Advancing Windows Security

Bluehat Shanghai 2019 | David "dwizzzle" Weston | Microsoft OS Security Group Manager


早上好 上海！
震旦

Windows is evolving....

Windows for PCs
Familiar desktop experience
Broad hardware ecosystem
Desktop app compat

Windows on XBOX
10 Shell experience
Unique security model
Shared gaming experience

Windows on IOT
Lean core platform
Azure connected
Runtimes and Frameworks

Windows for ...
Form factor appropriate shell experience
Device specific scenario support

One Core OS
Base OS
App and Device Platform
Runtimes and Frameworks

All code executes with integrity.

User identities cannot be compromised, spoofed, or stolen.

Attacker with casual physical access cannot modify data or code on the device.

Malicious code cannot persist on a device.

Violations of promises are observable.

All apps and system components have only the privilege they need.

Increasing Security

Windows 10 S

1 Run as admin

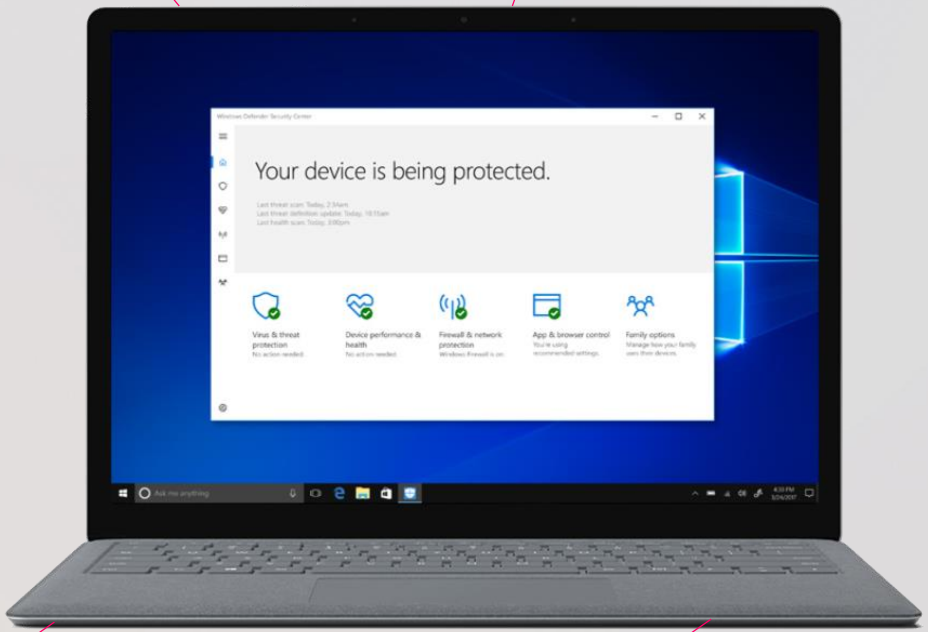
2 Execute Unsigned Code

3 Use passwords

4 Mitigations not always on

1 Mandatory Code Signing

3 Internet scripts and macros blocked



2 Complete Password-less

4 "Admin-less" user account

10 S: Millions of installs, no widespread detections of malware

Classic

10 S

All code executes with integrity.

Windows 10 S

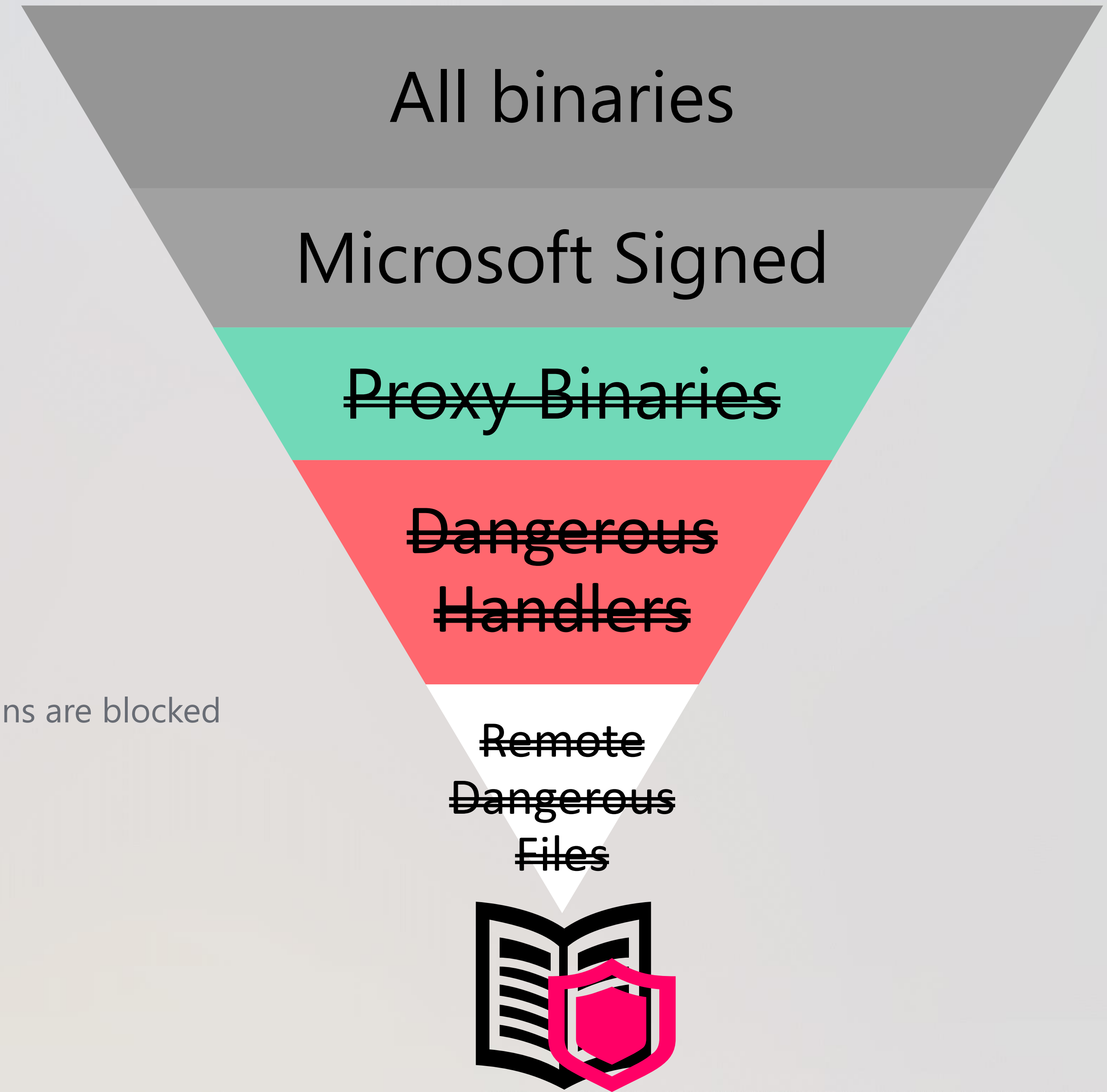
Code Integrity Improvements

CI policy removes many “proxy” binaries

Store signed only apps (UWP or Centennial)

“Remote” file extensions that support dangerous actions are blocked

Remote Office Macros are blocked by default



Windows 10 S

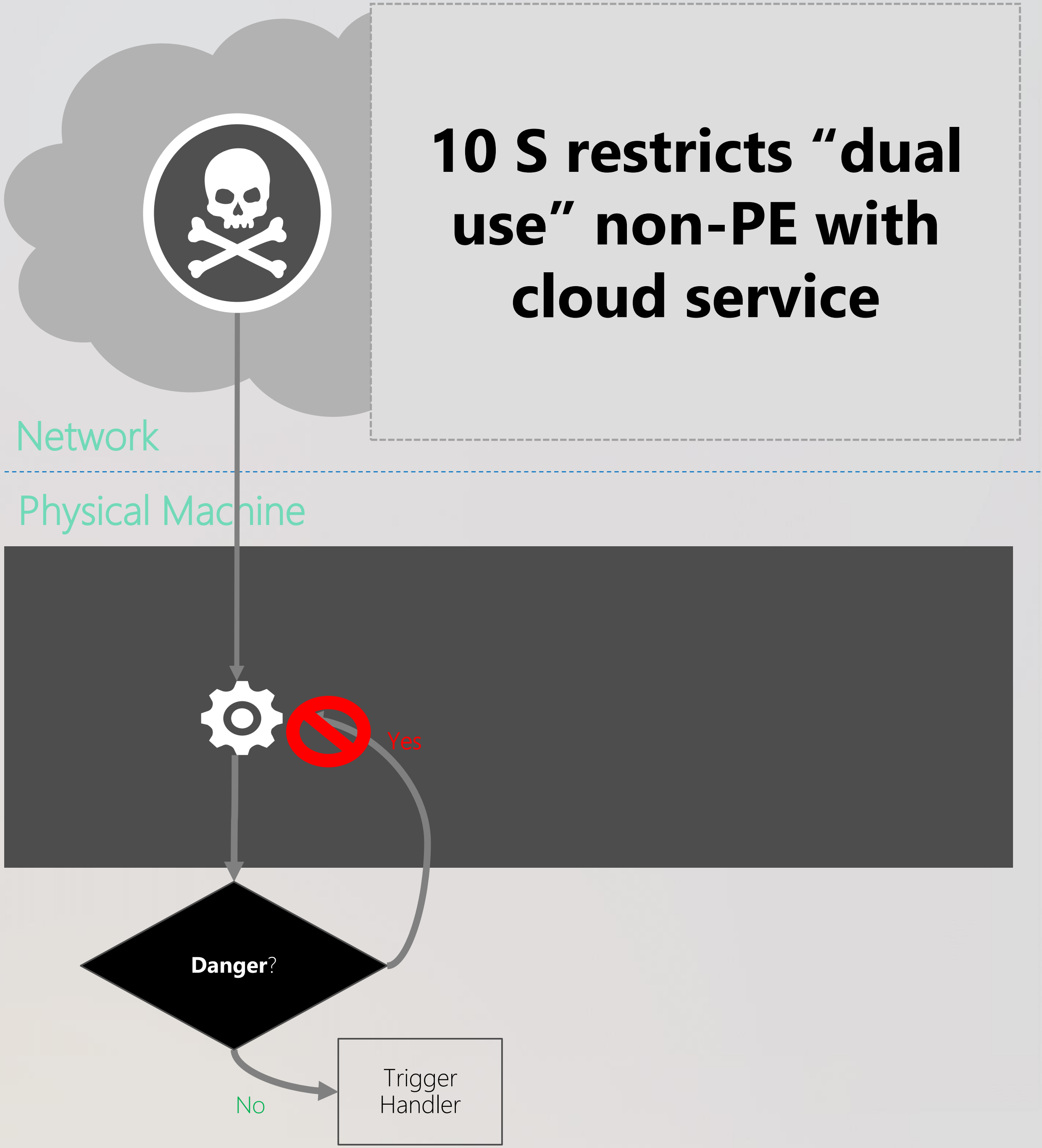
1st Order Code Integrity protection

A “1st order” CI bypass enables a remote attack to trigger initial unsigned code execution

