LEGBACORE

WE DO DIGITAL VOODOO

BIOS Necromancy: Utilizing "Dead Code" for BIOS Attacks

Corey Kallenberg & Xeno Kovah

October 14, 2015

Contents

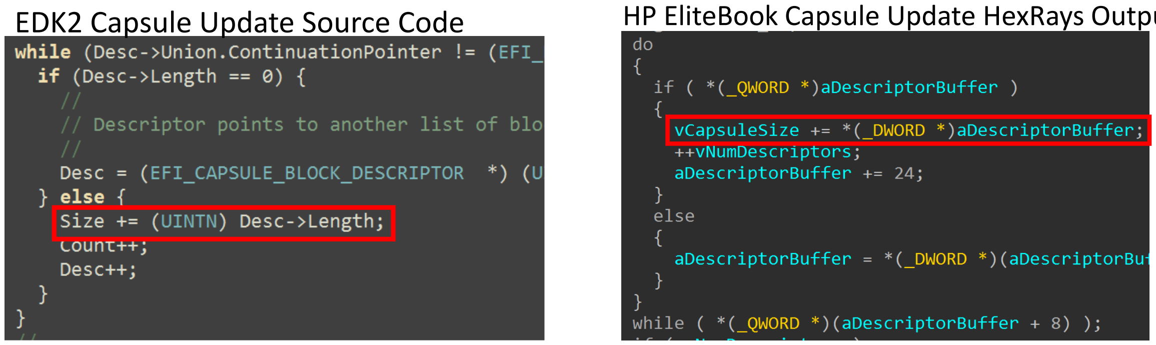
1 Introduction

The transition from legacy BIOS to UEFI firmware has brought with it a growing interest in firmware level attacks. At BlackHat USA 2015 alone, several talks [7][5][4] were dedicated specifically to attacks at this level. The recent NSA and Hacking Team leaks have also underscored the fact that PC firmware attacks are now used in the wild[1][9]. To further exploration of this security critical domain, this paper uses an old vulnerability to bring to light a new problematic area of the UEFI ecosystem.

2 VU #552286

In 2013 several memory corruption vulnerabilities[8] were identified in the UEFI reference implementation[3] firmware update mechanism. CVE-2014-4860 ("Queen's Gambit") describes a buffer overflow in firmware update fragment coalescing in the PEI phase. CVE-2014-4859 ("King's Gambit") describes a buffer overflow in firmware update header processing in the DXE phase. Both of these vulnerabilities can be leveraged by an attacker to reflash the firmware with an unsigned image, despite the presence of signed firmware update enforcement.

Because these vulnerabilities were initially located in reference implementation code, as opposed to in an actual vendor firmware, it was unclear to what degree the vulnerabilities would be present "in the wild." As each OEM is free to implement their own firmware update scheme, the only options for discovering which OEMs the vulnerabilities impacted were to either reverse engineer each OEMs firmware, or rely on the OEMs to self report.


EDK2 Capsule Update Source Code
HP EliteBook Capsule Update HexRays Output

```
while (Desc->Union.ContinuationPointer != (EFI_
  if (Desc->Length == 0) {
    //
    // Descriptor points to another list of blo
    //
    Desc = (EFI_CAPSULE_BLOCK_DESCRIPTOR  *) (U
  } else {
    Size += (UINTN) Desc->Length;
    Count++;
    Desc++;
  }
}
```

```
do
{
  if ( *(_QWORD *)aDescriptorBuffer )
  {
    vCapsuleSize += *(_DWORD *)aDescriptorBuffer;
    ++vNumDescriptors;
    aDescriptorBuffer += 24;
  }
  else
  {
    aDescriptorBuffer = *(_DWORD *)(aDescriptorBu
  }
}
while ( *(_QWORD *)(aDescriptorBuffer + 8) );
```

