Extreme Privilege Escalation on Windows 8/UEFI Systems

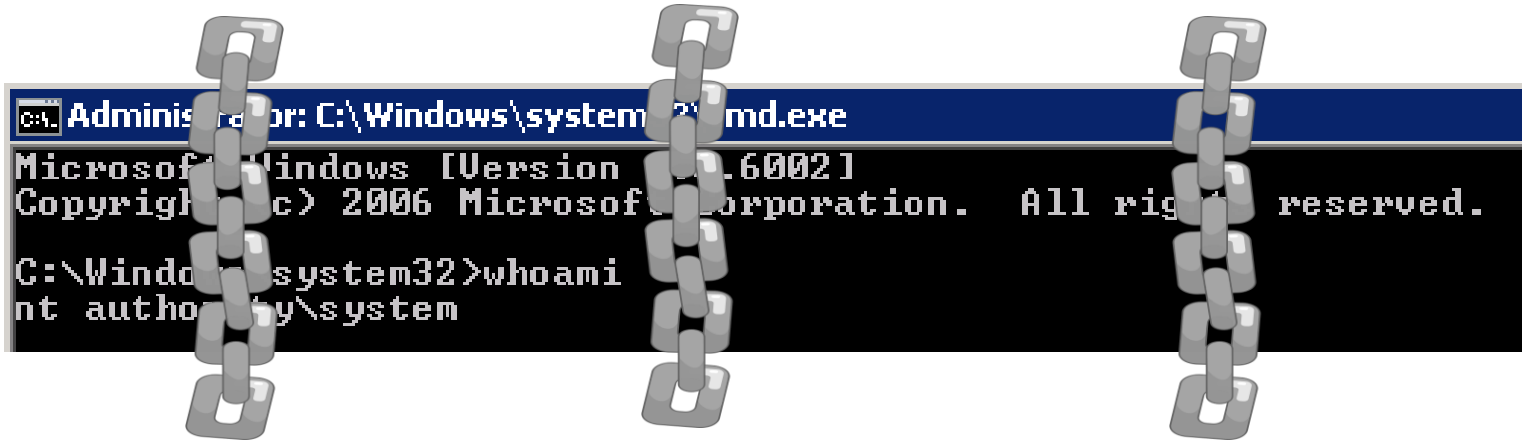
Corey Kallenberg	@coreykal
Xeno Kovah	@xenokovah
John Butterworth	@jwbutterworth3
Sam Cornwell	@ssc0rnwell

MITRE

Focused Presentation Goals

- **Offensive people:**
  - Highlight an attack surface worth exploring
  - Describe a versatile rootkit proof of concept
- **Defensive people:**
  - Highlight an attacker vector to be aware of
  - Provide tools and insight to help you

Attack Model (1 of 2)

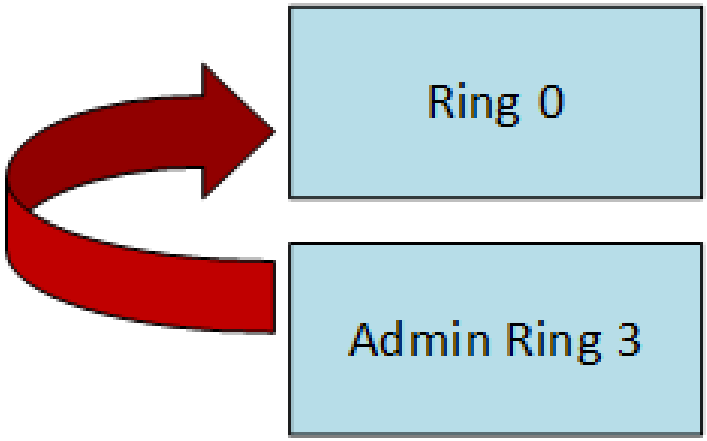


- **An attacker has gained administrator access on a victim Windows 8 machine**
- **But they are still constrained by the limits of ring 3**

Attack Model (2 of 2)

- **Attackers always want**
  - More Power
  - More Persistence
  - More Stealth

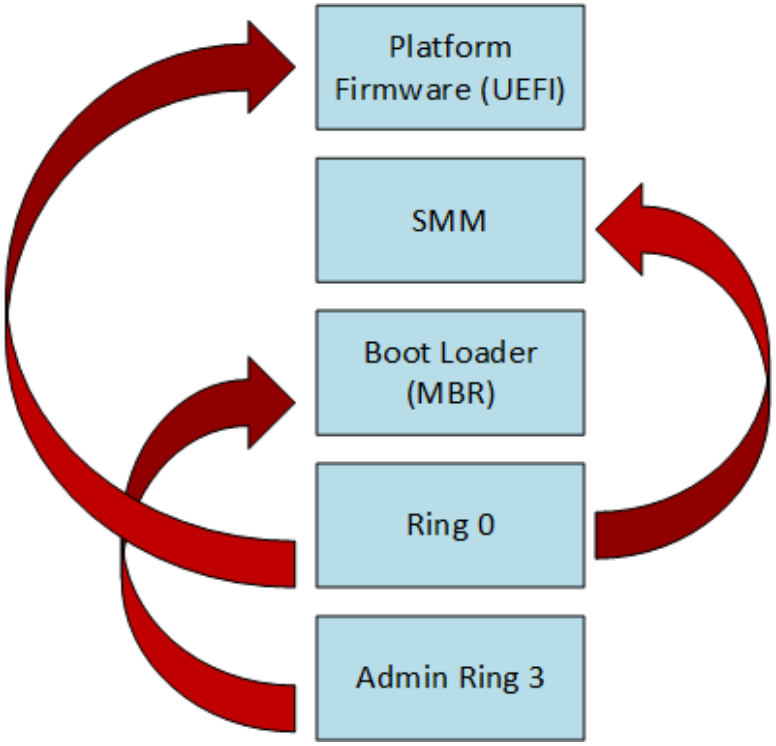
Typical Post-Exploitation Privilege Escalation



- **Starting with x64 Windows vista, kernel drivers must be signed and contain an Authenticode certificate**
- **In a typical post-exploitation privilege escalation, the attacker wants to bypass the signed driver requirement to install a kernel level rootkit**
- **Various methods to achieve this are possible, including:**
  - Exploit existing kernel drivers
  - Install a legitimate (signed), but vulnerable, driver and exploit it
- **This style of privilege escalation has been well explored by other researchers such as [6][7].**
- **There are other, more *extreme,* lands the attacker may wish to explore**

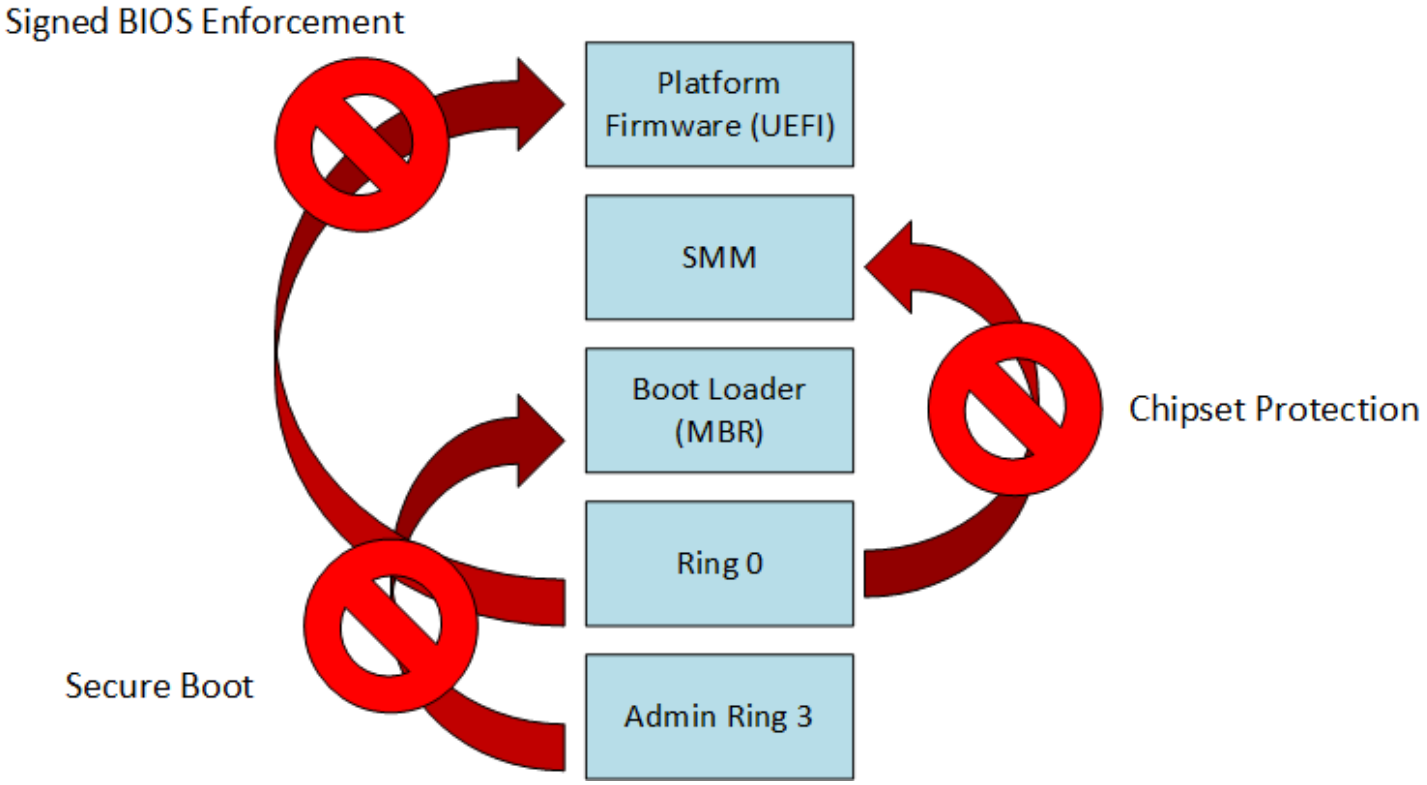
MITRE

Other Escalation Options (1 of 2)



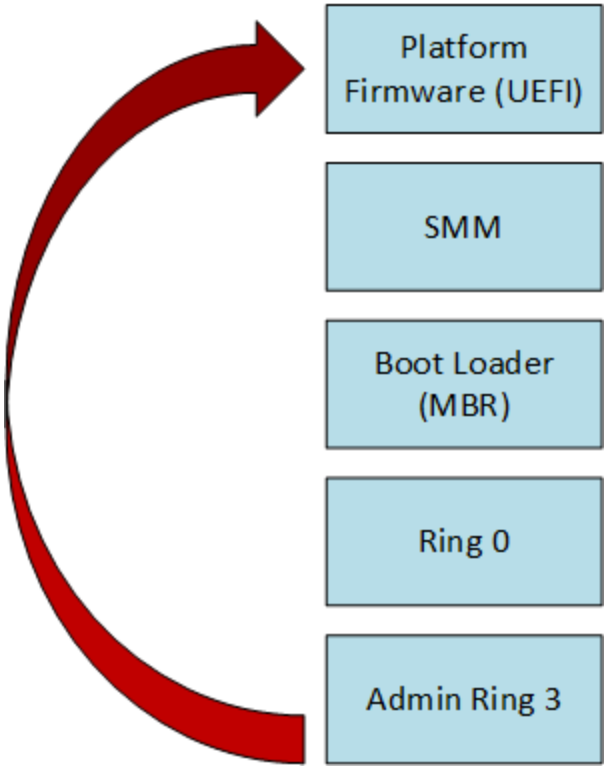
- **There are other more interesting post-exploitation options an attacker may consider:**
  - Bootkit the system
  - Install SMM rootkit
  - Install BIOS rootkit

Other Escalation Options (2 of 2)



- **Modern platforms contain protections against these more exotic post-exploitation privilege-escalations**
  - Bootkit the system (Prevented by Secure Boot)
  - Install SMM rootkit (SMM is locked on modern systems)
  - Install BIOS rootkit (SPI Flash protected by lockdown mechanisms)

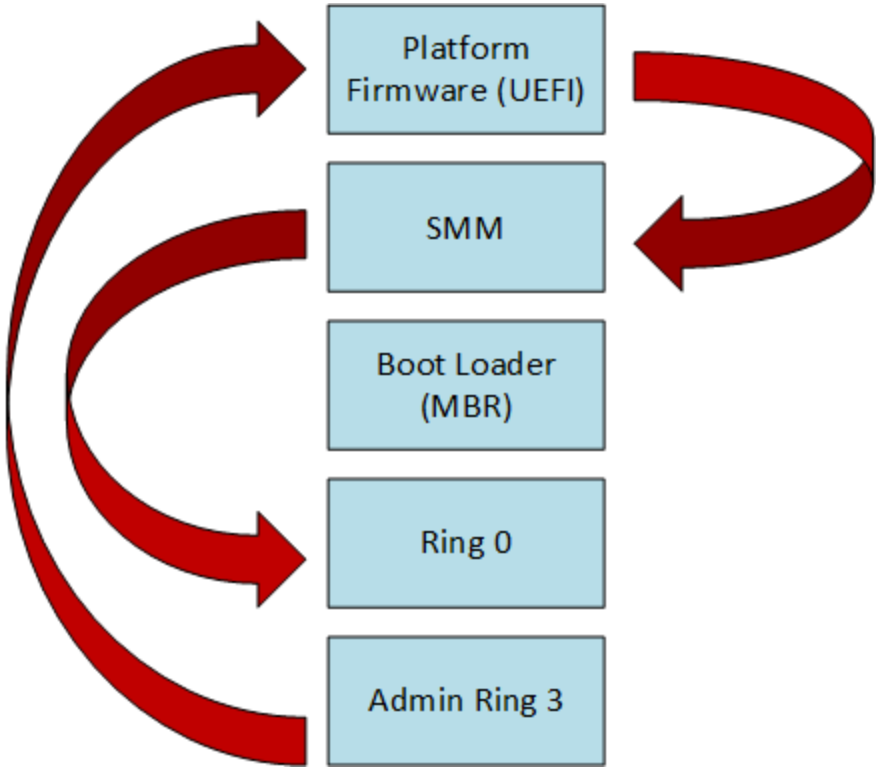
Extreme Privilege Escalation (1 of 2)



- **This talk presents *extreme* privilege escalation**
  - Administrator userland process exploits the platform firmware (UEFI)
  - Exploit achieved by means of a new API introduced in Windows 8

MITRE

Extreme Privilege Escalation (2 of 2)