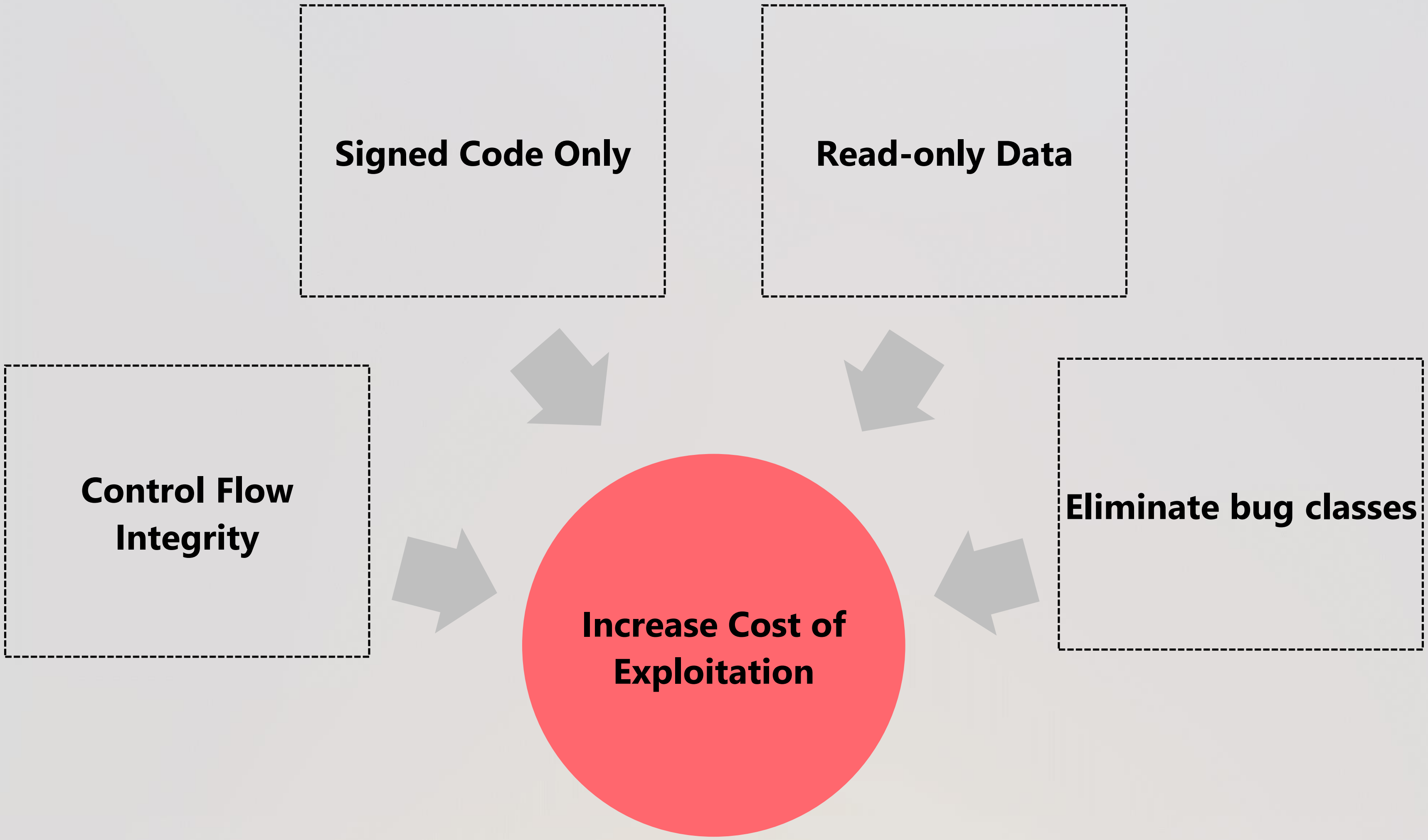
10 S focuses on preventing “1st” order bypasses

A “2nd order” bypass enabled additional unsigned code execution *after* reaching initial code execution

10 S offers less-durable guarantees for “2nd order” bypasses



Exploit mitigation Strategy



Control Flow Integrity Challenges

1. Dangerous call targets
2. Unprotected Stack
3. Data corruption

CFG

First generation CFI in Windows, coarse grained for compatibility and performance

"Export suppression" used to reduce number of call sites in specific processes (example: Microsoft Edge)

Call sites

```
((void(*)(int, int)) funcptr)(0, 1);

obj->method1();
```

Call Targets

```
void function_A(int, int) { ... }
int  function_B(int, int) { ... }
void function_C(Object*)  { ... }

void Object::method1()          { ... }
void Object::method1(int, int)  { ... }
void Object::method2()          { ... }
void Object2::method1()         { ... }
```

Introducing: XFG

Goal: Provide finer-grained CFI in a way that is efficient and compatible

Concept: Restrict indirect transfers through type signature checks

Call Sites

```
((void(*)(int, int)) funcptr)(0, 1);
```

```
obj->method1();
```

Call Targets

```
void function_A(int, int) { ... }
int  function_B(int, int) { ... }
void function_C(Object*)  { ... }
```

```
void Object::method1()          { ... }
void Object::method1(int, int)  { ... }
void Object::method2()          { ... }
void Object2::method1()         { ... }
```

XFG design: basics

Assign a type signature-based tag to each address-taken function

For C-style functions, could be:

hash(type(return_value), type(arg1), type(arg2), ...)

For C++ virtual methods, could be:

hash(*method_name, type(retval), highest_parent_with_method(type(this), method_name), type(arg1), type(arg2), ...)*

Embed that tag immediately before each function so it can be accessed through function pointer

Add tag check to call-sites: fast fail if we run into a tag mismatch

CFG instrumentation: Call Site

```
mov  rax, [rsi+0x98]                ; load target address
call [__guard_dispatch_icall_fptr]
```

Target

```
.align 0x10
function:
     push rbp
     push rbx
     push rsi
     ...
```

xFG instrumentation : Call Site

```
mov  rax, [rsi+0x98]                   ; load target address
mov  r10, 0xdeadbeefdeadbeef           ; load function tag
call [__guard_dispatch_icall_fptr_xfg] ; will check tag
```

Target

```
.align 0x10
dq 0xcccccccccccccccc ; just alignment
dq 0xdeadbeefdeadbeef ; function tag
function:
     push rbp
     push rbx
     push rsi
     ...
```

XFG Security

C-style function pointers can only call address-taken functions with same type signature

Call-site and targets have same number of arguments, arguments and return value have same types

C++ virtual methods can only call methods with same name and type in their class hierarchy

Can't call wrong-type overload methods
Can't call methods from other class hierarchies
Can't call differently-named methods with same type in same hierarchy

This is much stronger than CFG, although it is an over-approximation

While the use of a hash function means there could technically be collisions, that's very unlikely (especially in a useful way) on a ~55 bit hash

Glossing over a lot of implementation details here, but this is the basic idea ☺

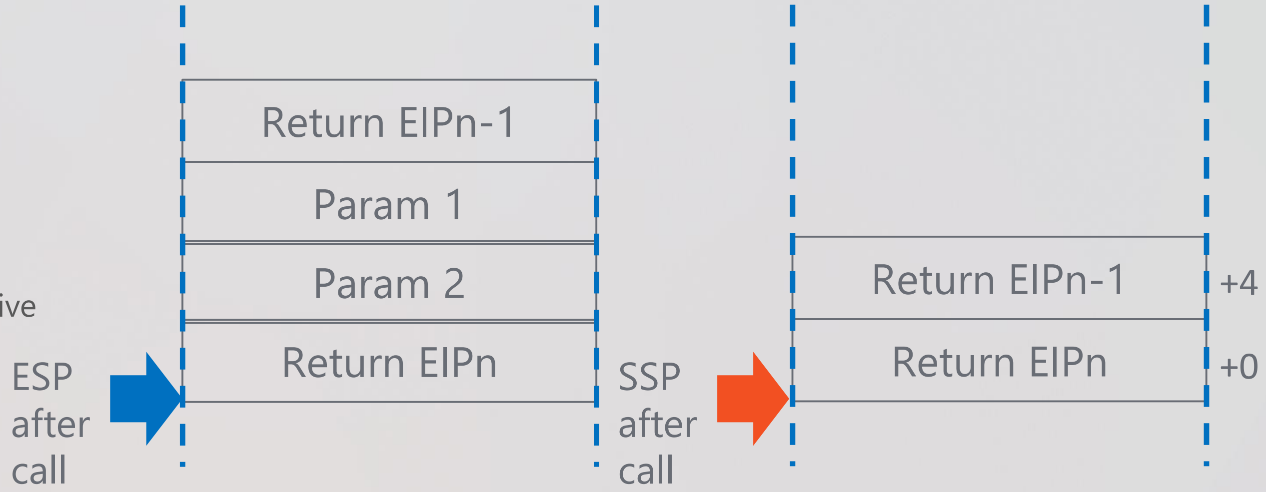
Control Flow Integrity Challenges

1. Dangerous call targets
2. Unprotected Stack
3. Data corruption

Shadow Stack Protection

Initial attempt to implement stack protection in software failed

OSR designed software shadow stack (RFG) did not survive internal offensive research



Stack usage on near CALL

Control-flow Enforcement Technology (CET)

Return address protection via a shadow stack

Hardware-assists for helping to mitigate control-flow hijacking & ROP

Robust against our threat model (assume arbitrary RW)

CET Shadow Stack Flow:

Call pushes return address on both stacks

Ret/ret_imm

pops return address from both stack

Execption if the return addresses don't match

No parameters passing on shadow stack

Control Flow Integrity Challenges

1. Dangerous call targets
2. Unprotected Stack
3. Data corruption

Data Corruption Protection

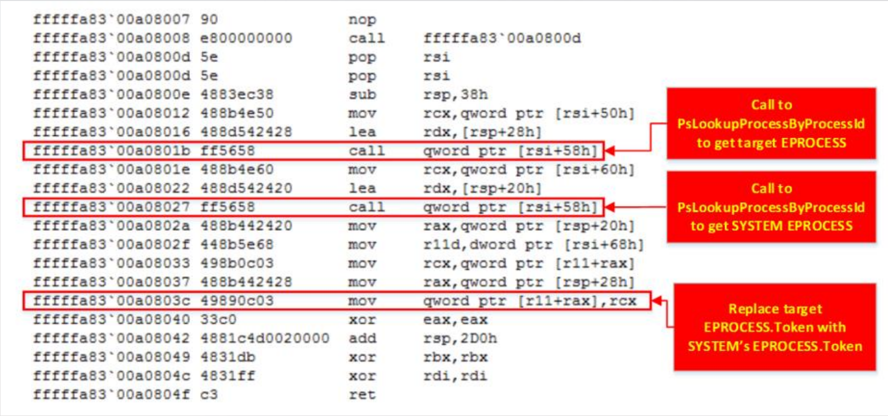
Introducing: Kernel Data Protection

Problem: Kernel exploits in Windows leverage data corruption to obtain privilege escalation

