180048A5E                  align 20h
rdata:0000000180048A60 aE7846imsM30     db 'E7846IMS.M30',0     ; DATA XREF: .data:00000001800500C8↓o
rdata:0000000180048A6D                  db    0
rdata:0000000180048A6E                  db    0
rdata:0000000180048A6F                  db    0
```

Therefore, we became curious to determine the exact targeted platform. The first forensic artifact is the PE header timestamp for the CORE_DXE module (Fri Jul 18 03:29:55 2014).

```
CORE_DXE.efi                    ⚠ 2 warnings
  Size                          1.7 MB (1738768 bytes)
  MD5                           2d4991c3b6da35745e0d4f76dffbca56
  SHA1                          61340c41787d16b753598670de2cb1dcf50718c5
  SHA256                        3dacf5cd40090a6d011f1e522eaed2d29699b9d892ce122ea406e0c9d03d5d2d
  Imphash
  Entropy                       2.176121
  MD5 (no overlay)              4ae71336e44bf9bf79d2752e234818a5
  SHA1 (no overlay)             e129f27c5103bc5cc44bcdf0a15e160d445066ff
  SHA256 (no overlay)           374708fff7719dd5979ec875d56cd2286f6d3cf7ec317a3b25632aab28ec37bb
  Entropy (no overlay)          0.000000
  Architecture                  AMD64 (PE+)
  Compiled                      2014-07-18T03:29:55
  IMAGE_DOS_HEADER
  DOS_STUB
  IMAGE_NT_HEADERS
    NT_HEADERS
      Signature                 0x00004550 PE
      FILE_HEADER
        Machine                 0x8664 AMD64
        NumberOfSections        0x0007
        TimeDateStamp           0x53c894b3 Fri Jul 18 03:29:55 2014 UTC
        PointerToSymbolTable    0x00000000
        NumberOfSymbols         0x00000000
        SizeOfOptionalHeader    0x00f0 240 bytes
        Characteristics         0x2022 EXECUTABLE_IMAGE | LARGE_ADDRESS_AWARE | DLL
      OPTIONAL_HEADER
    IMAGE_SECTION_HEADER
  OVERLAY
```

PE Tree snapshot of the CORE_DXE module

On the same day, Taiwanese company MSI released a firmware update for their hardware platform which is very similar to the firmware that has been infected by MoonBounce.

Searching on Google for the string constant "E7846IMS.M30", we found the following website:

E7846IMS.M20	8.00Mb	May 31 2014
E7846IMS.M30	8.00Mb	July 18 2014
E7850IMS.180	8.00Mb	July 21 2014

hxxps://www.mmnt[.]net/db/0/13/lupd01[.]eu[.]msi[.]com/ALL_BIOS_Update_Genie_update_20140831

After obtaining the original firmware image, we were curious about the similarity between the original CORE_DXE component and the modified one in MoonBounce. According to our code analysis (binary diffing the MoonBounce UEFI component with the original firmware image from the vendor), there were no other modifications besides hooks for *InternalAllocatePool*, *CoreExitBootServices* and *CoreCreateEventInternal* services.

Matched Functions

Similarity	Confid	Change	EA Primary	Name Primary	EA Secondary	Name Secondary
0.95	0.99	GI--…	000000018001…	CoreCreateEventInternal	000000018001F10C	sub_000000018001F10C
0.99	0.99	-I--…	000000018000…	CoreExitBootServices	0000000180009F4C	sub_0000000180009F4C
0.99	0.99	-I--…	000000018002…	InternalAllocatePool	00000001800233B0	sub_00000001800233B0
1.00	0.99	----…	000000018000…	sub_18000030C	000000018000030C	sub_000000018000030C
1.00	0.99	----…	000000018000…	sub_180000358	0000000180000358	sub_0000000180000358
1.00	0.99	----…	000000018000…	sub_180000420	0000000180000420	sub_0000000180000420
1.00	0.99	----…	000000018000…	sub_18000045C	000000018000045C	sub_000000018000045C
1.00	0.99	----…	000000018000…	sub_1800004EC	00000001800004EC	sub_00000001800004EC
1.00	0.99	----…	000000018000…	sub_1800005D0	00000001800005D0	sub_00000001800005D0
1.00	0.99	----…	000000018000…	sub_180000624	0000000180000624	sub_0000000180000624
1.00	0.99	----…	000000018000…	sub_1800006D8	00000001800006D8	sub_00000001800006D8
1.00	0.99	----…	000000018000…	sub_180000730	0000000180000730	sub_0000000180000730
1.00	0.99	----…	000000018000…	sub_1800007A4	00000001800007A4	sub_00000001800007A4
1.00	0.99	----…	000000018000…	sub_1800007E4	00000001800007E4	sub_00000001800007E4
1.00	0.99	----…	000000018000…	sub_1800007FC	00000001800007FC	sub_00000001800007FC
1.00	0.99	----…	000000018000…	sub_180000910	0000000180000910	sub_0000000180000910
1.00	0.99	----…	000000018000…	sub_1800009E0	00000001800009E0	sub_00000001800009E0
1.00	0.99	----…	000000018000…	sub_180000A4C	0000000180000A4C	sub_0000000180000A4C
1.00	0.99	----…	000000018000…	sub_180000AD4	0000000180000AD4	sub_0000000180000AD4

BinDiff analysis with the original CORE_DXE {5AE3F37E-4EAE-41AE-8240-35465B5E81EB} driver from E7846IMS.M30 firmware

Using the Binarly Cloud Platform, we found a DXE Core driver which is very similar to the MoonBounce UEFI firmware dump component. The similarity was confirmed by matching control flow graphs through further analysis with Google BinDiff tool. The code similarity search reveals exactly three modified/hooked routines that we discussed earlier.

There are other interesting questions we need to discuss, such as potential methods of delivering malware to the target system. How could such malware be written into SPI flash storage of the targeted system? It is worth noticing that **the analyzed MoonBounce UEFI component was built for a target hardware related to a MSI system from 2014**. This fact allows us to suggest two possible initial points of compromise:

- **Physical access-based implant delivery** to the target system - no Intel Boot Guard technology present or enabled thus there are no physical or hardware restrictions to get access to SPI flash storage of the system.
- **Software-based implant delivery** to the target system - keeping in mind the historical ignorance of many vendors on firmware security threats, we infer that no SPI protections were enabled on the system, hence

SPI write-operations could be issued easily with no exploits required (access only to physical memory is required for working with PCH SPI controller MMIO).

Public information regarding firmware related implants is more present in the media lately, but it is just scratching the surface in terms of threat detection. Many security research papers have been published which provide examples of more advanced techniques for threat actors to persist in firmware. The supply chain complexity significantly increases the chances for the attackers to effectively reuse 1/N-day vulnerabilities (“The Firmware Supply-Chain Security is broken: Can we fix it?”.

We need to increase the industry awareness to firmware related threats and build more effective threat hunting programs with cross-industry collaboration between the vendors to mutually benefit customers and provide better detection rates.