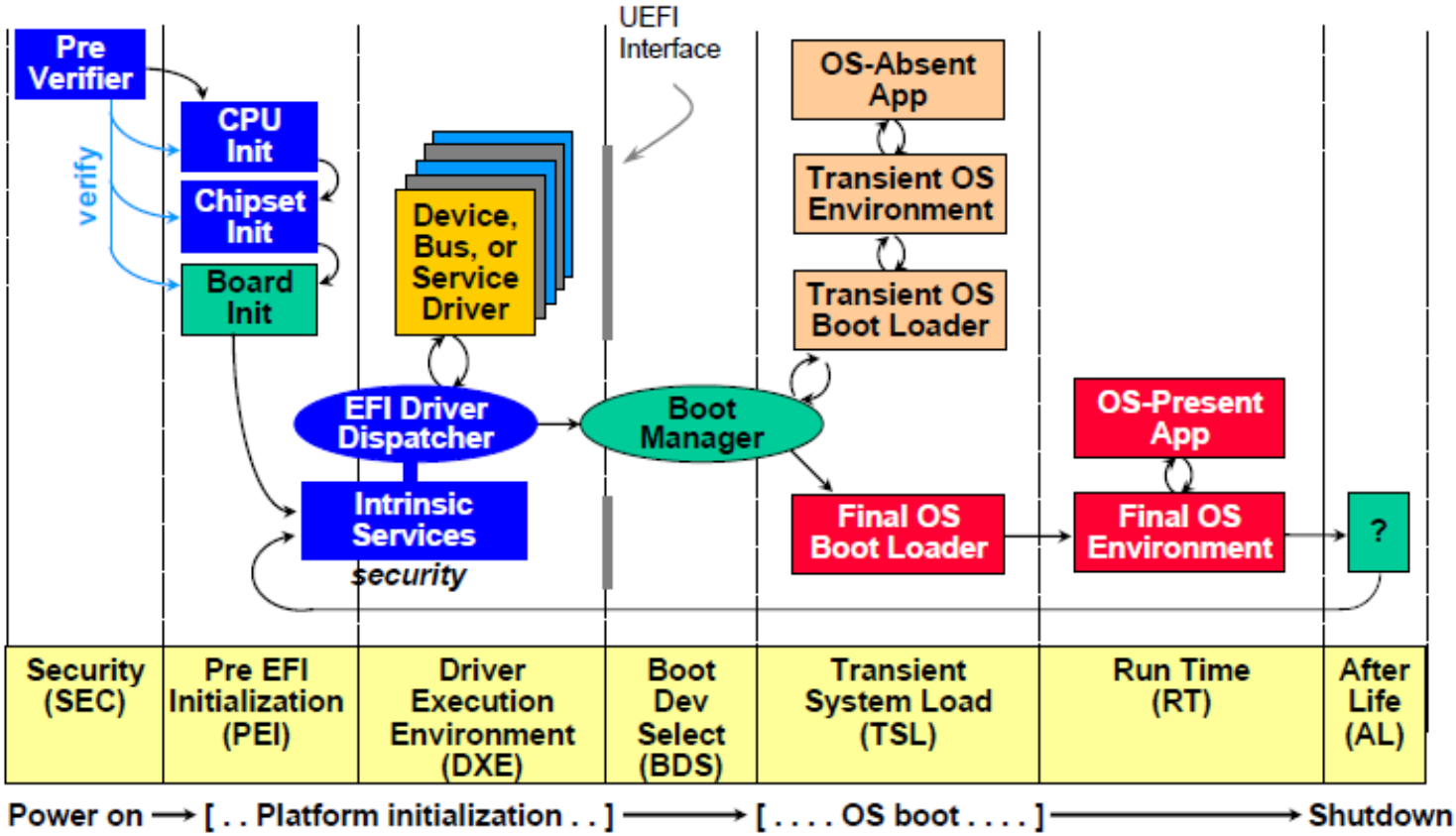


- **Once the attacker has arbitrary code execution in the context of the platform firmware, he is able to:**
  - Control other "rings" on the platform (SMM, Ring 0)
  - Persist beyond operating system re-installations
  - Permanently "brick" the victim computer

Target Of Attack

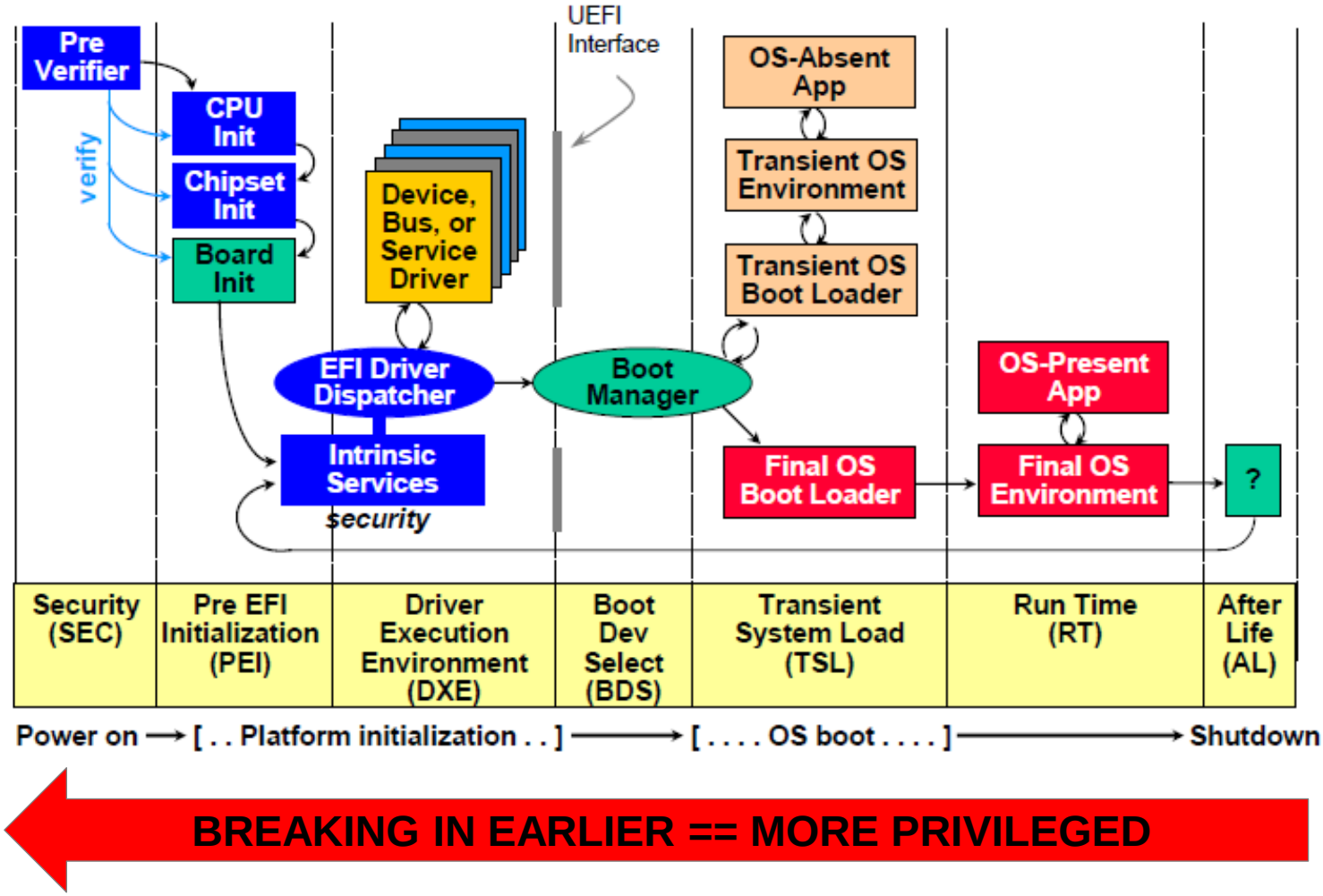
- **Modern Windows 8 systems ship with UEFI firmware**
- **UEFI is designed to replace conventional BIOS and provides a well defined interface to the operating system**

UEFI Purpose



- **Initialize hardware**
  - Configure and lock security relevant parts of the hardware
- **Find and transfer control to OS**

Attacking UEFI



Windows 8 API

SetFirmwareEnvironmentVariable function

Sets the value of the specified firmware environment variable.

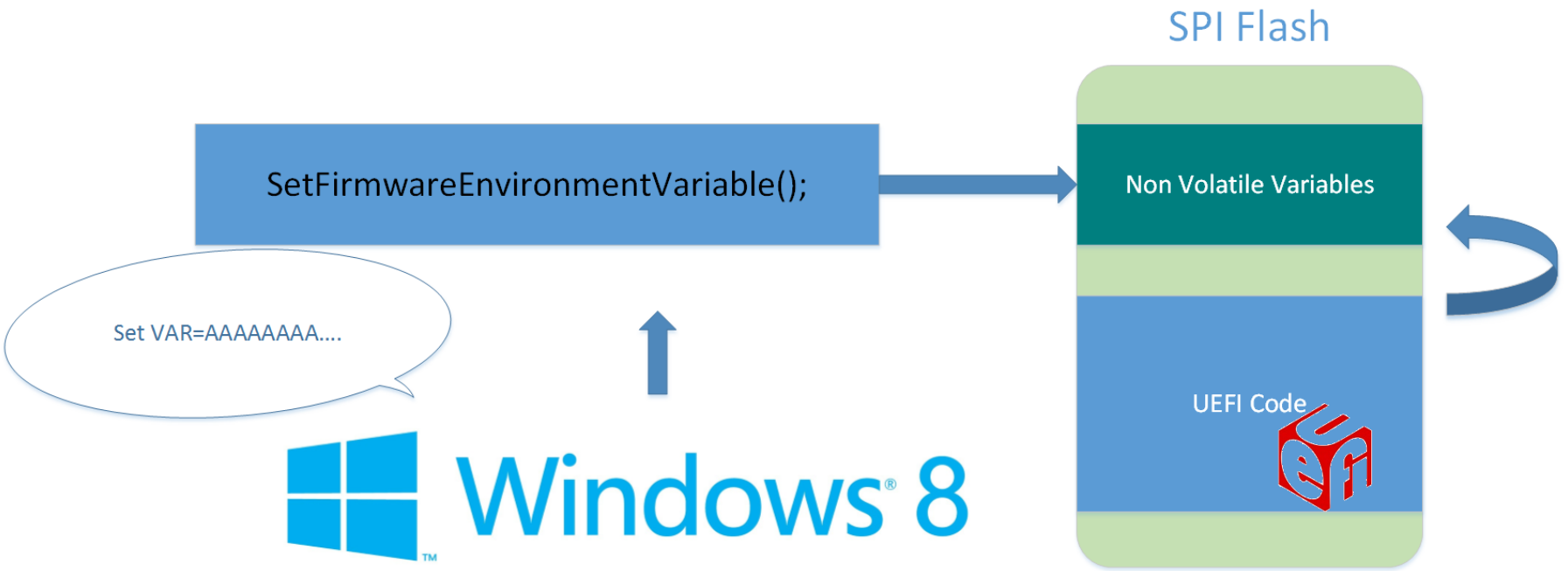
Syntax

C++

```
BOOL WINAPI SetFirmwareEnvironmentVariable(
  _In_  LPCTSTR lpName,
  _In_  LPCTSTR lpGuid,
  _In_  PVOID pBuffer,
  _In_  DWORD nSize
);
```

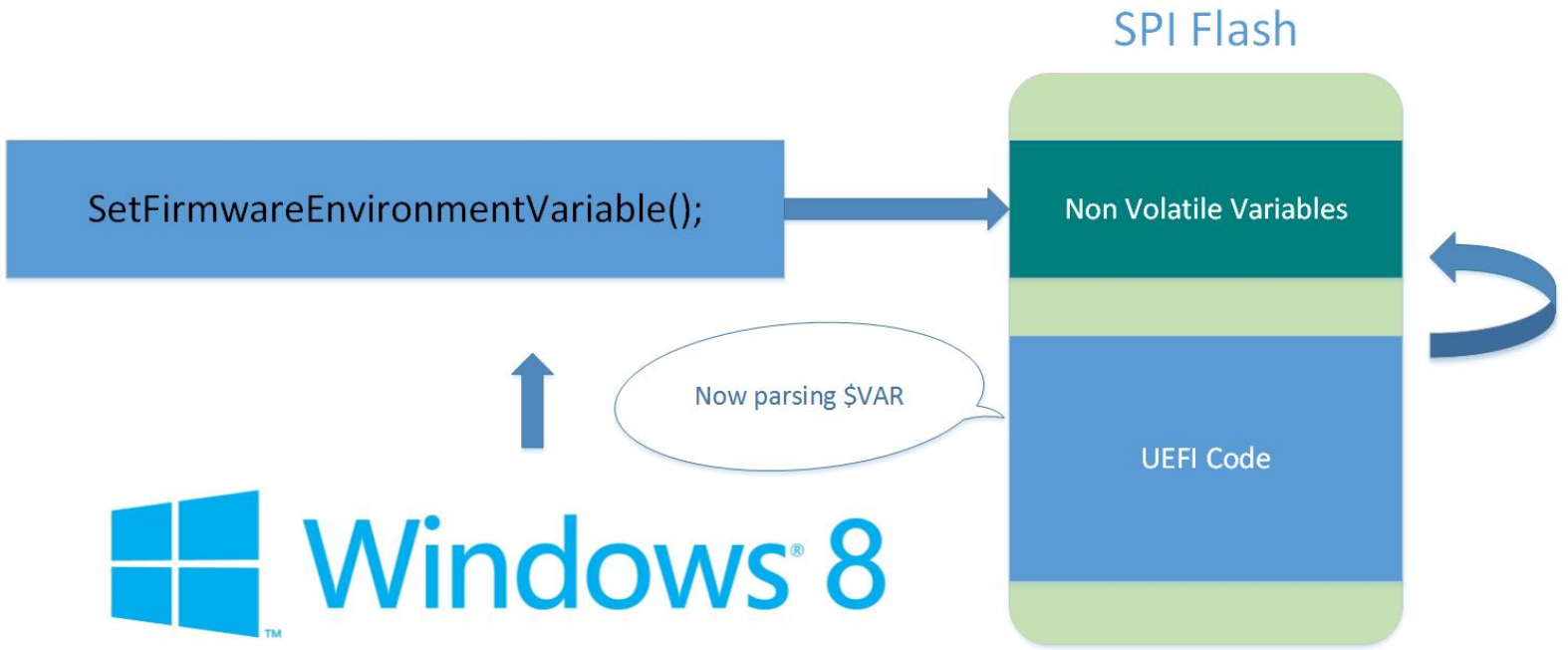
- **Windows 8 has introduced an API that allows a privileged userland process to interface with a subset of the UEFI interface**

EFI Variable Creation Flow



- **Certain EFI variables can be created/modified/deleted by the operating system**
  - For example, variables that control the boot order and platform language
- **The firmware can also use EFI variables to communicate information to the operating system**

EFI Variable Consumption



- **The UEFI variable interface is a conduit by which a less privileged entity (admin Ring 3) can produce data for a more complicated entity (the firmware) to consume**
- **This is roughly similar to environment variable parsing attack surface on *nix systems**

Previous EFI Variable Issues (1 of 2)

Vulnerability Note VU#758382

Unauthorized modification of UEFI variables in UEFI systems

Original Release date: 09 Jun 2014 | Last revised: 19 Jun 2014

Print Tweet Send Share

Overview

Certain firmware implementations may not correctly protect and validate information contained in certain UEFI variables. Exploitation of such vulnerabilities could potentially lead to bypass of security features and/or denial of service for the platform.

Description

As discussed in recent conference publications (CanSecWest 2014, Syscan 2014, and Hack-in-the-Box 2014) certain UEFI implementations do not correctly protect and validate information contained in the 'Setup' UEFI variable. On some systems, this variable can be overwritten using operating system APIs. Exploitation of this vulnerability could potentially lead to bypass of security features, such as secure boot, and/or denial of service for the platform. Please refer to the conference publications for further details.

Impact

A local attacker that obtains administrator access to the operating system may be able to modify UEFI variables. Exploitation of such vulnerabilities could potentially lead to bypass of security features and/or denial of service for the platform.

- **We've already co-discovered[13] with Intel some vulnerabilities associated with EFI Variables that allowed bypassing secure boot and/or bricking the platform**

MITRE

Previous EFI Variable Issues (2 of 2)

Vulnerability Note VU#758382

Unauthorized modification of UEFI variables in UEFI systems

Original Release date: 09 Jun 2014 | Last revised: 19 Jun 2014

Print Tweet Send Share

Overview

Certain firmware implementations may not correctly protect and validate information contained in certain UEFI variables. Exploitation of such vulnerabilities could potentially lead to bypass of security features and/or denial of service for the platform.

Description

As discussed in recent conference publications (CanSecWest 2014, Syscan 2014, and Hack-in-the-Box 2014) certain UEFI implementations do not correctly protect and validate information contained in the 'Setup' UEFI variable. On some systems, this variable can be overwritten using operating system APIs. Exploitation of this vulnerability could potentially lead to bypass of security features, such as secure boot, and/or denial of service for the platform. Please refer to the conference publications for further details.

Impact

A local attacker that obtains administrator access to the operating system may be able to modify UEFI variables. Exploitation of such vulnerabilities could potentially lead to bypass of security features and/or denial of service for the platform.