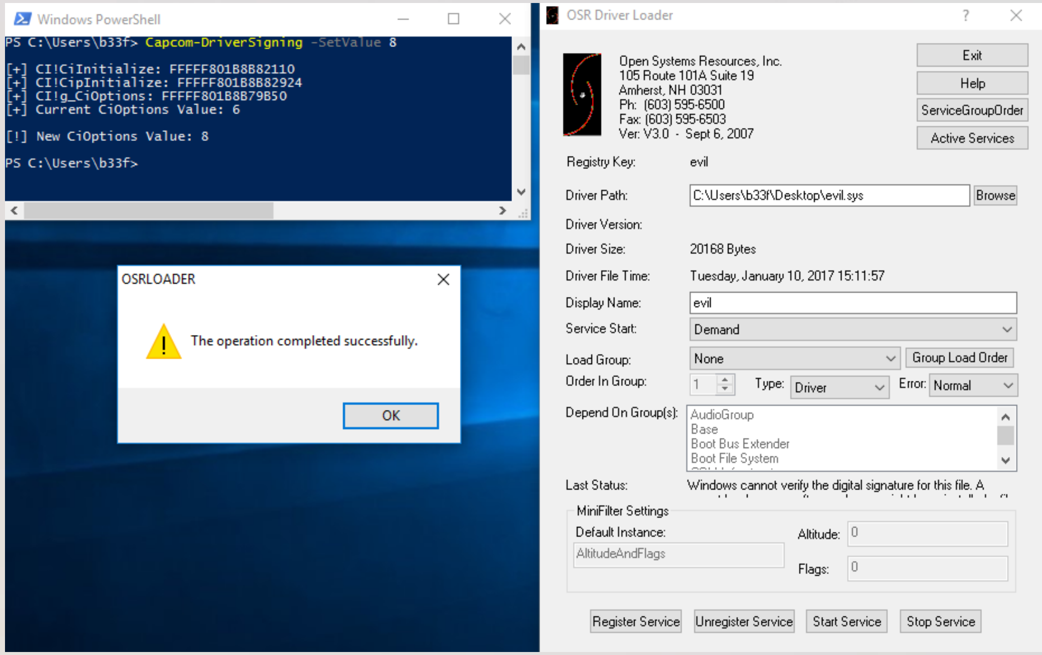
Current State: Hypervisor-based code integrity prevents dynamic code injection and enforces signing policy

Prevent code injection is not enough, kernel has many sensitive data structures

Kernel Data Protection (KDP) uses Secure Kernel to enforce immutability

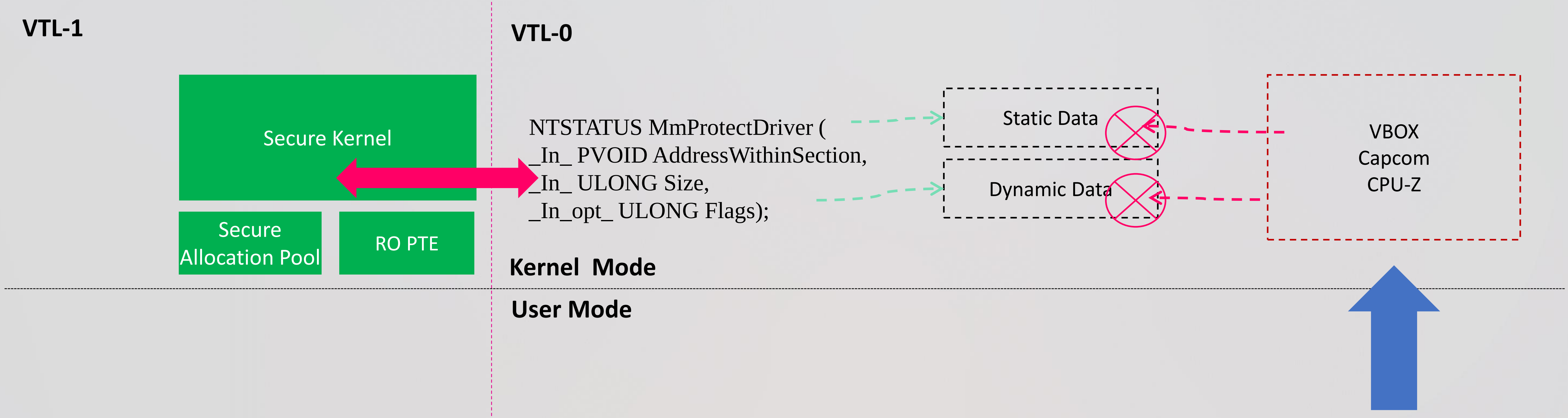


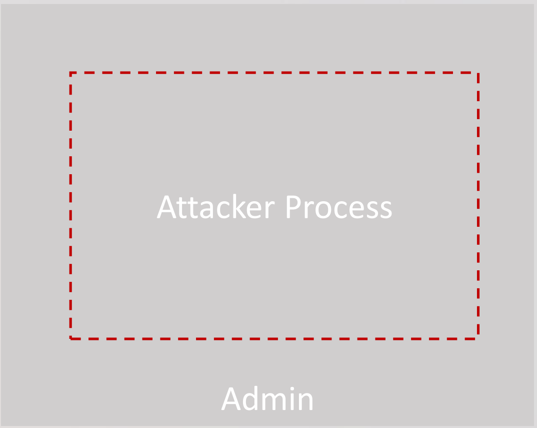
CVE-2016-7256 exploit: Open type font elevation of privilege



Corrupting Code Integrity Globals (credit: FuzzySec)

Data Corruption Protection





Kernel Data Protection:

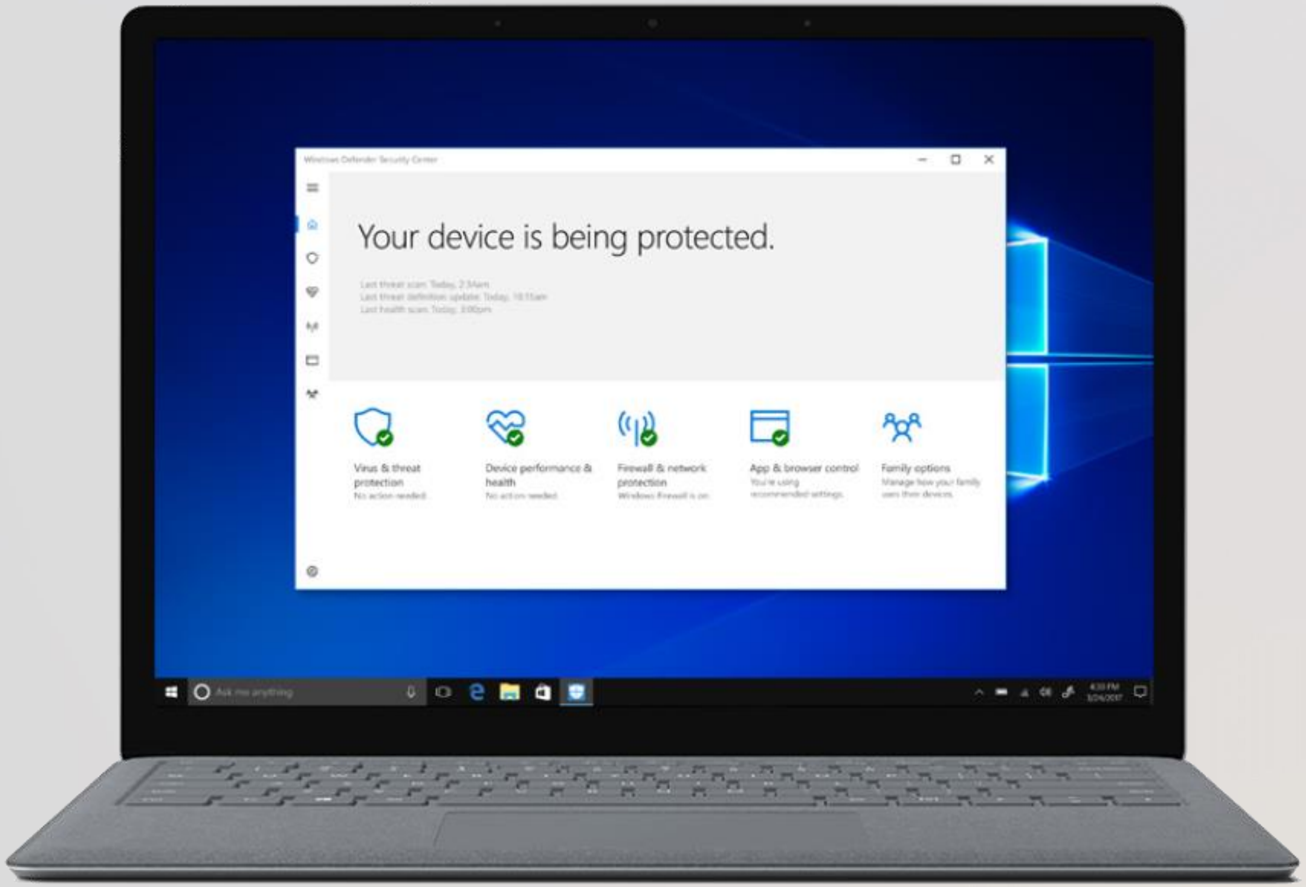
Mechanism to perform read-only pool allocations

RO PTE Hypervisor Protected when VBS is enabled

Validation mechanism to allow callers to detect whether the memory they're referencing is protected pool allocation

All apps and system components have only the privilege they need

“Admin-less” Mode



Introducing: Admin-less

Elevation is been blocked Admin-less S mode

New standard user type can make some device-wide changes

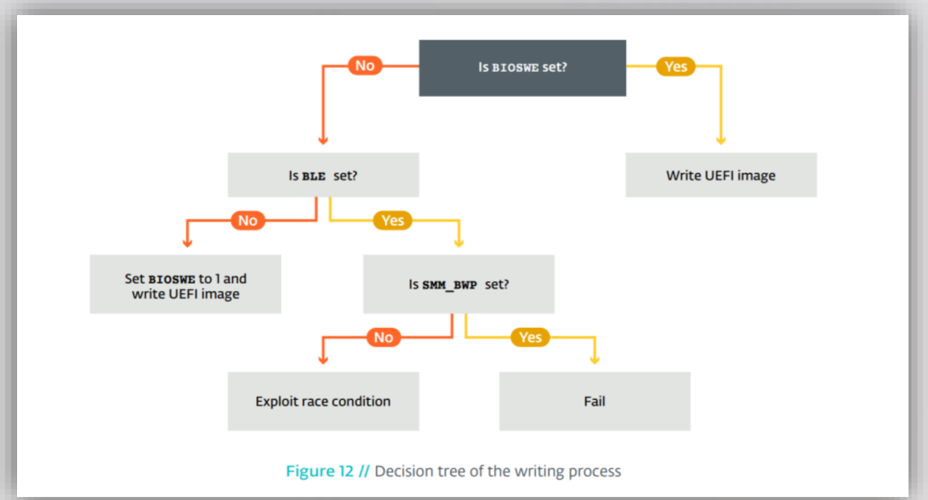
Broker many functions which previously required elevation

Removes ability to read/write/debug, requires unlock or specific developer tools

Standard user security with much less friction

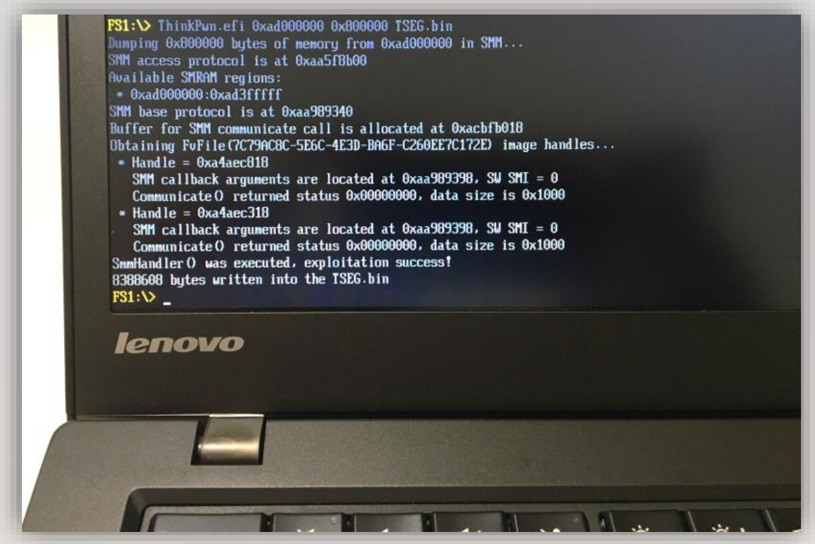
Malicious code cannot persist on a device.