Figure 1: CVE-2014-4860 confirmed in HP EliteBook 2540p

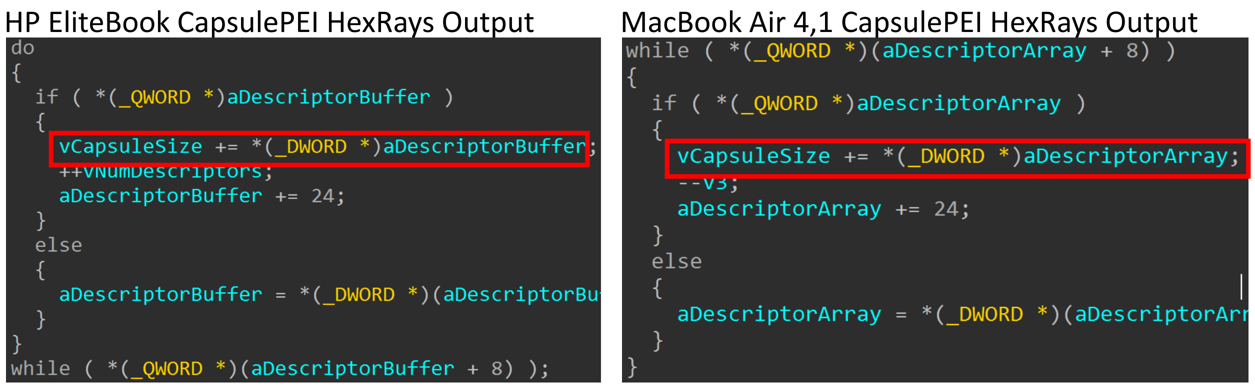
A relatively straight forward way to attempt to detect the presence of the CVE-2014-4860 vulnerability in an OEM firmware is to use UEFITool[10] and attempt to locate and extract the CapsulePEI module. Typically this module is assigned guid {C779F6D8-7113-4AA1-9648-EB1633C7D53B}[1]. Then, one can compile the edk2 CapsulePEI module with full debug symbols. Next, a binary diffing application[2] can be used to compare the edk2 CapsulePEI with the OEM CapsulePEI. Typically this yields strong matches on several functions, which allows for easy identification of the relevant functions (GetCapsuleInfo in this case) through cross referencing. Finally, use of a disassembly or decompilation tool allows for rapid confirmation as to whether or not the vulnerability exists. Figure 1 shows the result of this process against an HP EliteBook 2540p which resulted in a positive confirmation for the existence of CVE-2014-4860.

Although this confirmation process only takes approximately 1 hour, it was not deemed worth the time to repeat this process across all OEMs and all of their individual models (hundreds in total), each of which may have been using a customized firmware update routine. Thus ultimately the authors relied on vendors to self report whether or not they were vulnerable. While many vendors confirmed they were vulnerable[2], several indicated they were not vulnerable because they had "rolled their own" firmware update mechanism. This seemed like a reasonable response and so originally the authors did not investigate vendors who claimed this.

[1]though this is not necessarily the case on all systems
[2]e.g. Zynamics BinDiff http://www.zynamics.com/bindiff.html

3 Raising the dead

While investigating Mac firmware for known PC firmware attacks[7] the authors coincidentally noticed the presence of a PEI module with the CapsulePEI guid {C779F6D8-7113-4AA1-9648-EB1633C7D53B}. Apple had been one of the vendors that originally claimed they were not vulnerable to CVE-2014-4860 or CVE-2014-4859, giving the common reason that they used their own custom firmware update mechanism and hence were not vulnerable to the vulnerabilities contained in the reference implementation. The authors already knew this to be the case, based on the description of the Apple firmware update process given in the original Thunderstrike talk[6].


HP EliteBook CapsulePEI HexRays Output
MacBook Air 4,1 CapsulePEI HexRays Output

```
do
{
  if ( *(_QWORD *)aDescriptorBuffer )
  {
    vCapsuleSize += *(_DWORD *)aDescriptorBuffer;
    ++vNumDescriptors;
    aDescriptorBuffer += 24;
  }
  else
  {
    aDescriptorBuffer = *(_DWORD *)(aDescriptorBu
  }
}
while ( *(_QWORD *)(aDescriptorBuffer + 8) );
```

```
while ( *(_QWORD *)(aDescriptorArray + 8) )
{
  if ( *(_QWORD *)aDescriptorArray )
  {
    vCapsuleSize += *(_DWORD *)aDescriptorArray;
    --v3;
    aDescriptorArray += 24;
  }
  else
  {
    aDescriptorArray = *(_DWORD *)(aDescriptorArr
  }
}
```

Figure 2: CVE-2014-4860 identified in MacBook Air 4,1

Repeating the procedure described in Section 2, the presence of CVE-2014-4860 was confirmed in a MacBook Air. However, there remained the possibility that this was uninvokable dead code, and hence the platform would still be not vulnerable. The lack of a BIOS debug capability for this platform made confirming the reachability of this code path non-trivial. Although tedious, eventually the authors were able to hack in some basic debugging capability by reprogramming the CapsulePEI module on the SPI flash with an external flash programmer. Then, the authors were able to confirm the reachability of the vulnerable code path and develop a proof of concept to instantiate the vulnerability on the platform's release firmware. The approximate process to invoke the vulnerability on the MacBook Air is described below.

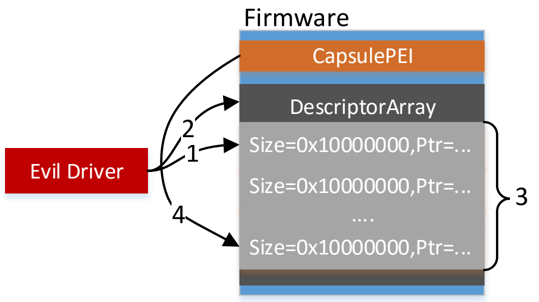


Figure 3: Instantiating CVE-2014-4860 on MacBook Air 4,1

1 A malicious kernel driver stages an evil capsule descriptor array in memory.

2 The malicious kernel driver creates the EFI variable "CapsuleUpdateData." The contents of this variable are a pointer to the evil capsule descriptor array.