- **However, VU #758382 was leveraging a proprietary Independent BIOS Vendor (IBV) implementation mistake, it would be more devastating if an attacker found a variable vulnerability more generic to UEFI**

MITRE

UEFI Vulnerability Proliferation

UEFI

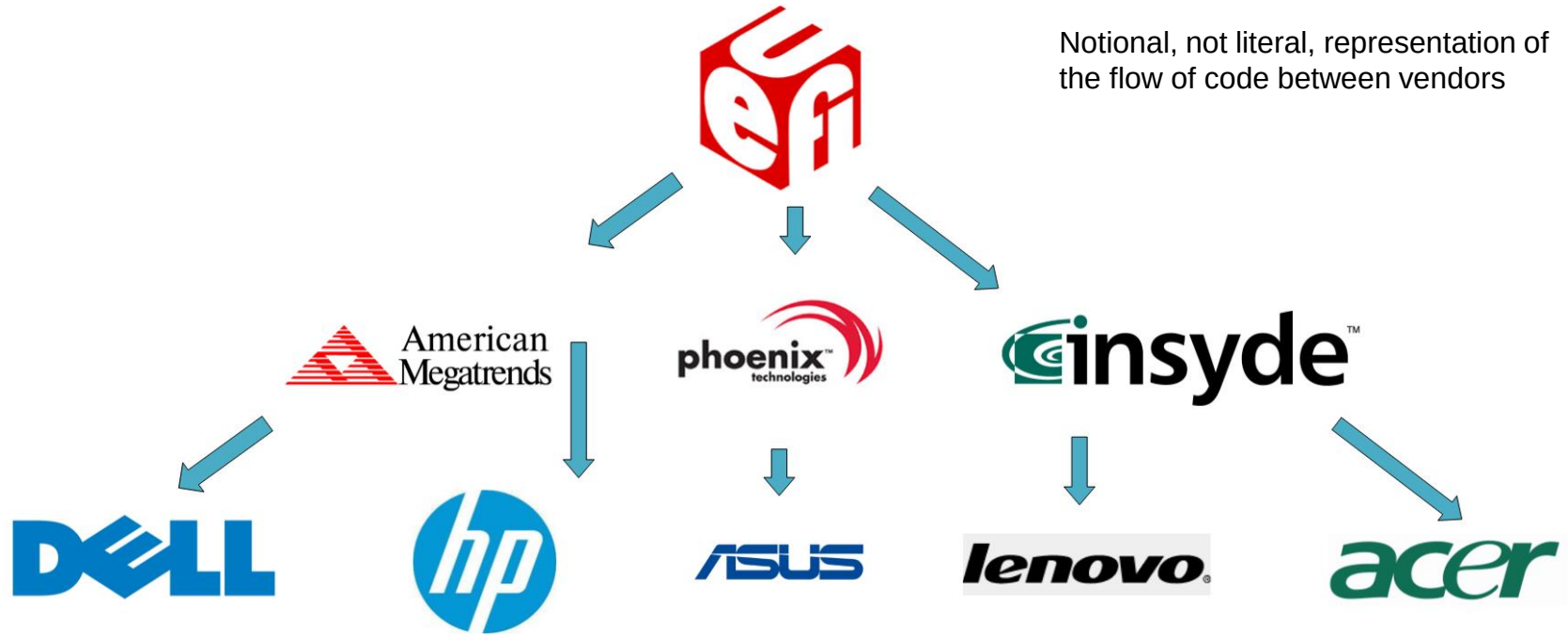
(Unified Extensible Firmware Interface)

IBVs

(Independent BIOS Vendors)

OEMs

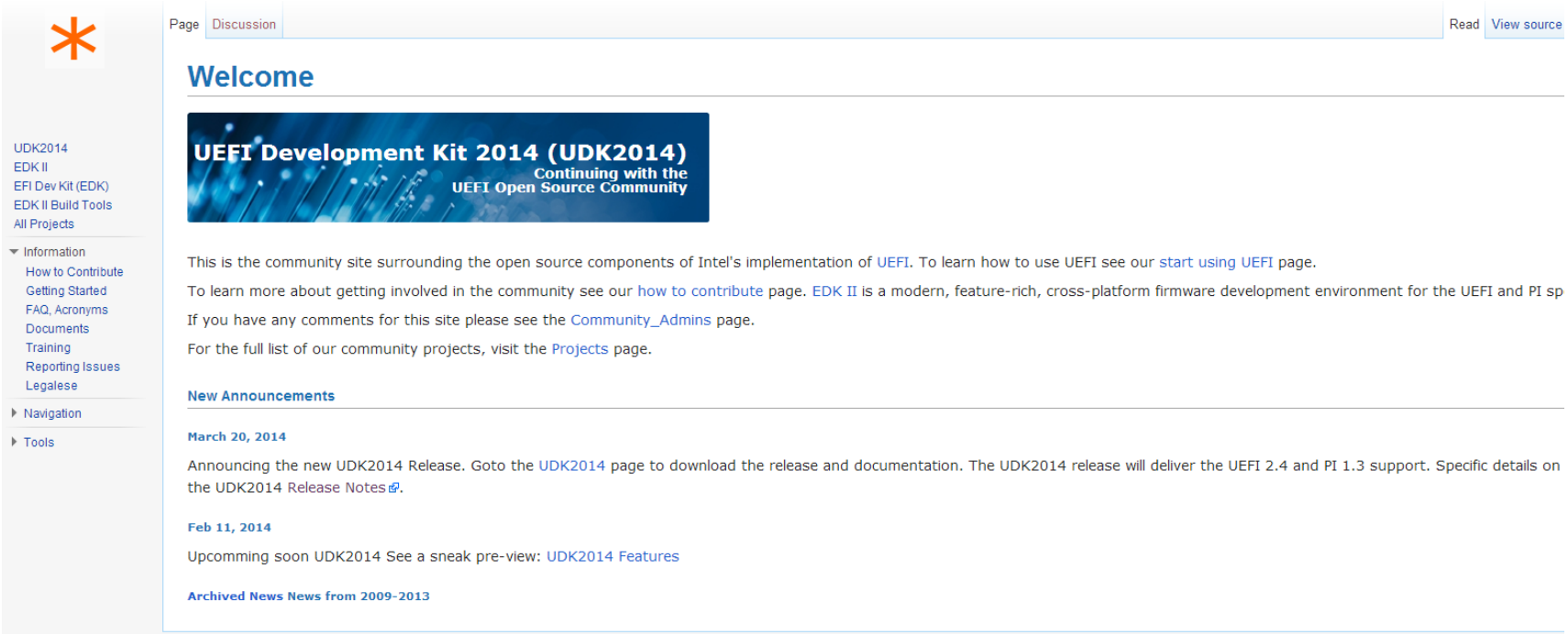
(Original equipment manufacturers)



Notional, not literal, representation of the flow of code between vendors

- **If an attacker finds a vulnerability in the UEFI "reference implementation," its proliferation across IBVs and OEMs would potentially be wide spread.**

Auditing UEFI



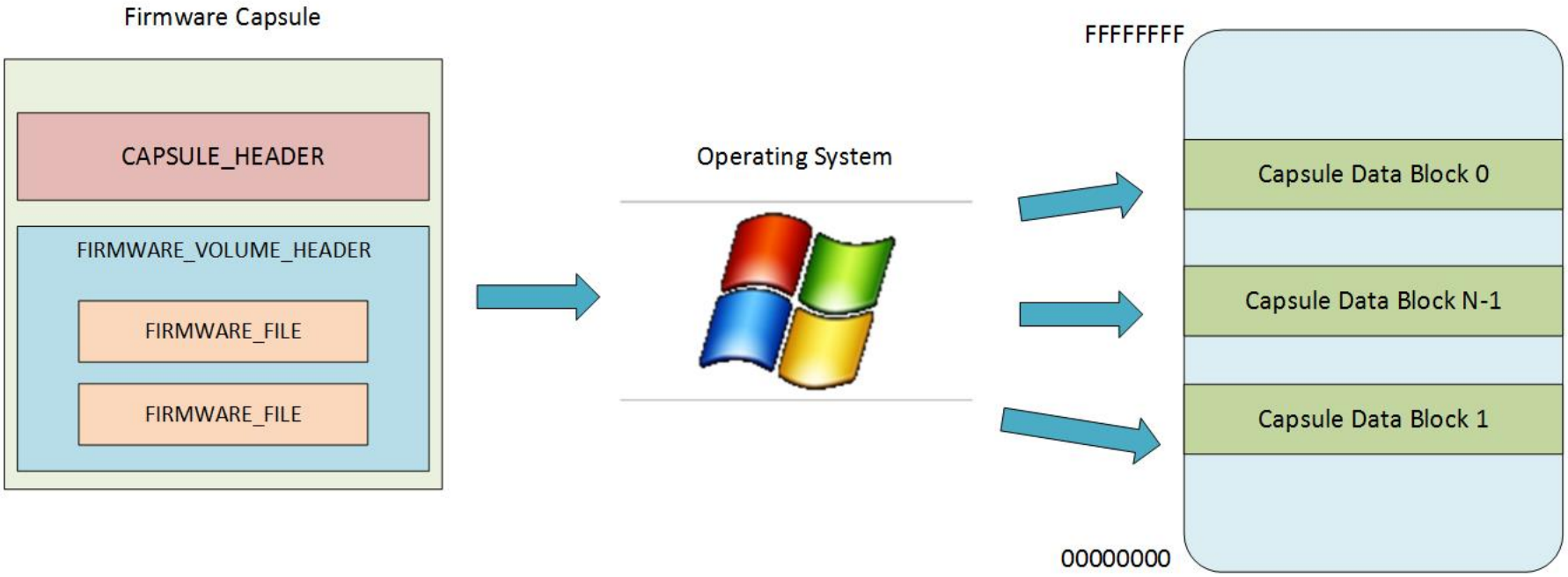
http://tianocore.sourceforge.net/wiki/Welcome

- **UEFI reference implementation is open source, making it easy to audit**
- **Let the games begin:**
  - Svn checkout https://svn.code.sf.net/p/edk2/code/trunk/edk2/

Where to Start Looking for Problems?

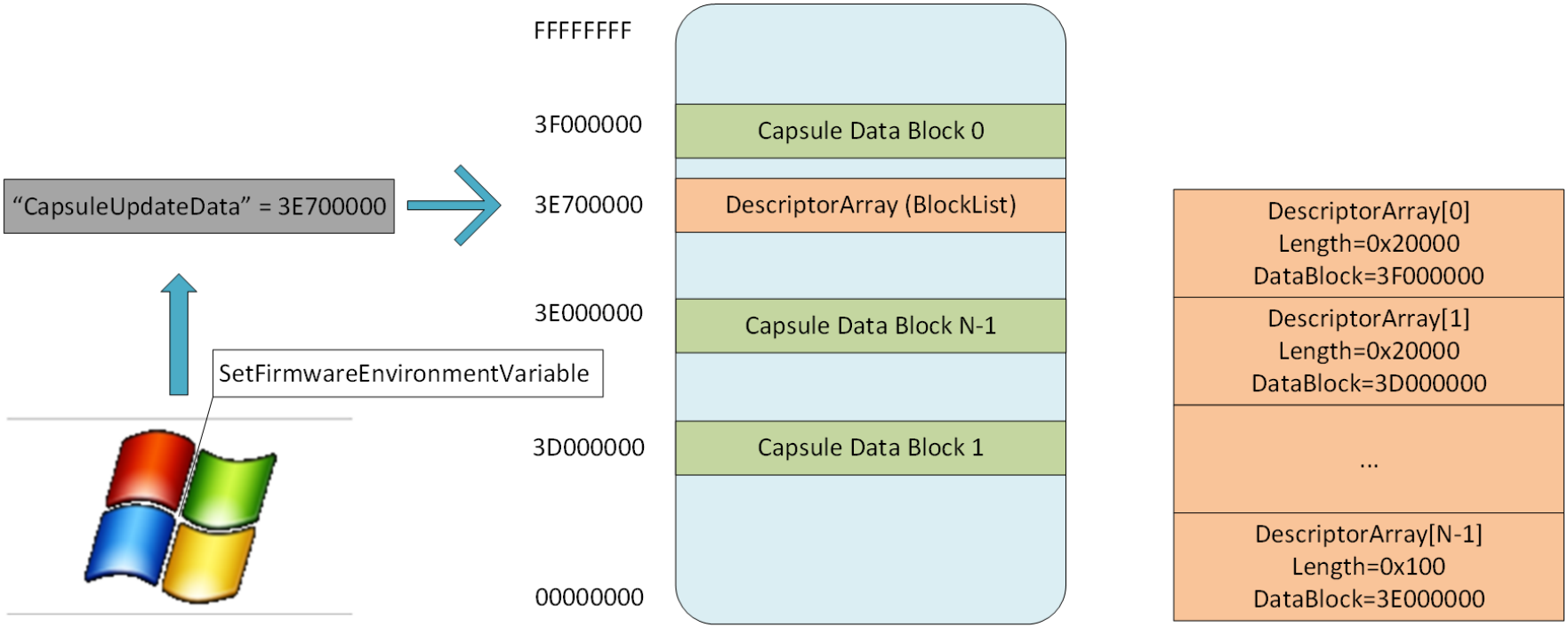
- **Always start with wherever there is attacker-controlled input**
  - Many of the UEFI variables are writeable by the OS, and are thus “attacker controlled”
- **We had good success last year exploiting Dell systems by passing an specially-crafted fake BIOS update…**
- **The UEFI spec outlines a "Capsule update" mechanism for firmware updates**
  - It’s not directly callable by ring 3 code…
  - But it can be initiated by the creation of a special EFI Variable!
  - We considered this to be a good target

Capsule Scatter Write



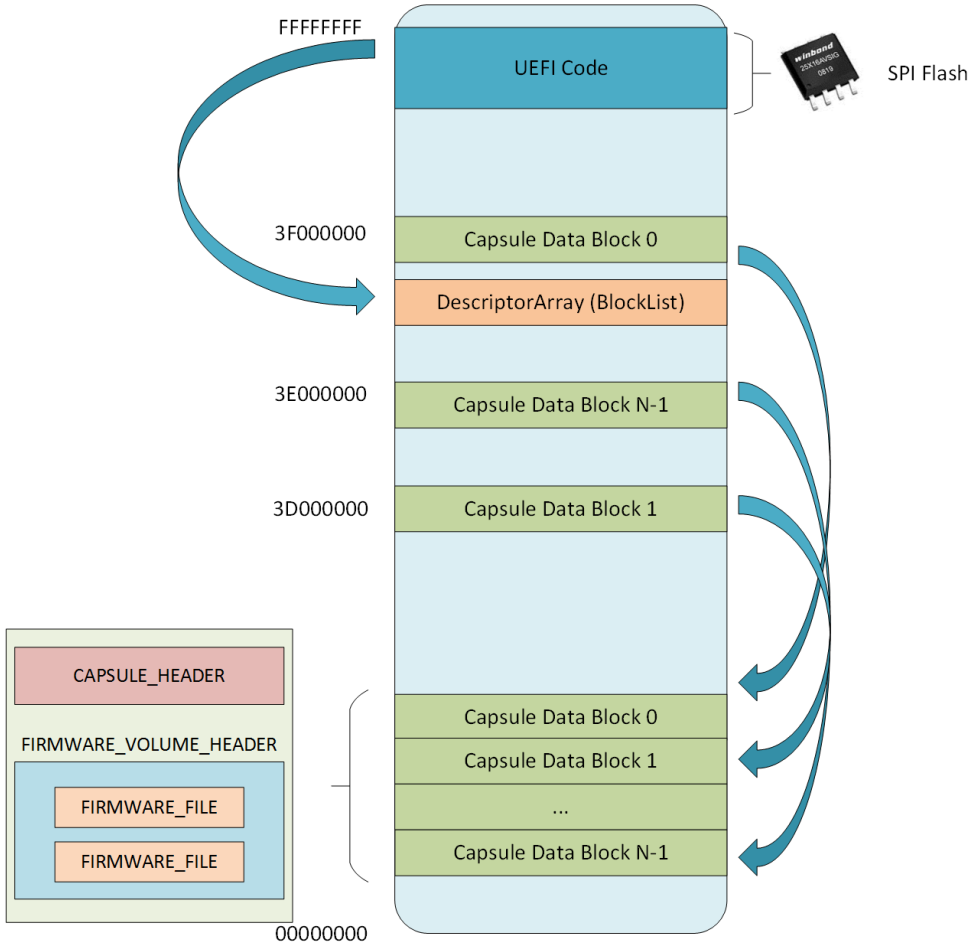
- **To begin the process of sending a Capsule update for processing, the operating system takes a firmware capsule and fragments it across the address space**