Firmware Security Issues



ESET discovers SEDNIT/APT28 UEFI malware

SMM attacks to bypass VBS



“ThinkPWN” exploit of Lenovo firmware

System Guard with DRTM

Utilize DRTM (Intel, AMD, QC) to perform TCB measurements from a Microsoft MLE

“Assume Breach” of UEFI and measure/seal critical code and data from hardware rooted MLE

Measured values:

- Code integrity Policy
- Hypervisor, kernel hashes
- UEFI Vars
- Etc.…

Zero Trust

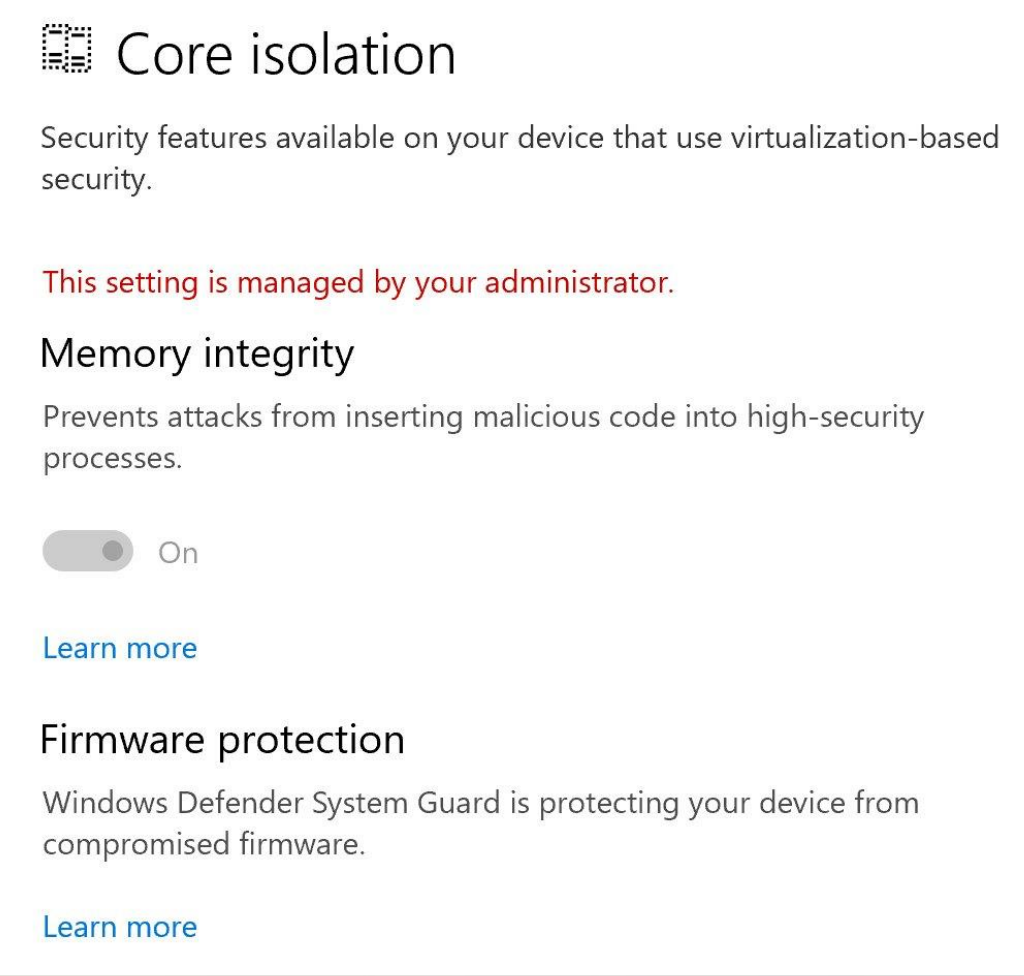
Measurements of key properties available in PCRs and TCG logs

Attest TCB components through System Guard runtime attestation + Microsoft Conditional Access + WDATP

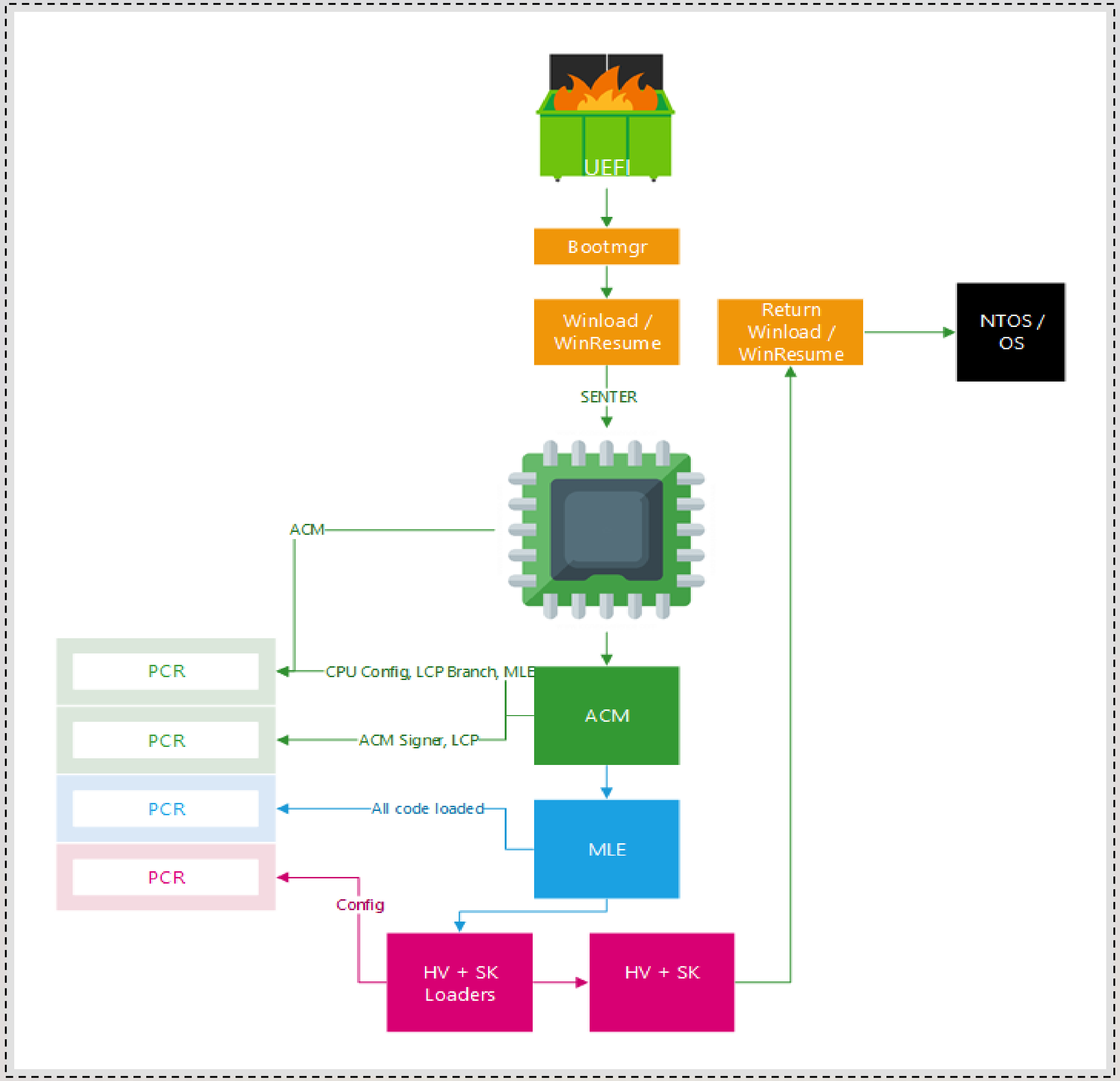
SMM Attacks

Can be used to tamper HV and SK post-MLE

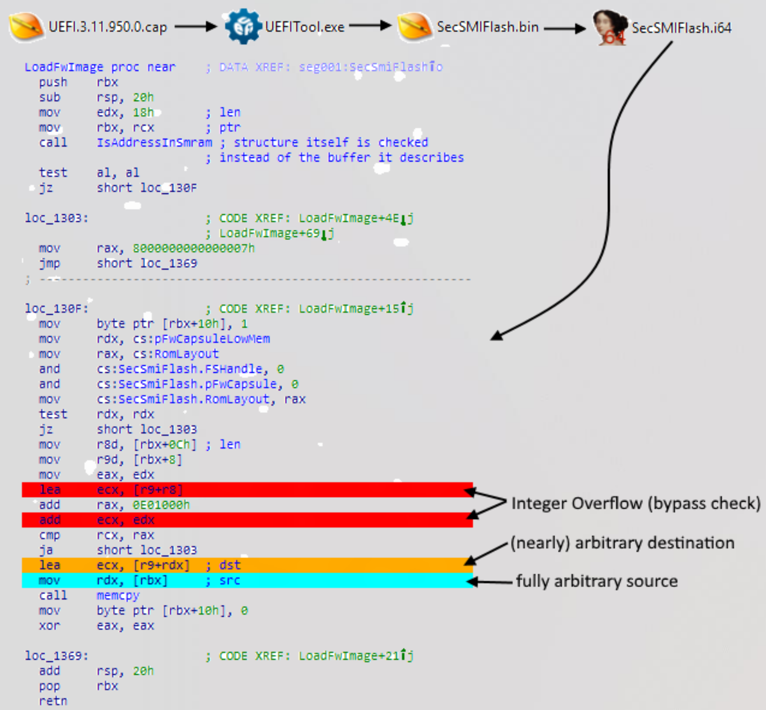
SMM paging protections + attestation on roadmap