3 The sum of the evil capsule descriptor array size elements overflows the total capsule size variable, leading to an undersized capsule reconstruction memory area.

4 CapsulePEI notices the presence of "CapsuleUpdateData" and begins the EDK2 firmware update process. Memory corruption occurs when the capsule fragments are copied into the undersized capsule reconstruction area.

4 Mitigation And Discussion

We refer to this technique as "necromancy" because we are resurrecting dead code[3] that exists on the SPI flash[4] purely for the purposes of exploitation. Under normal circumstances this code should never be invoked. This leads to a situation where the BIOS developers may look at some piece of code where a vulnerability is found and think "we don't invoke this ourselves, *so the vulnerability is uninvokable.*", which ends up being a deadly assumption. Hence, there are actually 2 unique firmware update code paths that can be invoked. This effectively doubles the attack surface against the platform's firmware. Although our case study in Section 3 focuses on a specific Apple platform, we know this issue is not unique to Apple and is instead a general problem with the UEFI ecosystem.

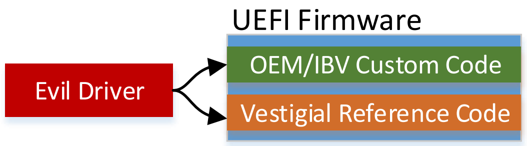


Figure 4: Doubled attack surface

Firmware developers will often opt to replace parts of the reference implementation with their own custom code paths for "value-add" purposes. However, unless they explicitly evict the original reference implementation code from their firmware, that code will remain invokable by an attacker and hence increases the attack surface against the firmware. The obvious mitigation against this unnecessary attack surface increase is to identify and remove vestigial code in the firmware. Unfortunately we believe this task is non-trivial because identifying code paths that should never be called under any legitimate circumstances can be difficult, unless the BIOS developers have very robust unit tests and QA processes. If sufficient tests are not available, the common practice for firmware developers is to err on the side of caution, and not remove any code. Because the worst case penalty for removing code that may be necessary under *some* circumstances is very tangible: a bricked platform. The reward for successfully removing vestigial code is much less tangible: decreased attack surface. Although as security researchers we encourage firmware developers to attempt to identify and evict vestigial code, we recognize it is a non-trivial task.

5 Vendor Response

For the specific issue considered in this paper, Apple's vulnerability to CVE-2014-4860, Apple has indicated that it has "made modifications to EFI to protect against running unused functions" and that a patch for this issue will be forthcoming in the OS X 10.11.1 security update.

References

[1] J. Appelaum, J. Horchert, O. Reissman, M. Rosenbach, J. Schindler, and C. Stocker. NSA's secret toolbox: Unit offers spy gadgets for every need. `http://www.spiegel.de/international/world/nsa-secret-toolbox-ant-unit-offers-spy-gadgets-for-every-need-a-941006.html`. Accessed: 6/01/2015.

[3]To be clear, we do not mean dead code, in the sense that a compiler could automatically optimize the code out. Rather, we mean it in the sense that the UEFI compile system is relatively complex, and there exist opportunities for code to be built and may be placed into the firmware filesystem, but never actually be looked up or invoked under normal circumstances.

[4]In this case, the referencing implementation capsule coalescing code.

[2] CERT. VU #552286. http://www.kb.cert.org/vuls/id/552286. Accessed: 10/13/2015.

[3] Intel Corporation. UEFI Development Kit 2010. http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=UDK2010. Accessed: 06/13/2014.

[4] C. Domas. The memory sinkhole - unleashing an x86 design flaw allowing universal privilege escalation. In *BlackHat*, Las Vegas, USA, 2015.

[5] M. Gorobets, O. Bazhaniuk, A. Matrosov, A. Furtak, and Y. Bulygin. Attacking hypervisors using firmware and hardware. In *BlackHat*, Las Vegas, USA, 2015.

[6] T. Hudson. Thunderstrike. https://trmm.net/Thunderstrike. Accessed: 6/01/2015.

[7] T. Hudson, X. Kovah, and C. Kallenberg. Thunderstrike 2: Sith strike. In *BlackHat*, Las Vegas, USA, 2015.

[8] C. Kallenberg, X. Kovah, J. Butterworth, and S. Cornwell. Extreme privilege escalation on windows 8/uefi systems. In *BlackHat*, Las Vegas, USA, 2014.

[9] Intel Advanced Threat Research. Hacking team's bad bios: A commercial rootkit for uefi firmware? http://www.intelsecurity.com/advanced-threat-research/ht_uefi_rootkit.html_7142015.html. Accessed: 10/13/2015.

[10] Nikolaj Schlej. Uefitool source code. https://github.com/LongSoft/UEFITool. Accessed: 6/01/2015.