Capsule Processing Initiation



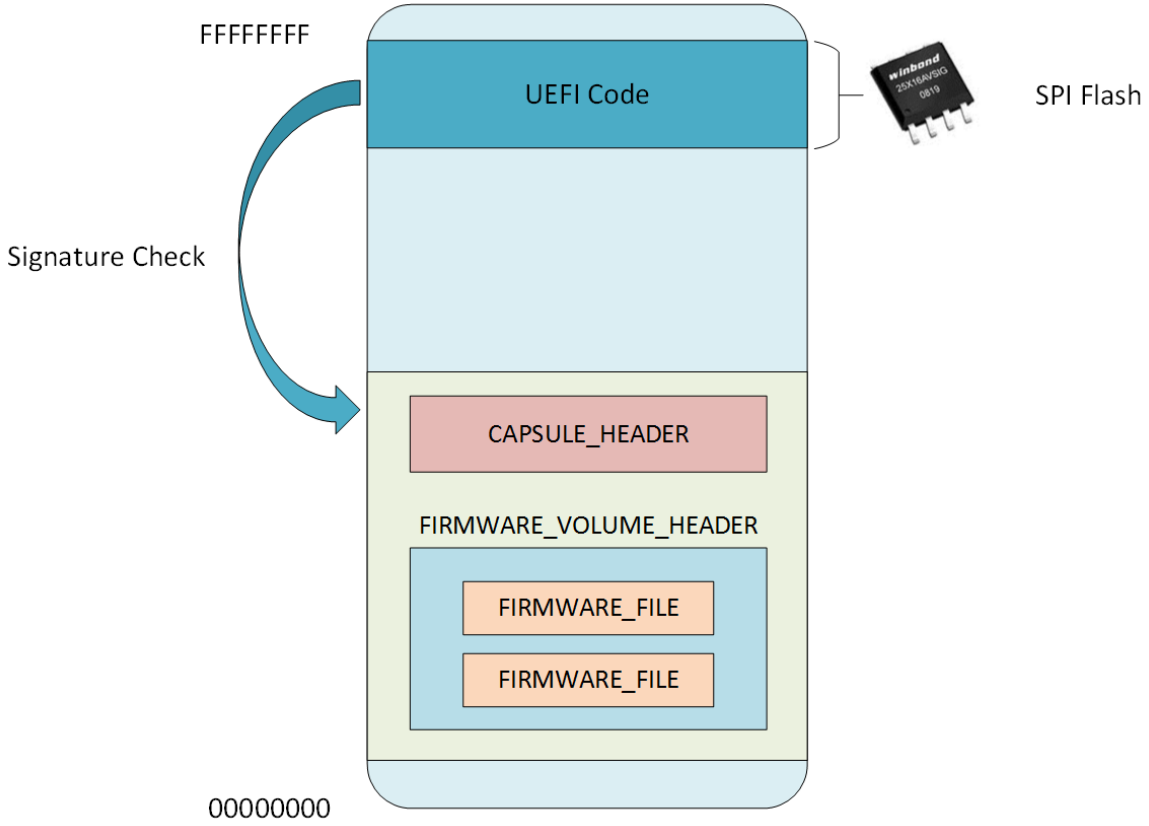
- **The operating system creates an EFI variable that describes the location of the fragmented firmware capsule**
- **A "warm reset" then occurs to transition control back to the firmware**

Capsule Coalescing



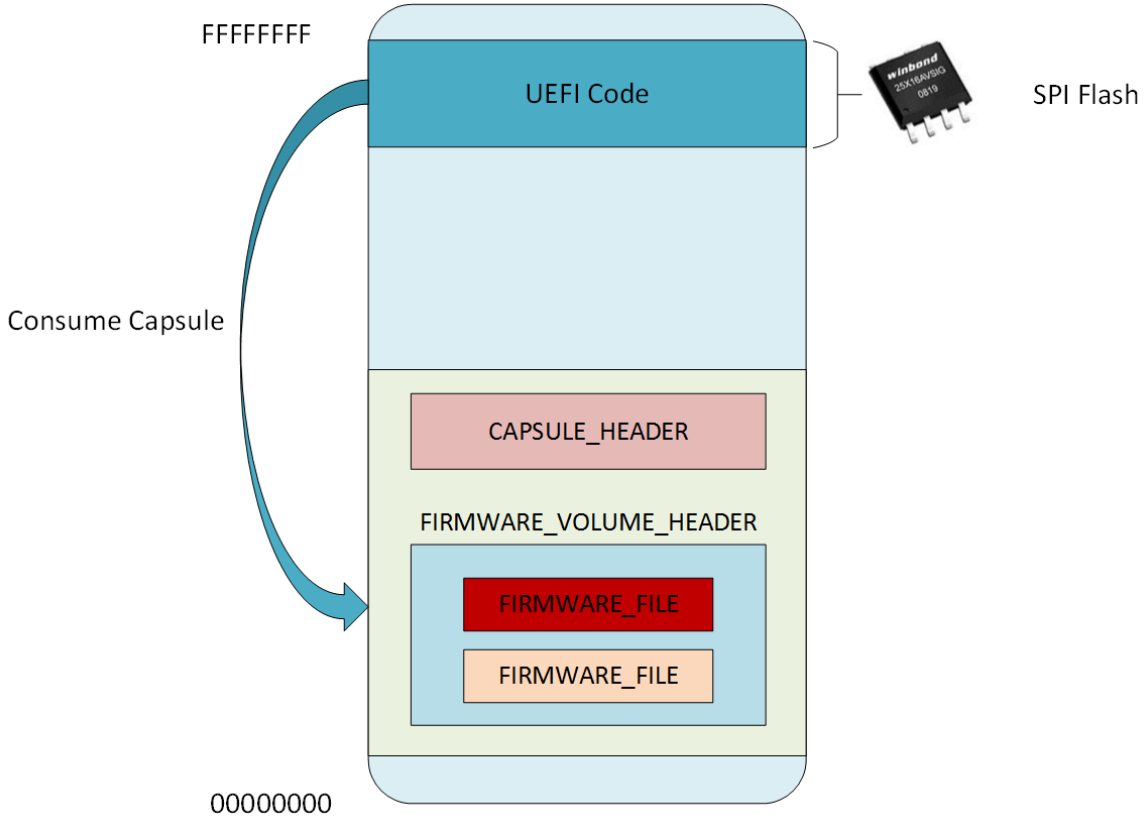
- **The UEFI code "coalesces" the firmware capsule back into its original form.**

Capsule Verification



- **UEFI parses the envelope of the firmware capsule and verifies that it is signed by the OEM**

Capsule Consumption



- **Contents of the capsule are then consumed….**
  - Flash contents to the SPI flash
  - Run malware detection independent of the operating system
  - Etc…

Opportunities For Vulnerabilities

- **There are 3 main opportunities for memory corruption vulnerabilities in the firmware capsule processing code**
  1. The coalescing phase
  2. Parsing of the capsule envelope
  3. Parsing of unsigned content within the capsule
- **Our audit of the UEFI capsule processing code yielded multiple vulnerabilities in the coalescing and envelope parsing code**
  - The first "BIOS reflash" exploit was presented by Wojtczuk and Tereshkin. They found it by reading the UEFI code which handled BMP processing and exploiting an unsigned splash screen image embedded in a firmware[1]

Bugs Galore

```
if (*MemorySize <= (CapsuleSize + DescriptorsSize)) { <= Bug 1
  return EFI_BUFFER_TOO_SMALL;
}
```

```
    //
    Desc = (EFI_CAPSULE_BLOCK_DESCRIPTOR  *
  } else {
    Size += (UINTN) Desc->Length; <= Bug 2
    Count++;
```

```
LbaCache = AllocatePool (FvbDev->NumBlocks * sizeof (LBA_CACHE)); <= Bug 3
```

```
if (((Buff1 + Size1) <= Buff2) || (Buff1 >= (Buff2 + Size2))) { <= Bug 4
  return FALSE;
}
```

- **We spent ~1 week looking at the UEFI reference implementation and discovered vulnerabilities in the capsule processing code**
  - We found 2 exploitable vulnerabilities code-named after chess moves. King's Gambit is in DXE phase, Queen's Gambit in PEI phase.
- **The vulnerabilities allow an attacker to get code execution in the context of an almost entirely unlocked platform**

Vulnerabilities Summary

```
} else {
  //
  //To enhance the reliability of check-up, the first capsule's header is checked here.
  //More reliabilities check-up will do later.
  if (CapsuleSize == 0) {
    //
    //Move to the first capsule to check its header.
    //
    CapsuleHeader = (EFI_CAPSULE_HEADER*)((UINTN)Ptr->Union.DataBlock);
    if (IsCapsuleCorrupted (CapsuleHeader)) {
      return NULL;
    }
    CapsuleCount ++;
    CapsuleSize = CapsuleHeader->CapsuleImageSize;
```

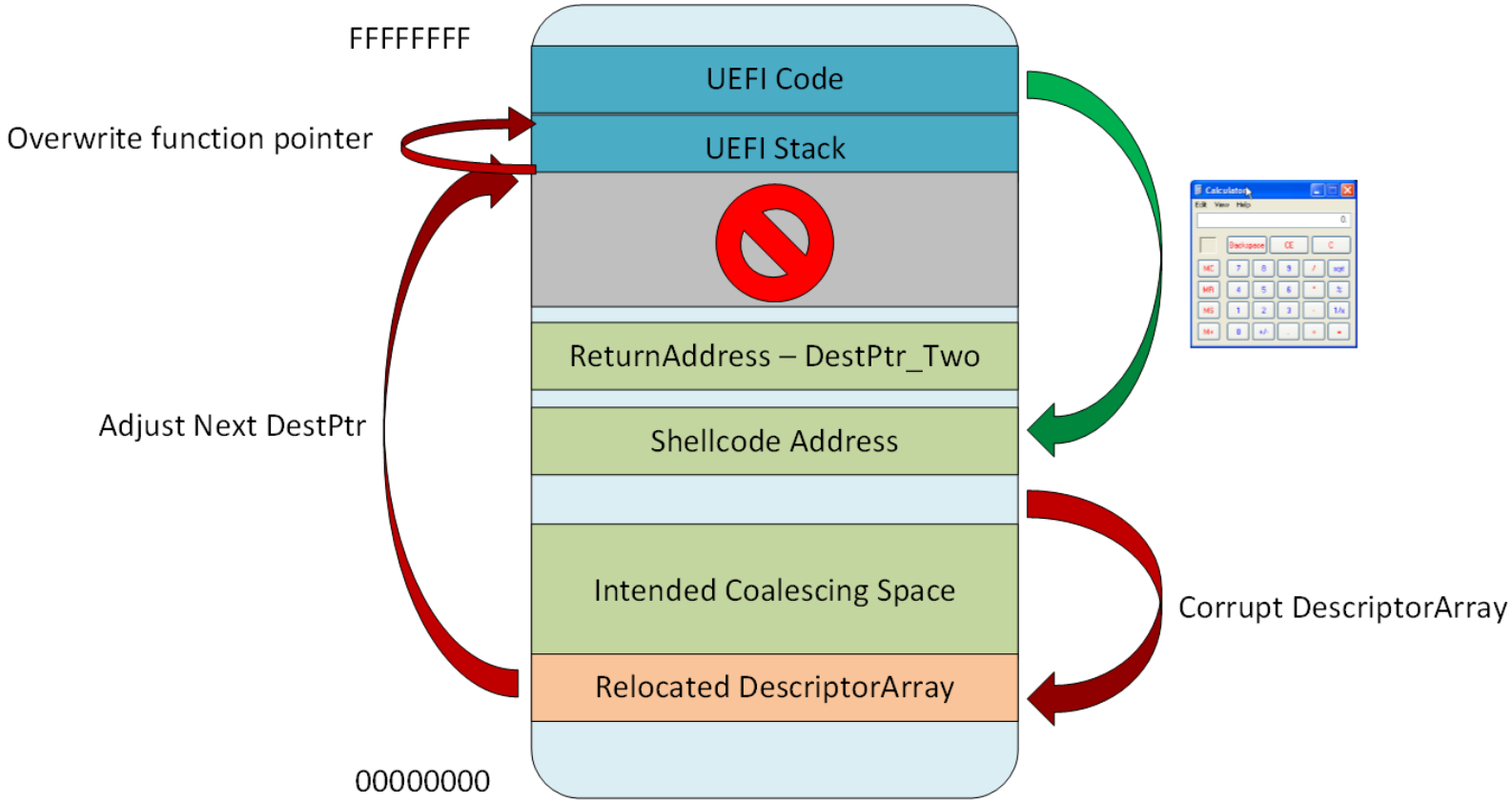
ValidateCapsuleIntegrity: Edk2/MdeModulePkg/Universal/CapsulePei/Common/CapsuleCoalesce.c

- **The presence of easy to spot integer overflows in open source and security critical code is… *disturbing***
  - "Many eyes make all bugs shallow"… so is anyone (defensive) looking?

Onward To Exploitation

- **The aforementioned code runs with read-write-execute permissions**
  - Flat protected mode with paging disabled
  - No mitigations whatsoever
- **However, successful exploitation in this unusual environment was non-trivial**

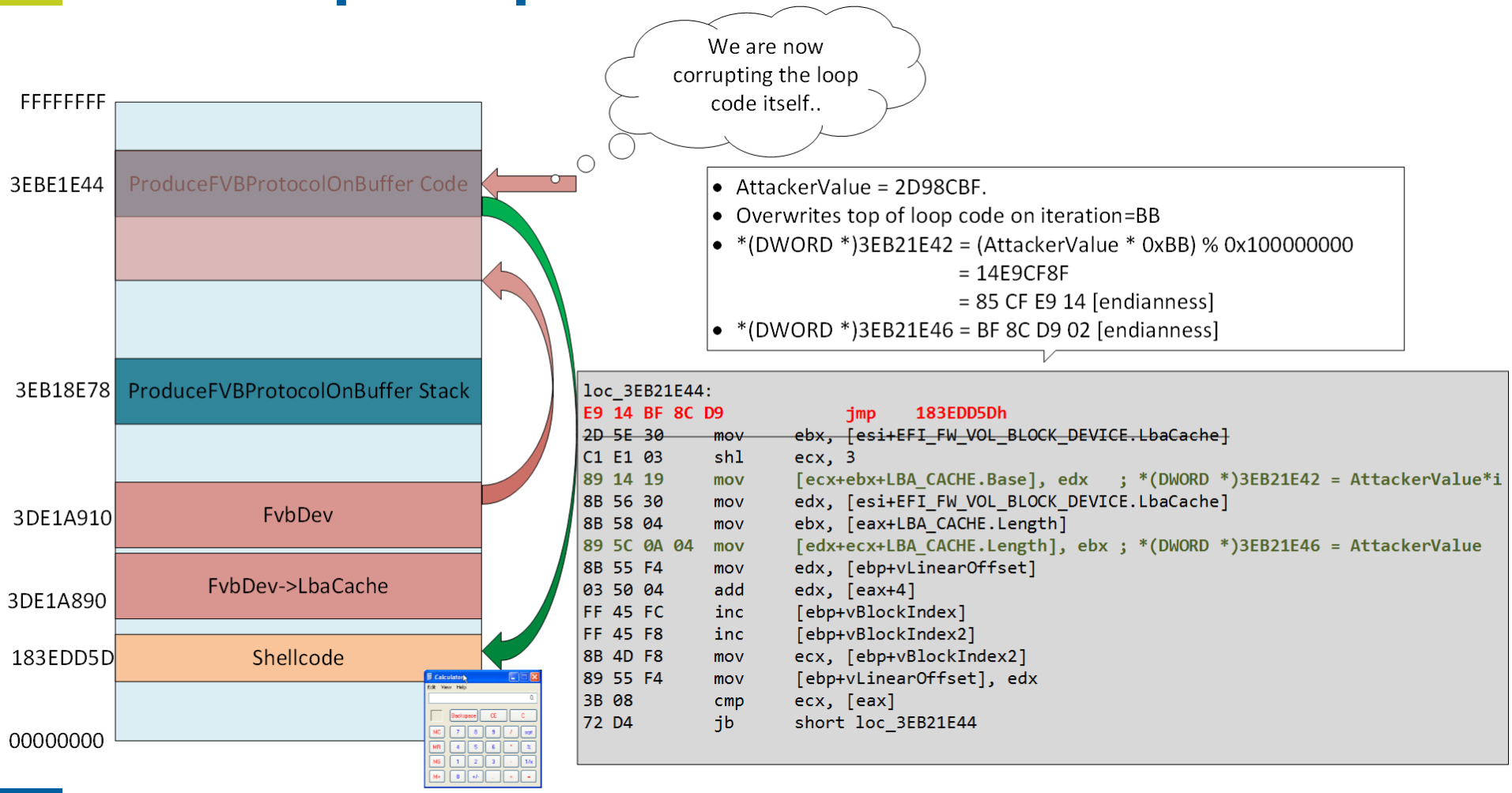
Coalescing Exploit Success



- **Exploited using a multistage approach that involved corrupting the scatter-gather list**
  - Achieves surgical write-what-where primitive