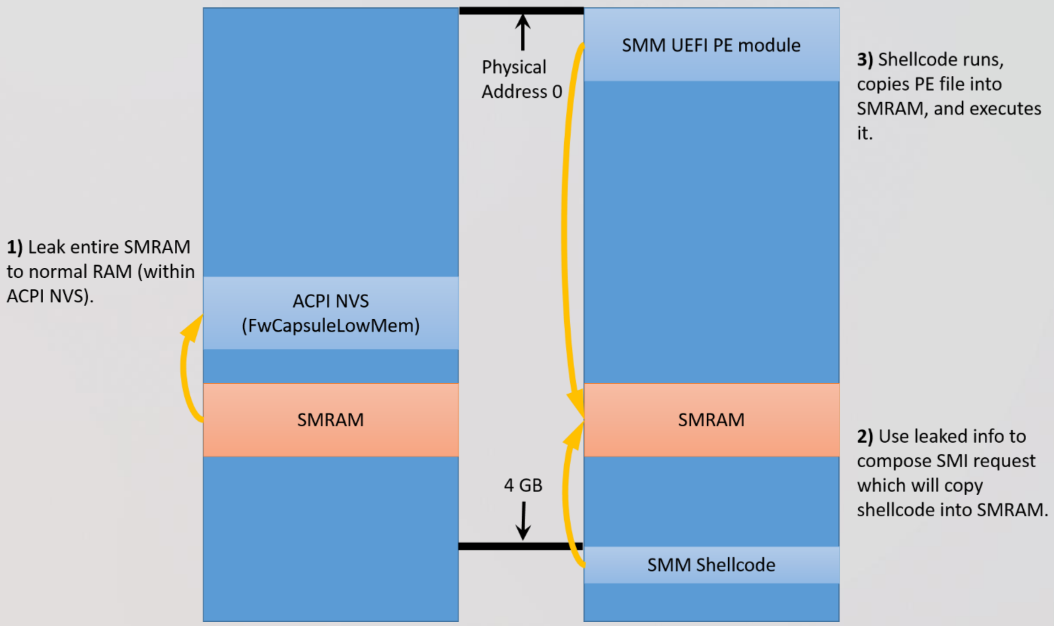


Improving Boot Security



Improving Boot Security





System Guard with DRTM

External researchers and OSR REDTEAM highlighted SMM risks for DRTM and VBS

Arbitrary code execution in SMRAM can be used to defeat Hypervisor

Malicious code running in SMM is difficult to detect

SMM vulnerabilities used in OSR REDTEAM reported to Lenovo

Protecting SMM

Mitigating SMM exploitation

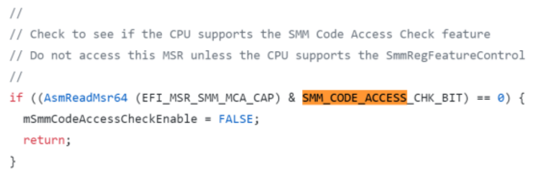
Intel Runtime BIOS resilience provides the following security properties for SMM:

SMM entry point locked down

All code within SMM locked down

Memory map and page properties locked down

OS and HV memory not directly accessible from SMM



```
//
// Check to see if the CPU supports the SMM Code Access Check feature
// Do not access this MSR unless the CPU supports the SmmRegFeatureControl
//
if ((AsmReadMsr64 (EFI_MSR_SMM_MCA_CAP) & SMM_CODE_ACCESS_CHK_BIT) == 0) {
  mSmmCodeAccessCheckEnable = FALSE;
  return;
}
```

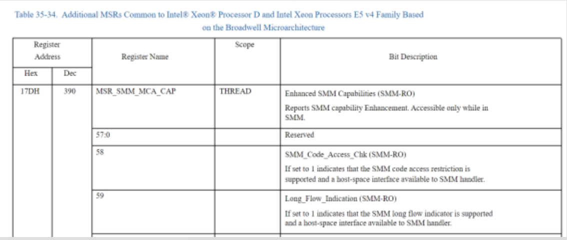
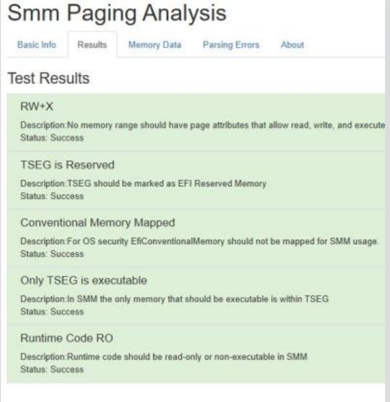


Table 35-34. Additional MSRs Common to Intel® Xeon® Processor D and Intel Xeon Processors E5 v4 Family Based on the Broadwell Microarchitecture

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
17DH	390	MSR_SMM_MCA_CAP	THREAD	Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.
		57:0		Reserved
		58		SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.
		59		Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.



Smm Paging Analysis

Basic Info | Results | Memory Data | Parsing Errors | About

Test Results

RW+X
Description:No memory range should have page attributes that allow read, write, and execute
Status: Success

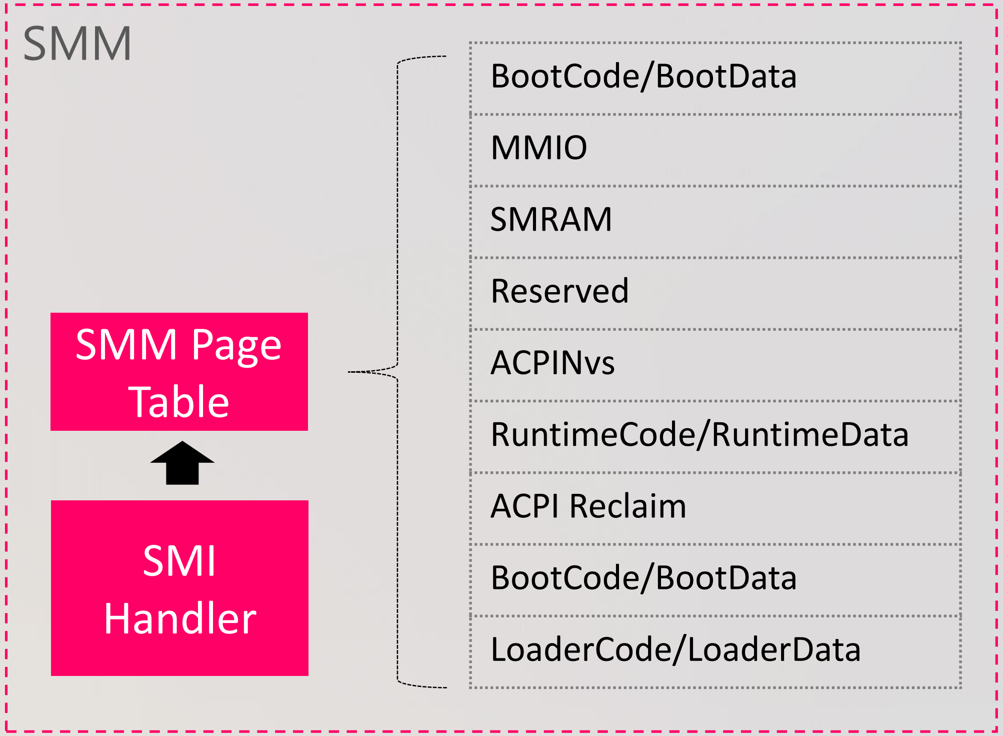
TSEG is Reserved
Description:TSEG should be marked as EFI Reserved Memory
Status: Success

Conventional Memory Mapped
Description:For OS security EfiConventionalMemory should not be mapped for SMM usage.
Status: Success

Only TSEG is executable
Description:In SMM the only memory that should be executable is within TSEG
Status: Success

Runtime Code RO
Description:Runtime code should be read-only or non-executable in SMM
Status: Success

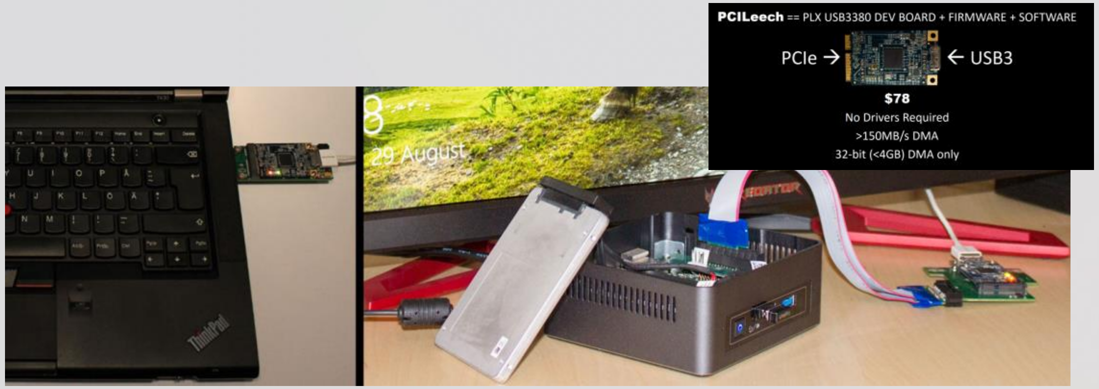
SMM Paging Audit



SMM Protection

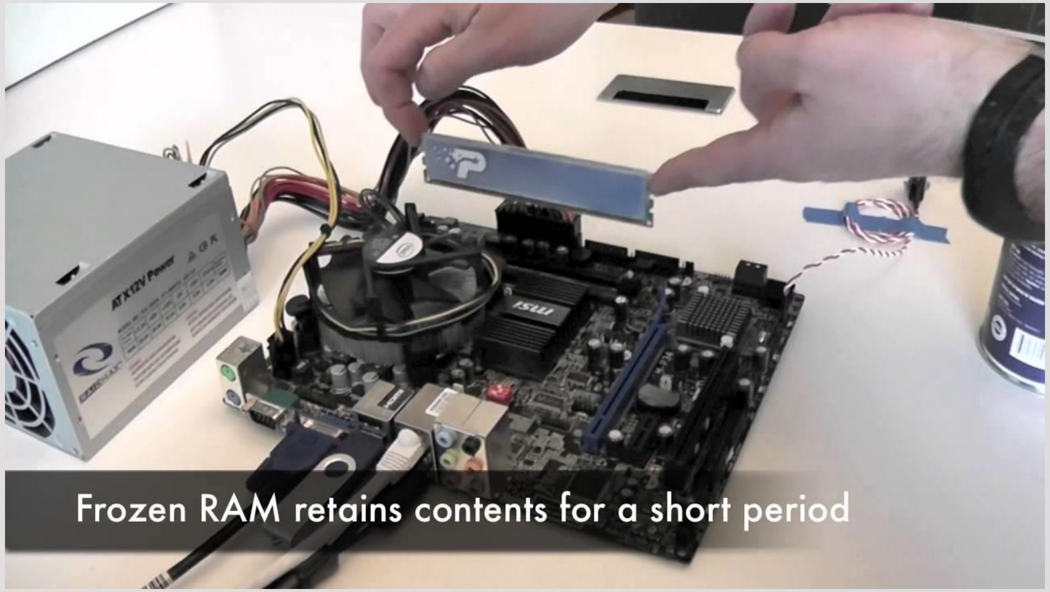
Attackers with casual physical access cannot modify data or code on the device.