See whitepaper for full details on the exploitation technique

Envelope Exploitation Success



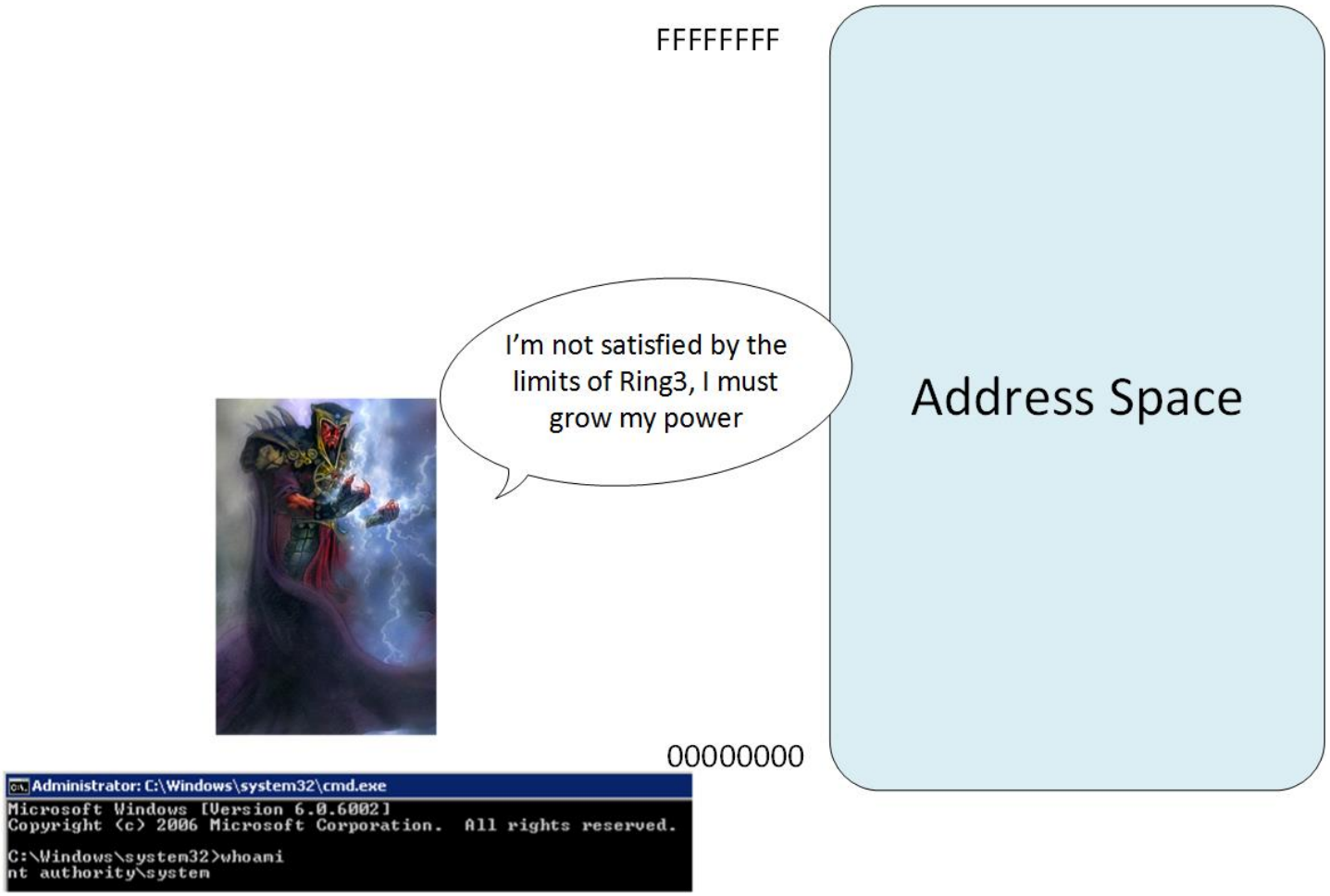
- **Memory corruption took the form of a non-terminating loop writing partially controlled values**
- **Exploited by having non-terminating loop self-overwrite**

See whitepaper for full details on the exploitation technique

Exploitation Mechanics Summary

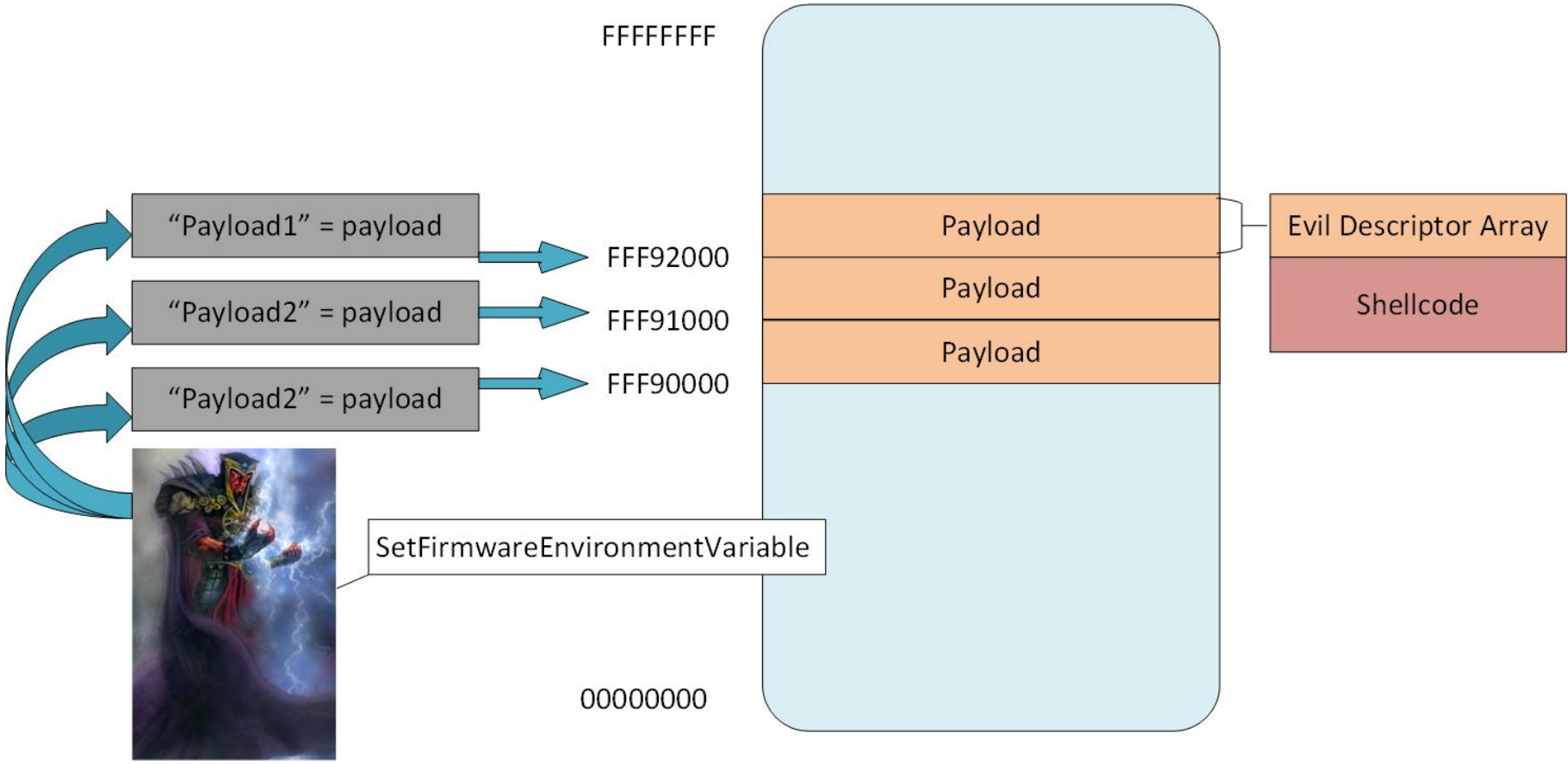
- **See the whitepaper for the super nitty-gritty details**
- **Capsule coalescing exploit (Queen's Gambit) allows for surgical write-what-where primitive resulting in reliable exploitation of the UEFI firmware**
  - Exploited using only Windows 8 EFI variable API
  - Stores payload at predictable physical addresses by spraying EFI variables onto the SPI flash
  - CVE-2014-4860
- **Capsule envelope parsing vulnerability (King's Gambit) can be exploited but corrupts a lot of the address space**
  - System possibly left in an unstable state if not rebooted
  - Relies on a 3rd party kernel driver to stage payload at a certain physical address
  - CVE-2014-4859
- **In both cases, attacker ends up with control of EIP in the early boot environment**

Exploitation Flow (1 of 9)



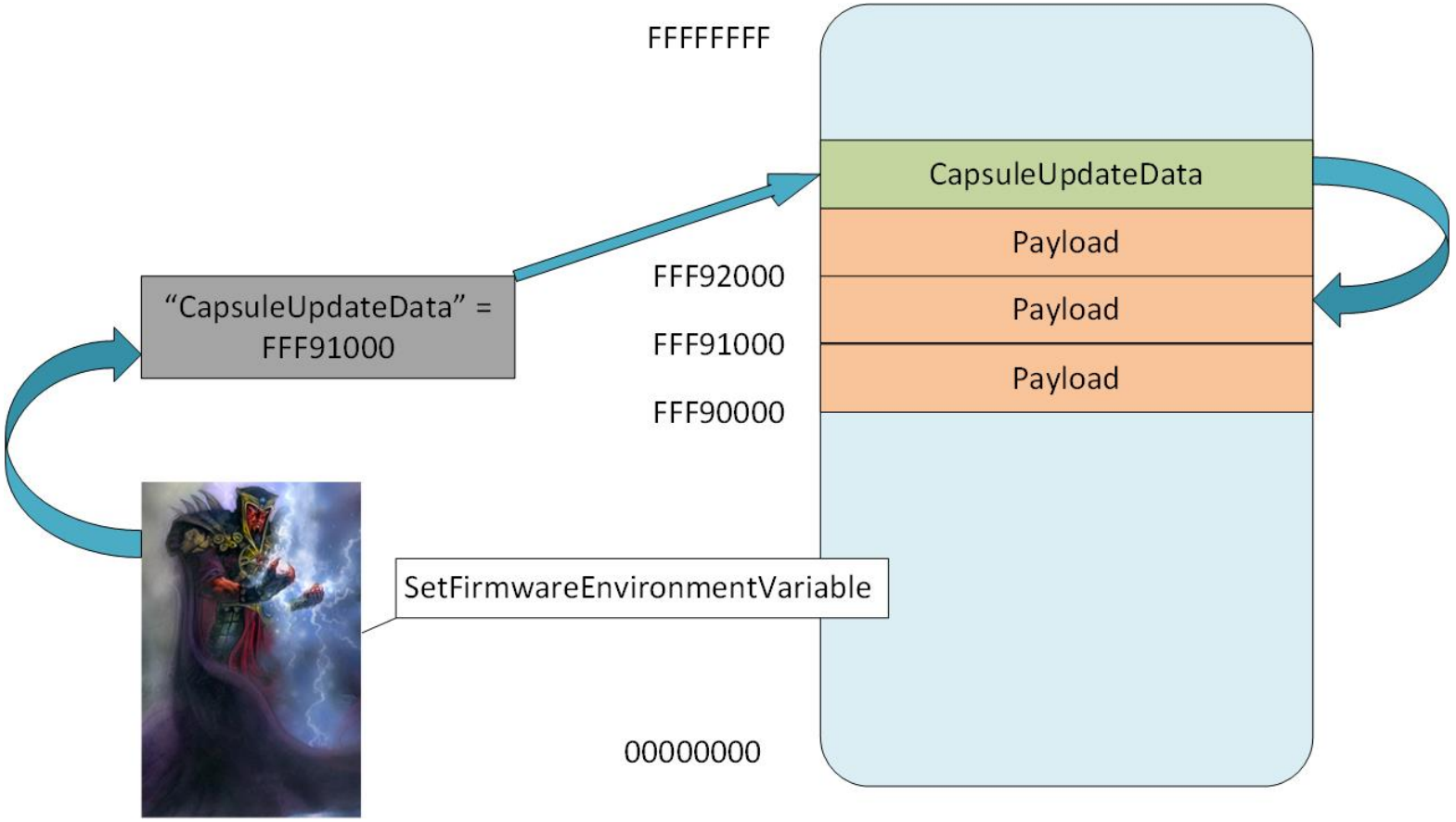
- **Our Sith attacker is unimpressed with his ring 3 admin privileges and seeks to grow his power through the dark side of the force**

Exploitation Flow (2 of 9)



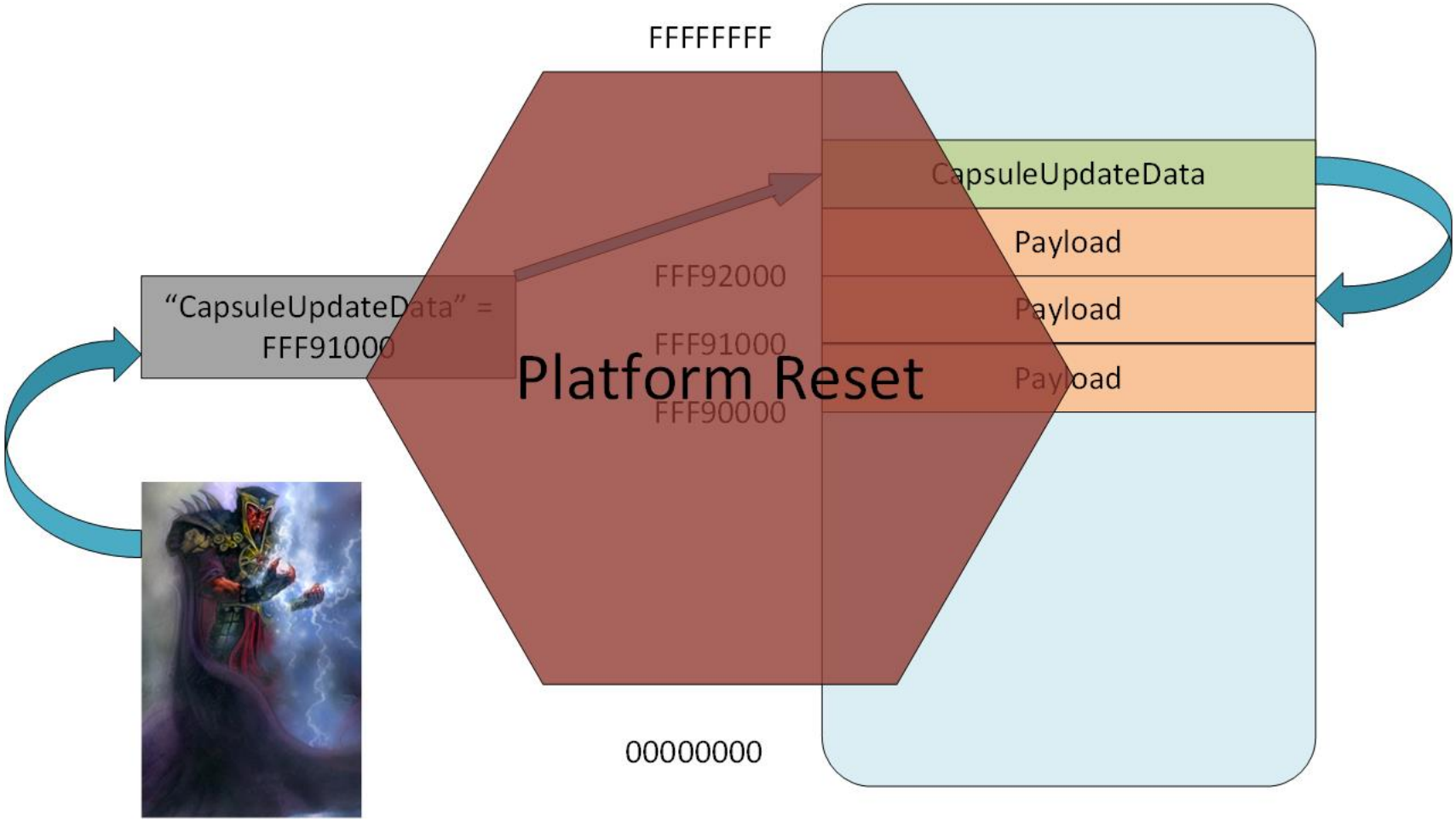
- **Attacker creates many copies of a payload variable**
  - Payload contains evil capsule as well as shellcode
- **Similar to heap spray, this technique puts the attackers payload at a predictable physical address**

Exploitation Flow (3 of 9)



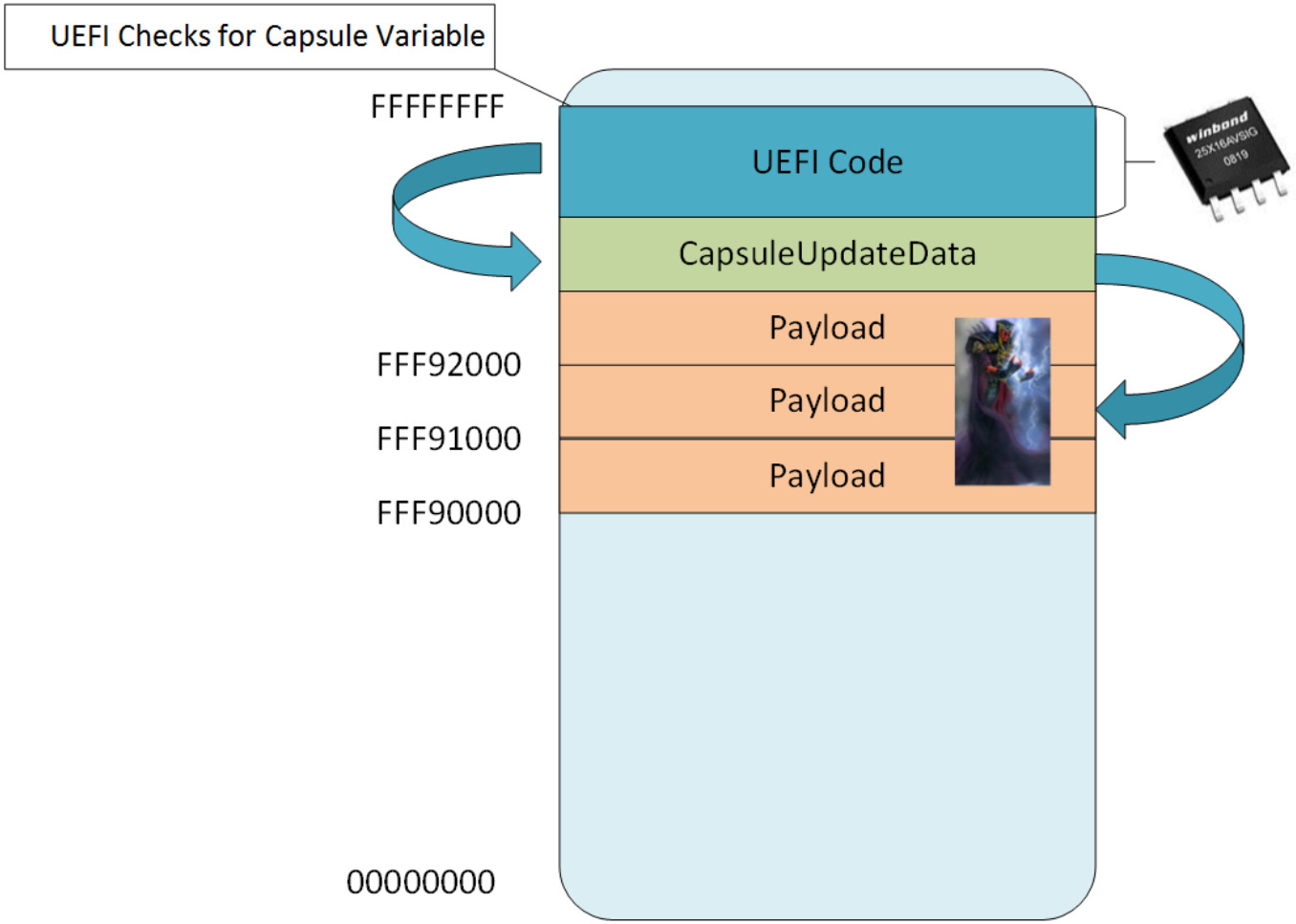
- **Attacker prepares to initiate capsule update by creating the CapsuleUpdateData variable**

Exploitation Flow (4 of 9)



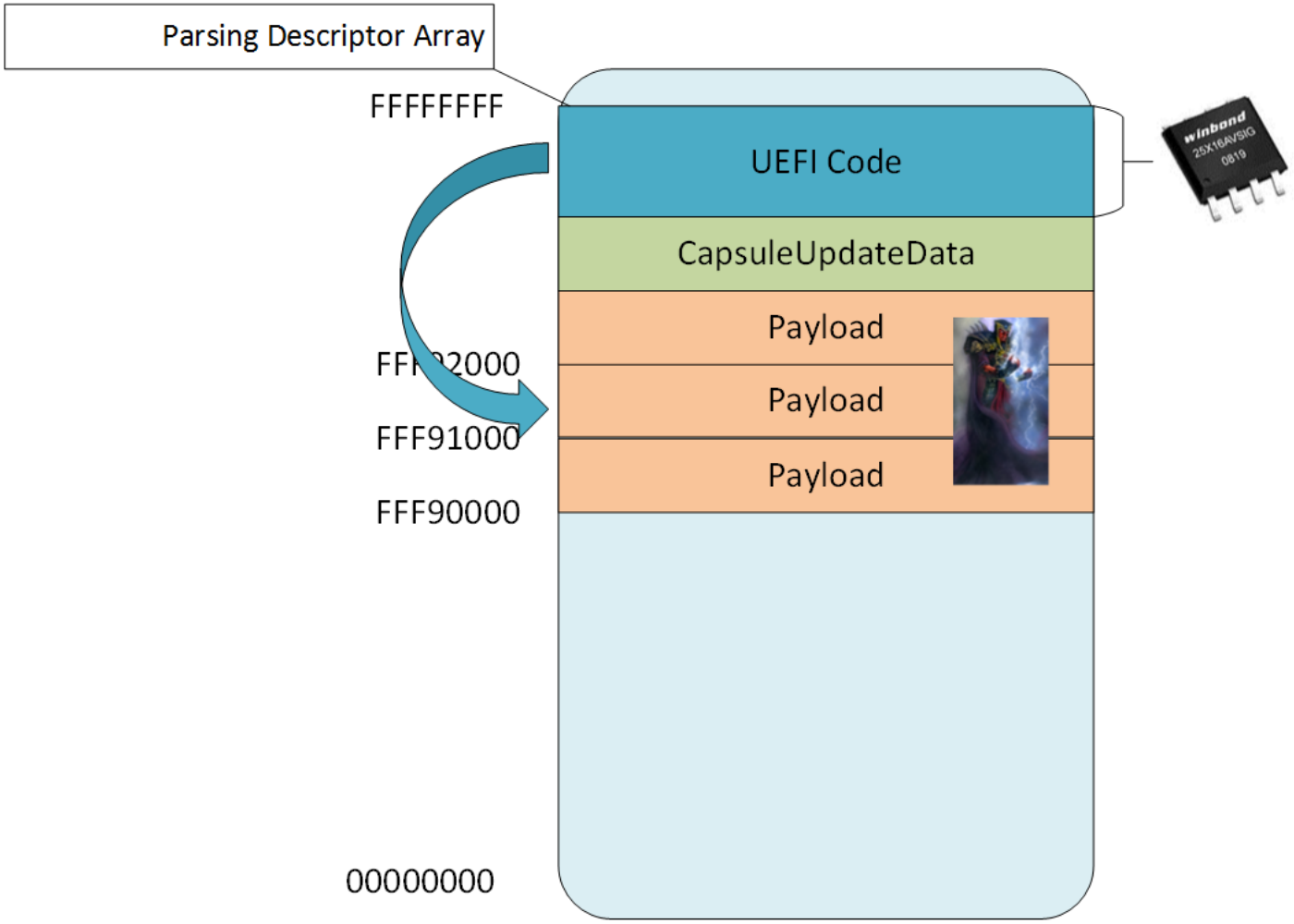
- **Warm reset is performed to transfer context back to UEFI**
  - "Warm reset" probably means S3 sleep but is implementation specific

Exploitation Flow (5 of 9)



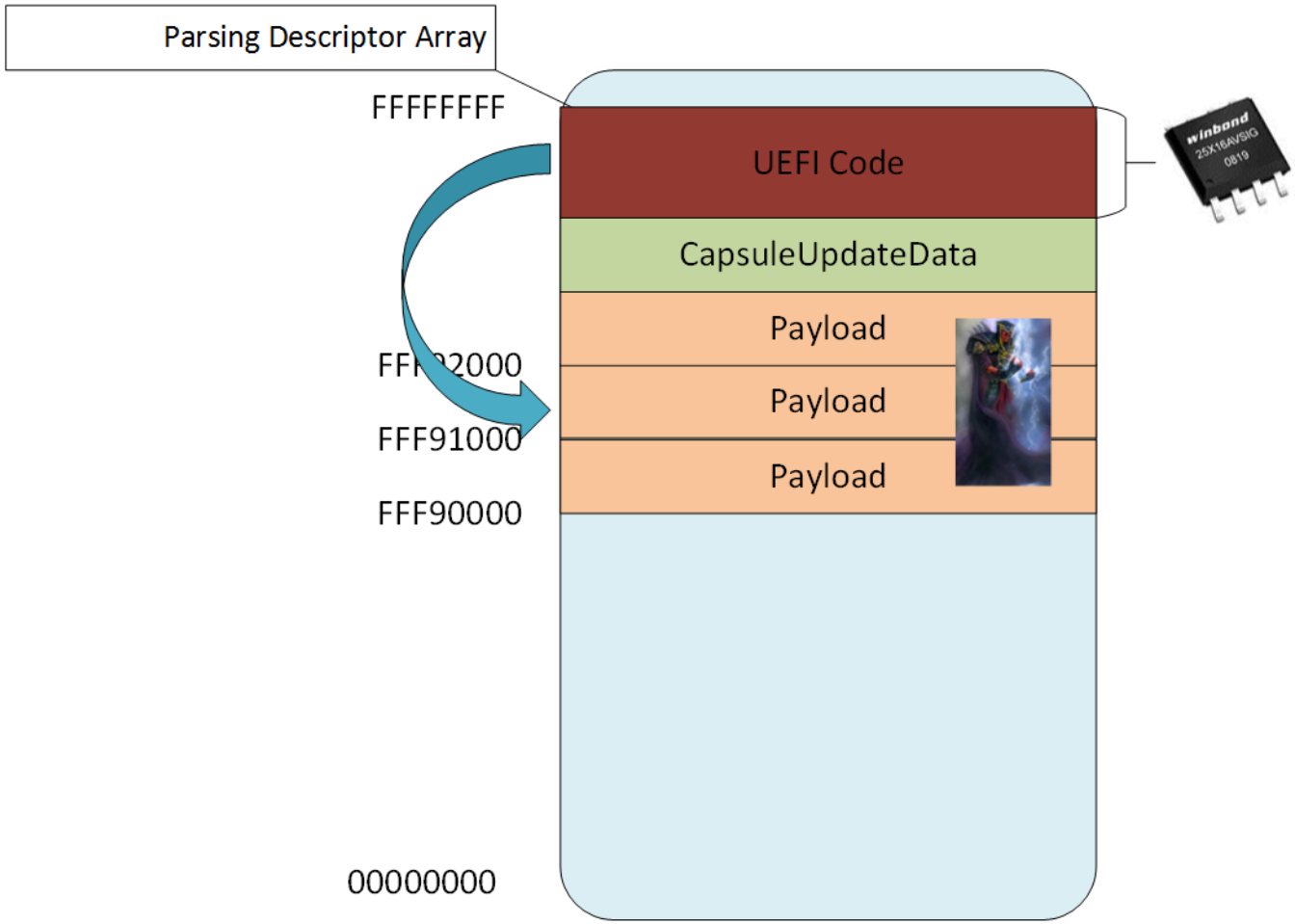
- **Capsule processing is initiated by the existence of the "CapsuleUpdateData" UEFI variable**

Exploitation Flow (6 of 9)



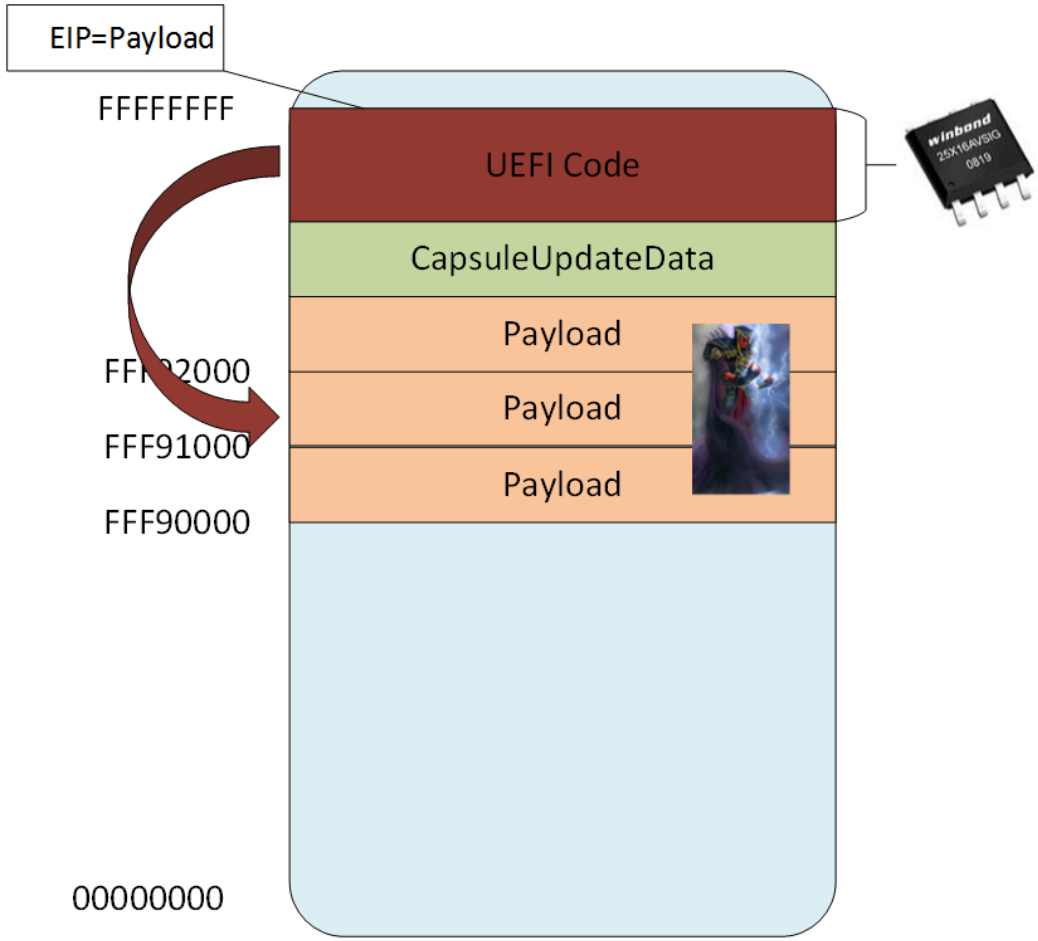
- **UEFI begins to coalesce the evil capsule**

Exploitation Flow (7 of 9)