Increasing Physical Attacks



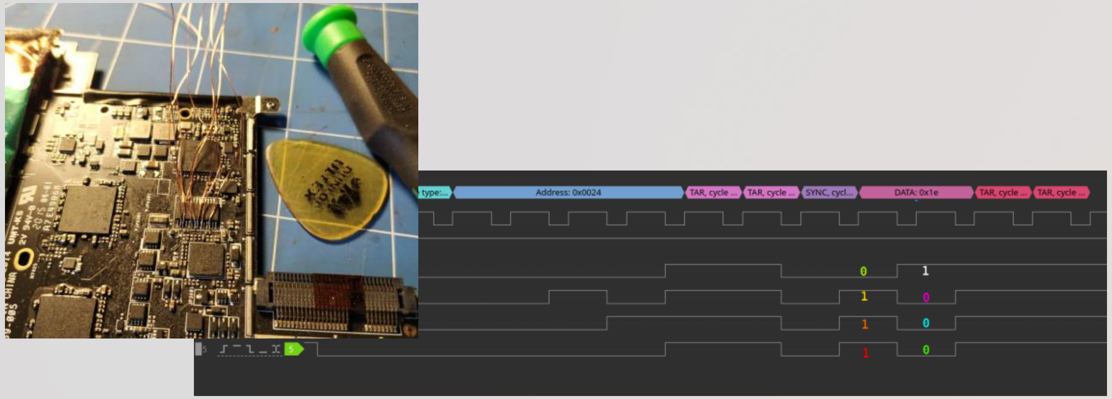
DMA Attacks with PCILeech

Sources: 1, 2



Bitlocker Cold Boot Attacks

Sources: 1



LPC/SPI TPM VMK Key Extraction with Logic Analyzer

Sources: 1, 2, 3

Windows DMA protection

Security Goals

Prevent "evil cleaner" drive by physical attacks from malicious DMA attacks

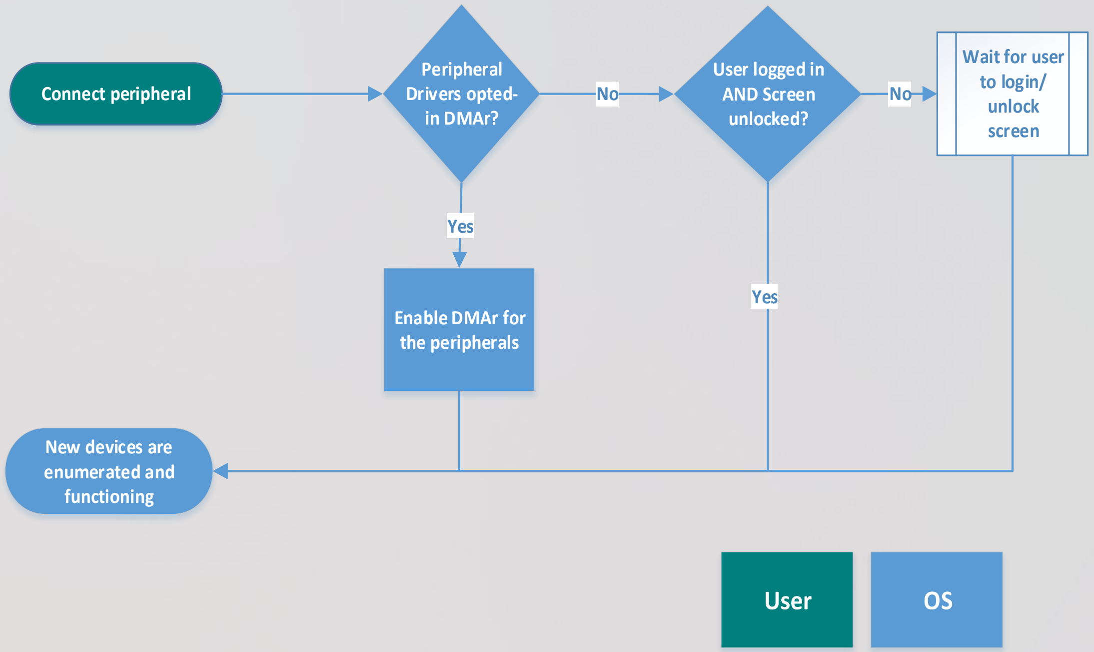
Design Details

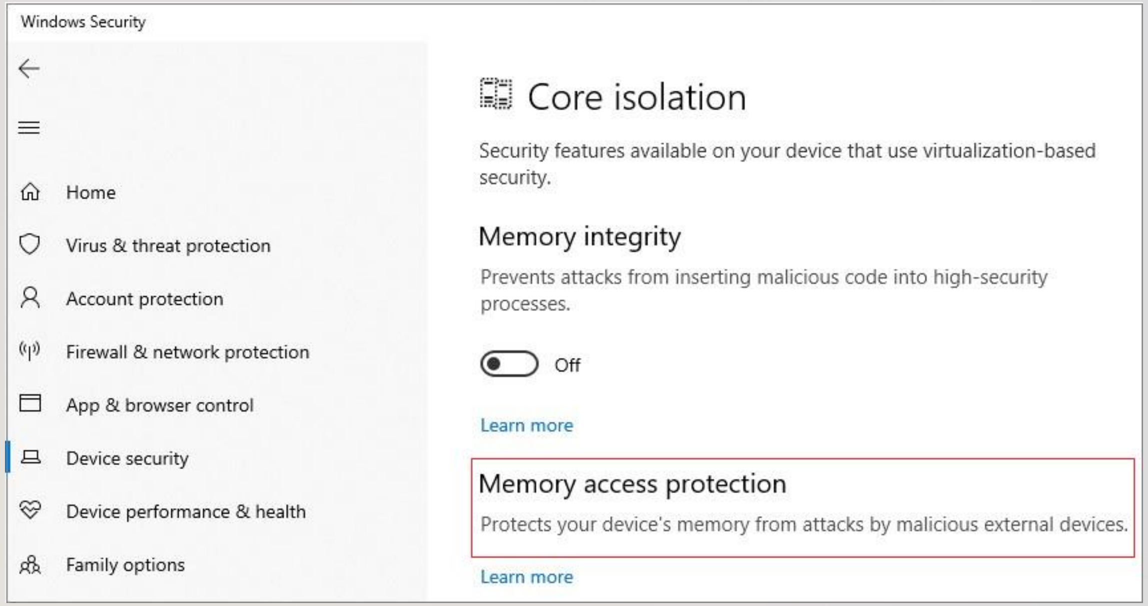
Use IOMMU to block newly attached Thunderbolt™ 3 devices from using DMA until a user is logged in

[UEFI can enable IOMMU an BME]in early boot until Windows boots (See [Project Mu])

Automatically enable DMA remapping with compatible device drivers

In future releases, we are looking to harden protection on all external PCI ports and cross-silicon platforms





Windows Data Protection Under Lock

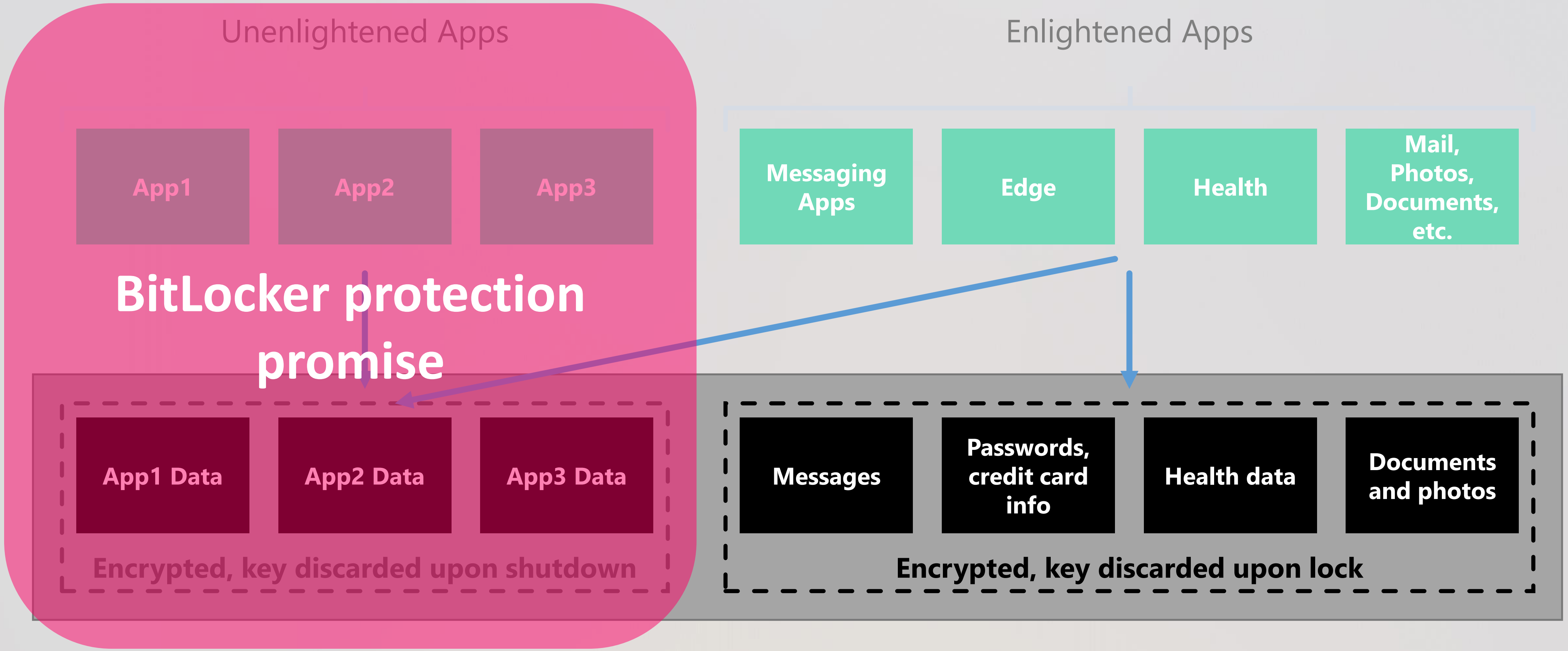
Locked device

Encryption key is removed from memory

Unlocked device

Encryption key is recomputed using user entropy

Per-file encryption provides a second layer of protection at rest
Key is derived from user secret (Hello, Biometric)


Unenlightened Apps
Enlightened Apps
App1
App2
App3
Messaging Apps
Edge
Health
Mail, Photos, Documents, etc.
BitLocker protection promise
App1 Data
App2 Data
App3 Data
Encrypted, key discarded upon shutdown
Messages
Passwords, credit card info
Health data
Documents and photos
Encrypted, key discarded upon lock

User identities cannot be compromised, spoofed, or stolen.