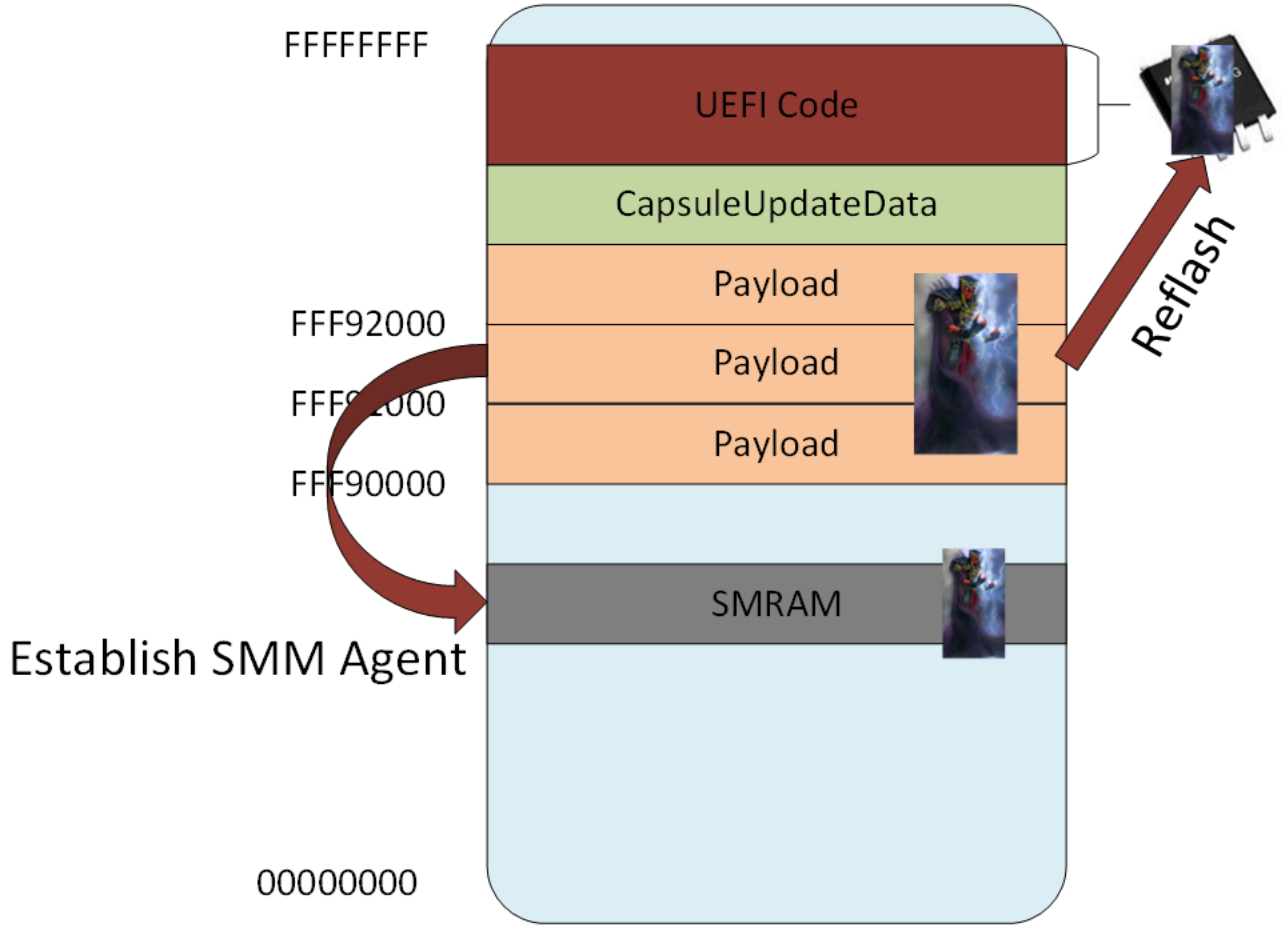
- **UEFI becomes corrupted while parsing evil capsule**

Exploitation Flow (8 of 9)



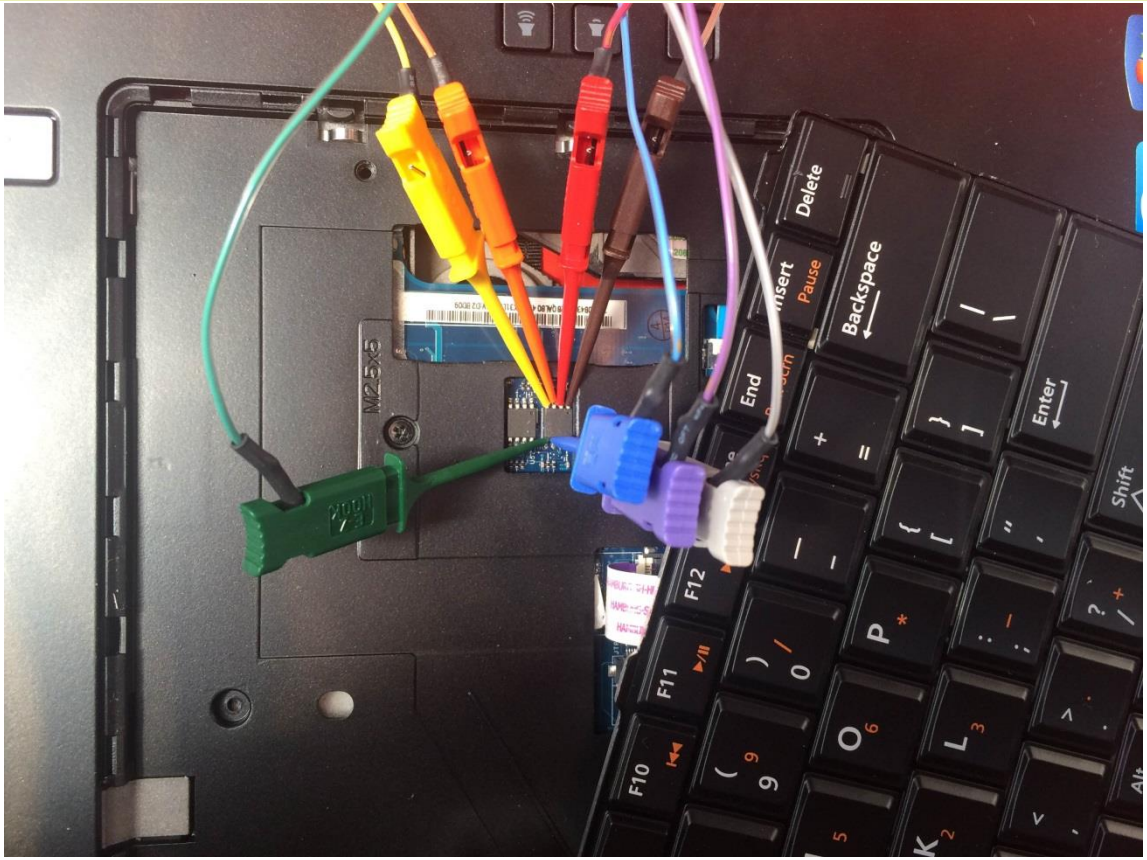
- **Attacker gains arbitrary code execution in the context of the early boot environment**
  - Platform is unlocked at this point

Exploitation Flow (9 of 9)



- **Attacker can now establish agents in SMM and/or the platform firmware to do their bidding**

Attack Result



- What previously required physical access can now be performed through software-only means.
- However, recovering from this attack would require physical access!

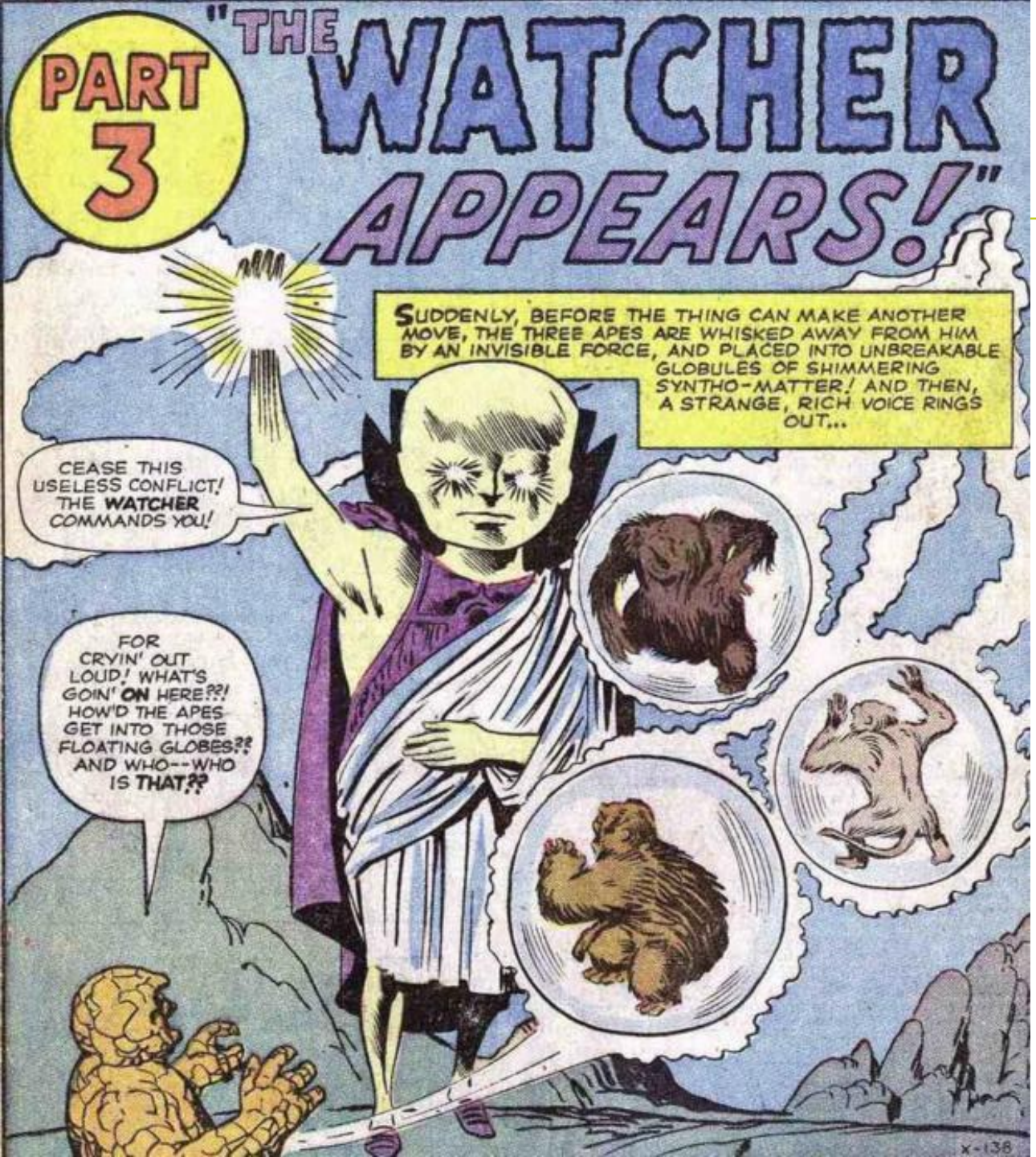
BIOS Attacks: So What?
What Can Attackers Do If They Break Into BIOS?

- **We get asked this question a lot, and our answer is "EVERYTHING! YOU CAN DO EVERY. SINGLE. THING!" or "A BIOS attacker has available to it a superset of the capabilities of all lower privileged attackers."**
- **But of course they can be excused for thinking we're just another group of security folks trying to spread FUD.**
- **We don't spread FUD, we talk about what we know to be technologically and architecturally possible.**

The Power of BIOS

- **With these new powers, an attacker can:**
  - Brick the platform
  - Defeat Secure Boot[2]
  - Establish an undetectable SMM rootkit[8][5]
  - Subvert hypervisors[9]
  - Subvert TXT launched hypervisors[3]
  - Circumvent operating system security functions[11]
  - Survive operating system reinstallation attempts
  - Other?

Marvel Comics
Fantastic Four #13, 1963



Presenting the first appearance of The Watcher!

The Watcher

- **The Watcher lives in SMM (where you can't look for him)**
- **It has no build-in capability except to scan memory for a magic signature**
- **If it finds the signature, it treats the data immediately after the signature as code to be executed**
- **In this way the Watcher performs *arbitrary code execution* on behalf of some controller, and is *completely OS independent***

- **A controller is responsible for placing into memory payloads for The Watcher to find**
- **These payloads can make their way into memory through any means**
  - Could be sent in a network packet which is never even processed by the OS
  - Could be embedded somewhere as non-rendering data in a document
  - Could be generated on the fly by some malicious javascript that's pushed out through an advertisement network
  - Could be pulled down by a low-privilege normal-looking dropper
  - Use your imagination

The Watcher, watching

Design tradeoffs:

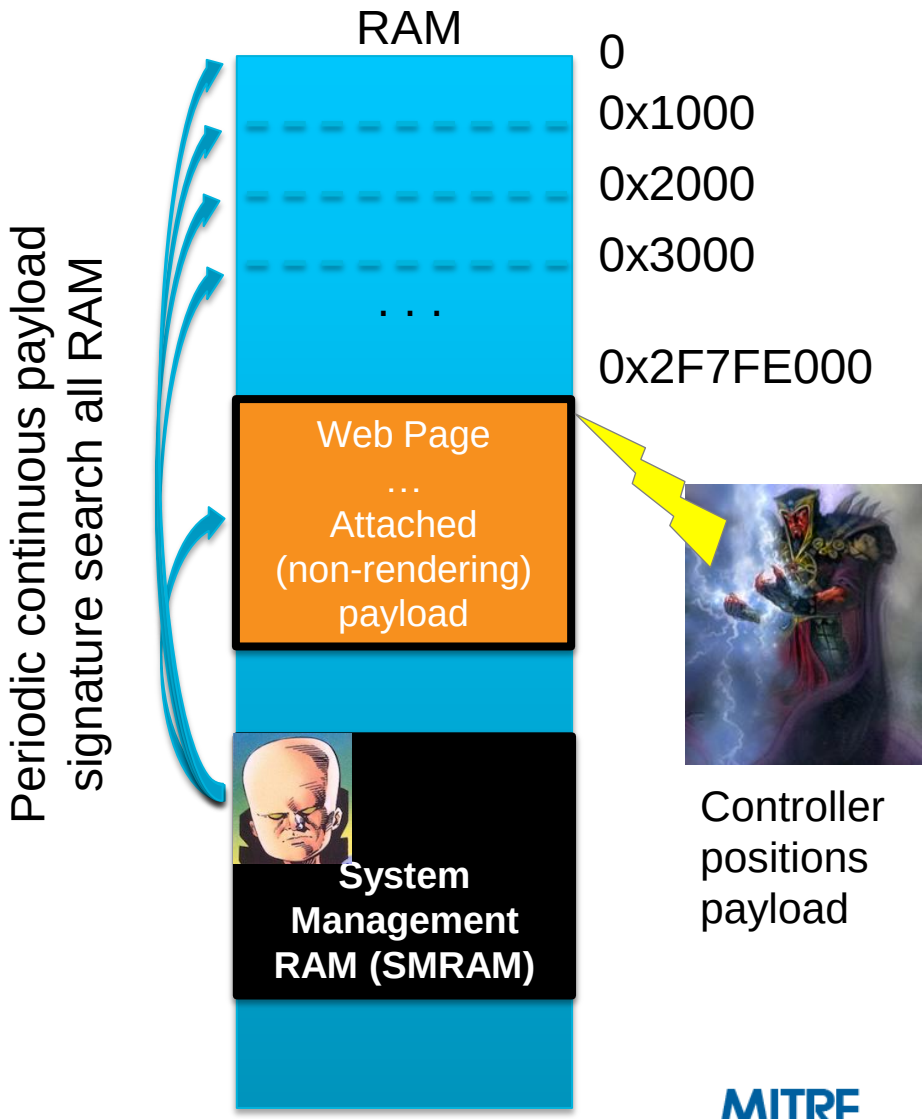
We don't want to scan every 4 byte chunk of memory. So instead we scan every 0x1000-aligned page boundary.

How do we guarantee a payload will be found on a page-aligned boundary?

a) Another agent puts it there

b) Controller prefixes the payload with a full 0x1000 worth of signatures and pointers to the code to be executed (this guarantees a signature will always be found at the boundary or boundary+4)

There are obviously many different ways it could be built.