Improving Identity Security

Windows Hello and NGC

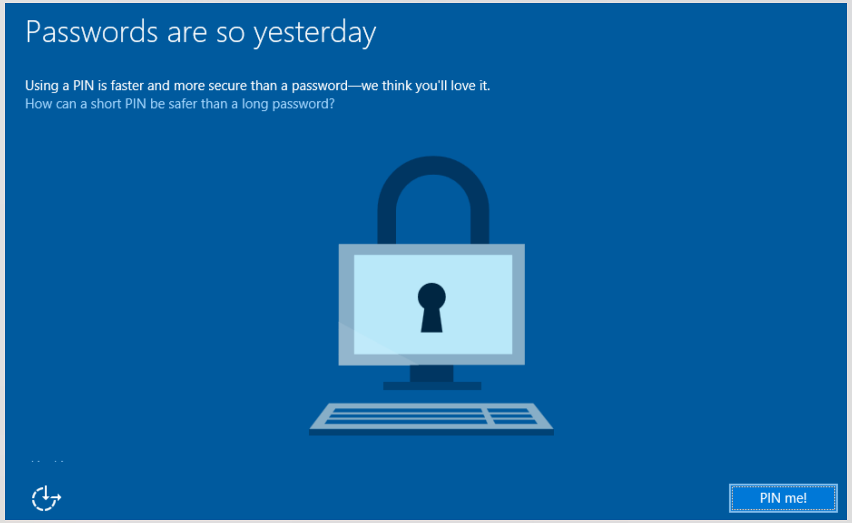
Offers biometric authentication and hardware backed key storage
PIN vulnerable to input attacks from malicious admin

Improving Identity Security

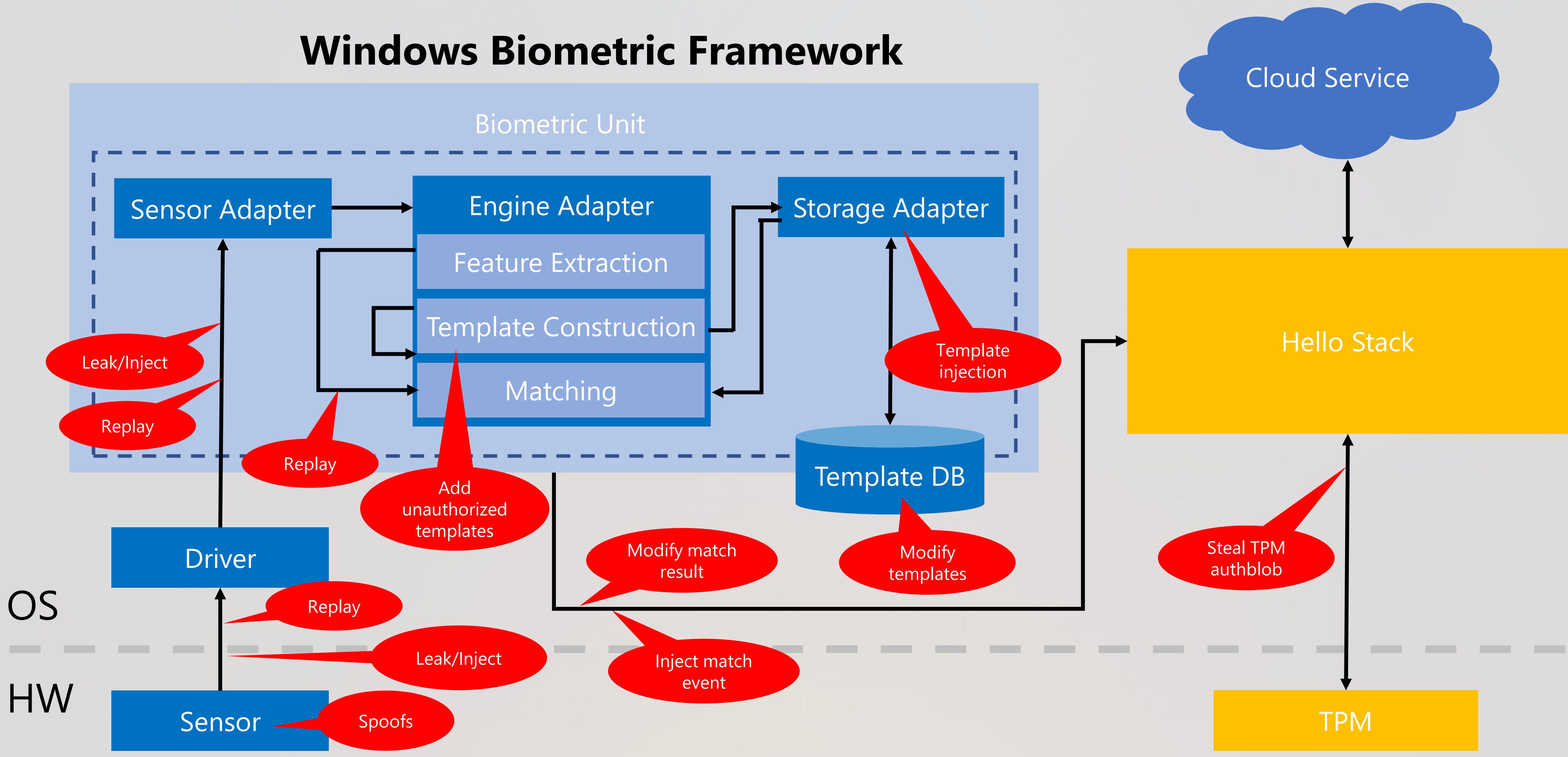
Future version of Windows include biometric hardening enabled through virtualization

Biometric hardening of the data path using virtualization

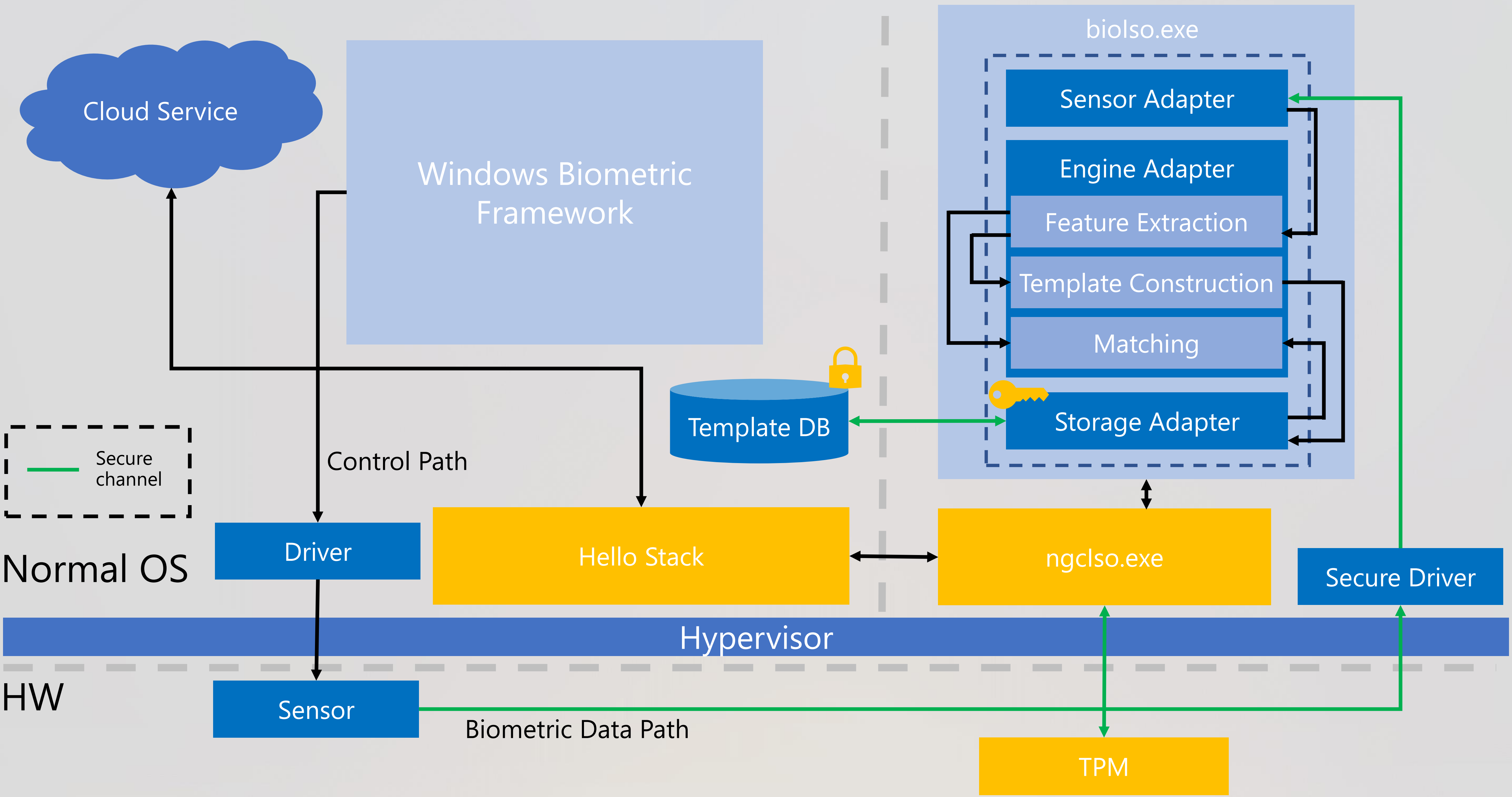
Hardening of credential release



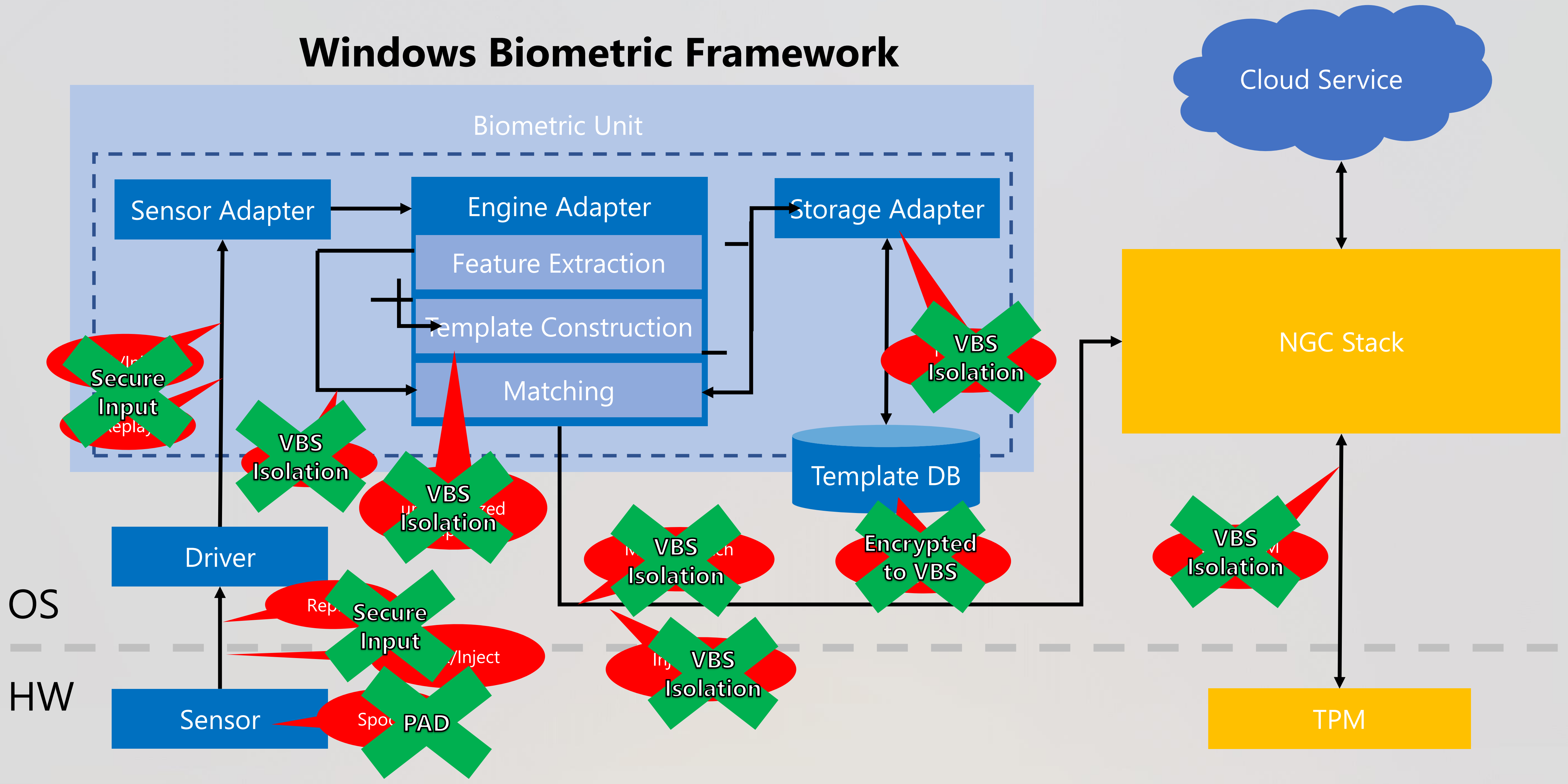
Windows Biometric Framework



Windows Hello Attack Surface



Windows Hello Attack Surface



Beyond Passwords

A web without passwords

Staying secure on the web is more important than ever. We trust web sites to process credit card numbers, save addresses and personal information, and even to handle sensitive records like medical information. All this data is protected by an ancient security model—the password. But passwords are difficult to remember, and are fundamentally insecure—often re-used, and vulnerable to phishing and cracking.

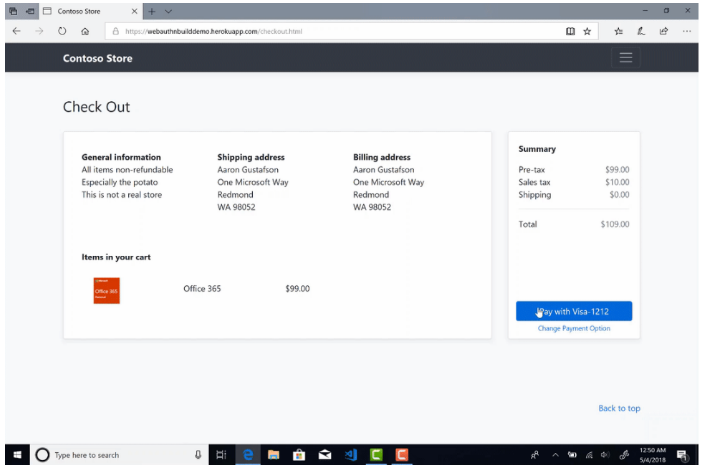
For these reasons, Microsoft has been leading the charge towards a world without passwords, with innovations like Windows Hello biometrics and pioneering work with the FIDO Alliance to create an open standard for passwordless authentication – Web Authentication.

We started this journey in 2016, when we shipped the industry's first preview implementation of the Web Authentication API in Microsoft Edge. Since then, we have been updating our implementation to as we worked with other vendors and the FIDO alliance to develop the standard. In March, the FIDO Alliance announced that the Web Authentication APIs have reached Candidate Recommendation (CR) status in the W3C, a major milestone for the maturity and interoperability of the specification.

Authenticators in Microsoft Edge

Beginning with build 17723, Microsoft Edge supports the CR version of Web Authentication. Our implementation provides the most complete support for Web Authentication to date, with support for a wider variety of authenticators than other browsers.

Windows Hello allows users to authenticate without a password on any Windows 10 device, using biometrics—face and fingerprint recognition—or a PIN number to sign in to web sites. With Windows Hello face recognition, users can log in to sites that support Web Authentication in seconds, with just a glance.



FIDO Alliance and W3C Achieve Major Standards Milestone in Global Effort Towards Simpler, Stronger Authentication on the Web

April 10, 2018

With support from Google Chrome, Microsoft Edge and Mozilla Firefox, FIDO2 Project opens new era of ubiquitous, phishing-resistant, strong authentication to protect web users worldwide

MOUNTAIN VIEW, Calif., and https://www.w3.org/ — April 10, 2018 – The FIDO Alliance and the World Wide Web Consortium (W3C) have achieved a major standards milestone in the global effort to bring simpler yet stronger web authentication to users around the world. The W3C has advanced Web Authentication (WebAuthn), a collaborative effort based on Web API specifications submitted by FIDO to the W3C, to the Candidate Recommendation (CR) stage. The CR is the product of the Web Authentication Working Group, which is comprised of representatives from over 30 member organizations. CR is a precursor to final approval of a web standard, and the W3C has invited online services and web app developers to implement WebAuthn.

WebAuthn defines a standard web API that can be incorporated into browsers and related web platform infrastructure which gives users new methods to securely authenticate on the web, in the browser and across sites and devices. WebAuthn has been developed in coordination with FIDO Alliance and is a core component of the FIDO2 Project along with FIDO's Client to Authenticator Protocol (CTAP) specification. CTAP enables an external authenticator, such as a security key or a mobile phone, to communicate strong authentication credentials locally over USB, Bluetooth or NFC to the user's internet access device (PC or mobile phone). The FIDO2 specifications collectively enable users to authenticate easily to online services with desktop or mobile devices with phishing-resistant security.

Violations of promises are observable.

Tamper Evident Windows

Platform Tamper Detection for Windows

Spanning device boot to ongoing runtime process tampering

Designed for remote assessment of device health

Platform approach to benefit a variety of 3rd parties and scenarios

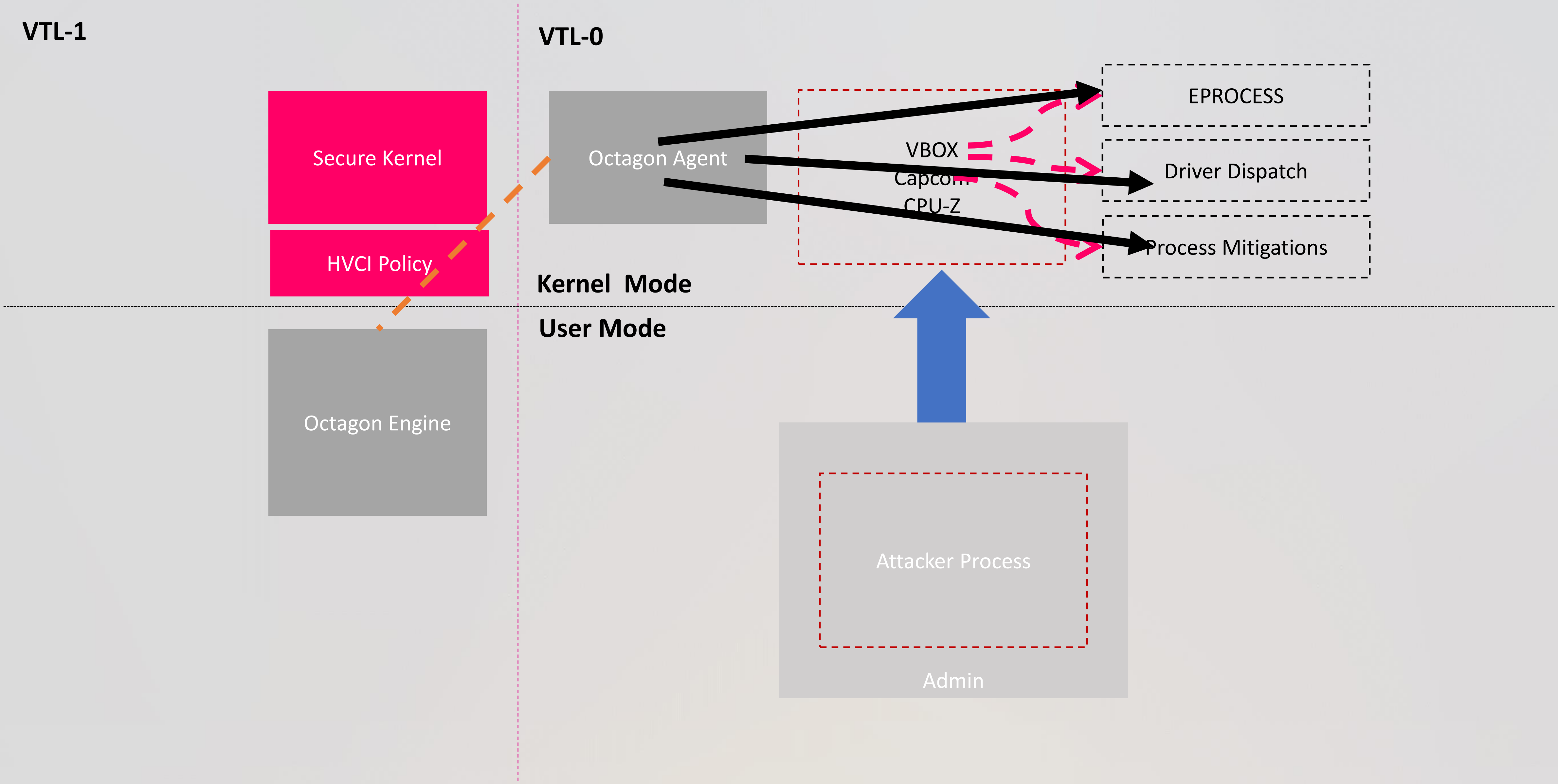
Hardware rooted device trust

Leverage the VBS security boundary to raise the bar on anti-tampering

Challenging to build tamper detection schemes on top of Windows

Extensible platform component that can be used via forthcoming public API

Photo Source 1, 2, 3


VTL-1
VTL-0
Secure Kernel
HVCI Policy
Octagon Agent
VBOX
Capcom
CPU-Z
EPROCESS
Driver Dispatch
Process Mitigations
Kernel Mode
User Mode
Octagon Engine
Attacker Process
Admin

Closing

Windows needs the community

Platform features rapidly changing

Windows is evolving quickly to increase protections against new attacks

Aspirational goals to provide strong guarantees across a growing threat model

Researchers and Community help us improve

Programs such as bug and mitigation bounty are critical

We want to work together with research communities in China and beyond to learn more about current and future attacks