Vulnerability Disclosure & Vendor Response

http://www.kb.cert.org/vuls/id/552286
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4859
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4860

- **We told Intel & CERT about the bugs we found on Nov 22nd (King's Gambit) and Dec 4th (Queen's Gambit) 2013**
  - We conveyed that we would extend our typical 6 month responsible disclosure deadline, and we would be targeting public disclosure in the summer at BlackHat/Defcon
    - MITRE sets a 6 month default deadline to help prioritization to fix the problems. Things without deadlines have a tendency to not get done.
  - We also directly contacted some of the OEMs that we had the ability to send encrypted email to
- **Intel patched the bugs in the UEFI source code in January 2014, and they are patched in the latest stable UEFI Developers Kit (UDK) 2014 release (March 2014)**
- **Intel held multiple meetings with many OEMs and IBVs to communicate and clarify issues. They also asked the vendors to report which systems were vulnerable.**

MITRE

Vulnerability Disclosure & Vendor Response

http://www.kb.cert.org/vuls/id/552286
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4859
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4860

- **Then we didn't hear anything for a while.**
- **In June we started to get nervous that there was a mismatch in our expectations about what vendors would be telling us**
  - We expected to get a list of before BlackHat of which BIOS revisions vendors had released that patched the vulnerabilities.
  - What we got instead was a taste of the bad old days where some vendors didn't reply Intel, others replied that they're not vulnerable when they actually are, and others replied under NDA and we don't know what they said.
- **In July we had to start an aggressive follow-up campaign with OEMs and IBVs where we specifically went and looked at their systems to try and identify signatures that indicate the presence of the vulnerable code, so we could cite specific evidence that they were vulnerable.**

Current Understanding (1 of 4)

- **As expected, many members of the ecosystem were vulnerable**

Current Understanding (2 of 4)

- HP: 33 enterprise and 470 consumer models vulnerable
- Dell: 39 enterprise models
- Lenovo: TBD models

Current Understanding (3 of 4)

American Megatrends

phoenix technologies

insyde™

DELL

hp

ASUS

lenovo

acer

- **Insyde – “We didn’t use vulnerable code from reference implementation”**

Current Understanding (4 of 4)

- **Unknown vulnerability status for many OEMs…**

USRT

Unified Extensible Firmware Interface Forum

Member Pag

Home » About

REPORTING A SECURITY ISSUE

If you have information about a security issue or vulnerability with a product that may be due to its UEFI-based firmware, please send an e-mail to security@uefi.org. Encrypt sensitive information using our PGP public key.

Please provide as much information as possible, including:

- The products and versions affected
- Detailed description of the vulnerability
- Steps to demonstrate the vulnerability or reproduce the exploit, including specific configurations or peripherals, if relevant
- Potential impact of the vulnerability, when exploited
- Information on known exploits

- **Our experience disclosing these issues revealed that the BIOS eco-system was not well prepared to handle security vulnerability reports**
- **The UEFI Forum has started a security response team to remedy the problem**

MITRE

What can you do about it?

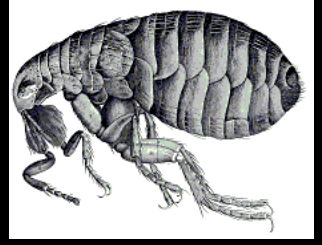
- **Run Copernicus. It has been updated to automatically report if your system is on the small list of currently known-affected systems for CERT VU # 552286 (the CERT VU and Copernicus will be updated as more vendors acknowledge their vulnerability)**
  - http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/copernicus-question-your-assumptions-about or just search for "MITRE Copernicus"
- **We are now releasing our UEFI binary integrity checking script (bios_diff.py) for use on UEFI BIOS dumps. This can help you detect if your BIOS has been backdoored**
  - You can often extract "known good" BIOS dumps from BIOS update applications. We have a basic collection, but this doesn't scale well.
  - We're going to be working with BIOS vendors to get a standard metadata format whereby they can provide true known good contents of the flash chips, and what should and shouldn't naturally change (e.g. where are the UEFI non-volatile variables, etc)

MITRE

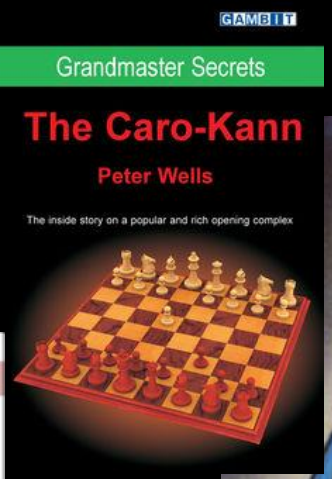
What can you do about it?

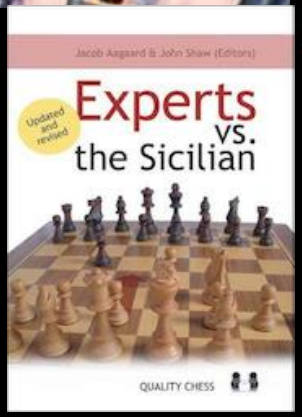
- **If you're in charge of an enterprise, start running BIOS updates**
  - And start requesting your asset management software vendor include BIOS revision and vulnerability status information
- **If you're a security vendor, start including BIOS checks**
  - If you're a customer, start asking for BIOS checks
- **We are happy to freely give away our Copernicus code to get vendors started with incorporating checking BIOSes. All we ask for in return is some data to help further our research and help show why BIOS security is so important.**
- **We want BIOS configuration & integrity checking to become standard capabilities which are widely available from as many vendors as possible.**
  - No more massive blind spot please!

Ticks



Fleas

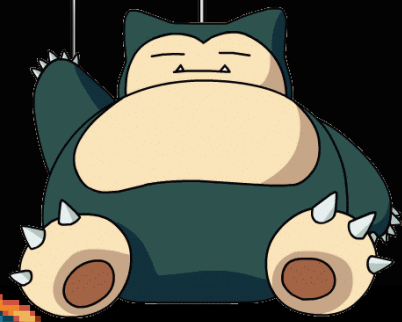




The Watcher

Sandman

(Coming soon!)



Snorlax

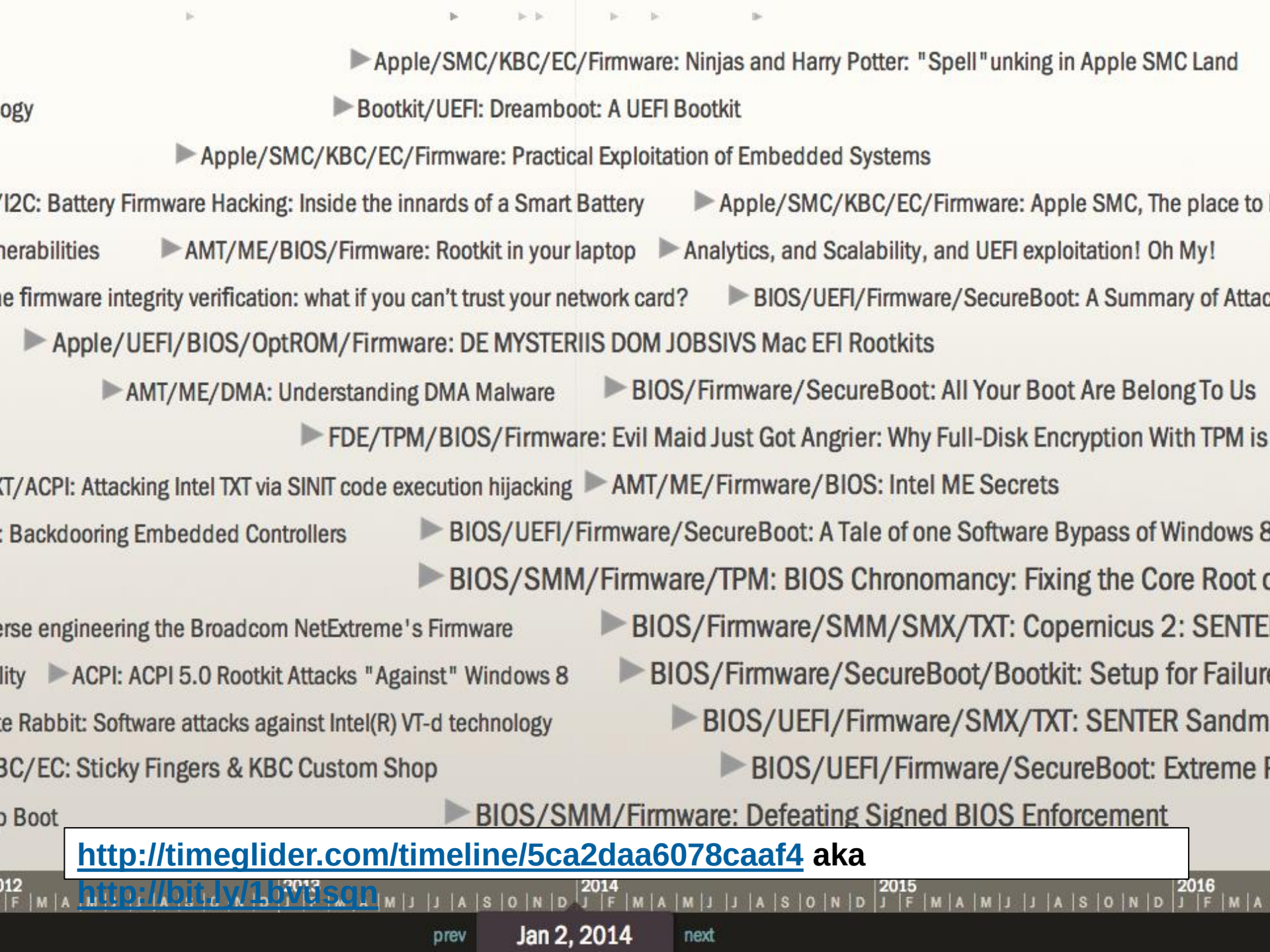
(Coming soon!)

Smite'em the Stealthy

Charizard

Queen's Gambit

King's Gambit


Apple/SMC/KBC/EC/Firmware: Ninjas and Harry Potter: "Spell"unking in Apple SMC Land
ogy
Bootkit/UEFI: Dreamboot: A UEFI Bootkit
Apple/SMC/KBC/EC/Firmware: Practical Exploitation of Embedded Systems
I2C: Battery Firmware Hacking: Inside the innards of a Smart Battery
Apple/SMC/KBC/EC/Firmware: Apple SMC, The place to
nerabilities
AMT/ME/BIOS/Firmware: Rootkit in your laptop
Analytics, and Scalability, and UEFI exploitation! Oh My!
e firmware integrity verification: what if you can't trust your network card?
BIOS/UEFI/Firmware/SecureBoot: A Summary of Attac
Apple/UEFI/BIOS/OptROM/Firmware: DE MYSTERIIS DOM JOBSIVS Mac EFI Rootkits
AMT/ME/DMA: Understanding DMA Malware
BIOS/Firmware/SecureBoot: All Your Boot Are Belong To Us
FDE/TPM/BIOS/Firmware: Evil Maid Just Got Angrier: Why Full-Disk Encryption With TPM is
T/ACPI: Attacking Intel TXT via SINIT code execution hijacking
AMT/ME/Firmware/BIOS: Intel ME Secrets
Backdooring Embedded Controllers
BIOS/UEFI/Firmware/SecureBoot: A Tale of one Software Bypass of Windows 8
BIOS/SMM/Firmware/TPM: BIOS Chronomancy: Fixing the Core Root
erse engineering the Broadcom NetExtreme's Firmware
BIOS/Firmware/SMM/SMX/TXT: Copernicus 2: SENTE
lity
ACPI: ACPI 5.0 Rootkit Attacks "Against" Windows 8
BIOS/Firmware/SecureBoot/Bootkit: Setup for Failur
e Rabbit: Software attacks against Intel(R) VT-d technology
BIOS/UEFI/Firmware/SMX/TXT: SENTER Sandm
BC/EC: Sticky Fingers & KBC Custom Shop
BIOS/UEFI/Firmware/SecureBoot: Extreme
o Boot
BIOS/SMM/Firmware: Defeating Signed BIOS Enforcement
http://timeglider.com/timeline/5ca2daa6078caaf4 aka
http://bit.ly/1bvusqn
2012
2013
2014
2015
2016
prev
Jan 2, 2014
next

Today's Presentation Results

- We have found and disclosed two new exploitable vulnerabilities.
- These vulnerabilities would allow an attacker to take control of the system before any security is enabled, and persist indefinitely via the SPI flash chip.
- We have also invented a new technique to make BIOS/kernel exploits more reliable by staging shellcode into UEFI non-volatile variables, which will be mapped at predictable locations.
- We have shown The Watcher, which is an example of how an attacker can gain OS-independent arbitrary code execution in the most privileged x86 execution domain, System Management Mode.
- We have updated our public "Copernicus" software which can integrity check a BIOS to look for backdoors, or check for the presence of known vulnerabilities.

MITRE

Conclusions

- **It's time to get serious about firmware security**
  - Start patching your BIOSes
  - Start demanding firmware inspection capabilities
- **UEFI has more tightly coupled the bonds of the operating system and the platform firmware**
- **Specifically, the EFI variable interface acts as a conduit by which a less privileged entity (the operating system) can pass information for consumption by a more privileged entity (the platform firmware)**
  - We have demonstrated how a vulnerability in this interface can allow an attacker to gain control of the firmware
- **Although the authors believe UEFI to ultimately be a good thing for the overall state of platform security, a more thorough audit of the UEFI code and OEMs/IBVs' extra "value added" code is needed**
- **MITRE's Copernicus continues to be updated and remains the only enterprise-deployable system that can integrity check and vulnerability check your BIOSes**
  - But *MITRE doesn't make products* so industry needs to come talk to us

MITRE

Questions & Contact

- {ckallenberg, xkovah, jbutterworth, scornwell} @ mitre . org
- Copernicus @ mitre . org
- @coreykal, @xenokovah, @jwbutterworth3, @ssc0rnwell
- @MITREcorp

- P.s., go check out OpenSecurityTraining.info!
- @OpenSecTraining

References

- [1] Attacking Intel BIOS – Alexander Tereshkin & Rafal Wojtczuk – Jul. 2009 http://invisiblethingslab.com/resources/bh09usa/Attacking%20Intel%20BIOS.pdf
- [2] A Tale of One Software Bypass of Windows 8 Secure Boot – Yuriy Bulygin – Jul. 2013 http://blackhat.com/us-13/briefings.html#Bulygin
- [3] Attacking Intel Trusted Execution Technology - Rafal Wojtczuk and Joanna Rutkowska – Feb. 2009 http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf
- [4] Defeating Signed BIOS Enforcement – Kallenberg et al., Sept. 2013 – http://www.mitre.org/sites/default/files/publications/defeating-signed-bios-enforcement.pdf
- [5] BIOS Chronomancy: Fixing the Core Root of Trust for Measurement – Butterworth et al., May 2013 http://www.nosuchcon.org/talks/D2_01_Butterworth_BIOS_Chronomancy.pdf
- [6] IsGameOver() Anyone? – Rutkowska and Tereshkin – Aug 2007 http://invisiblethingslab.com/resources/bh07/IsGameOver.pdf
- [7] Defeating Windows Driver Signature Enforcement – j00ru - Dec 2012 http://j00ru.vexillium.org/?p=1455

MITRE

References 2

- [8] Copernicus 2 – SENTER The Dragon – Kovah et al. – March 2014 http://www.mitre.org/sites/default/files/publications/Copernicus2-SENTER-the-Dragon-CSW-.pdf
- [9] Preventing and Detecting Xen Hypervisor Subversions – Rutkowska and Wojtczuk – Aug 2008 http://www.invisiblethingslab.com/resources/bh08/part2-full.pdf
- [10] A New Breed of Rootkit: The Systems Management Mode (SMM) Rootkit – Sparks and Embleton – Aug 2008 http://www.eecs.ucf.edu/~czou/research/SMM-Rootkits-Securecom08.pdf
- [11] Using SMM for "Other Purposes" – BSDaemon et al – March 2008 http://phrack.org/issues/65/7.html
- [12] Using SMM to Circumvent Operating System Security Functions – Duflot et al. – March 2006 http://fawlty.cs.usfca.edu/~cruse/cs630f06/duflot.pdf
- [13] Setup for Failure: Defeating UEFI SecureBoot – Kallenberg et al. – April 2014 http://www.syscan.org/index.php/download/get/6e597f6067493dd581eed737146f3afb/SyScan2014_CoreyKallenberg_SetupforFailureDefeatingSecureBoot.zip

MITRE