Inside the Octagon

Analyzing System Guard Runtime Attestation

Alex Ionescu
CrowdStrike

David Weston
Microsoft

Octagon Team
Microsoft

Shout outs

System Guard is the result of many many people’s hard work!

The entire ”Octagon” team

The DRTM team across Windows

Enclaves and VBS team

Microsoft Core UEFI team

Intel, AMD, QC teams

Why attestation?

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.

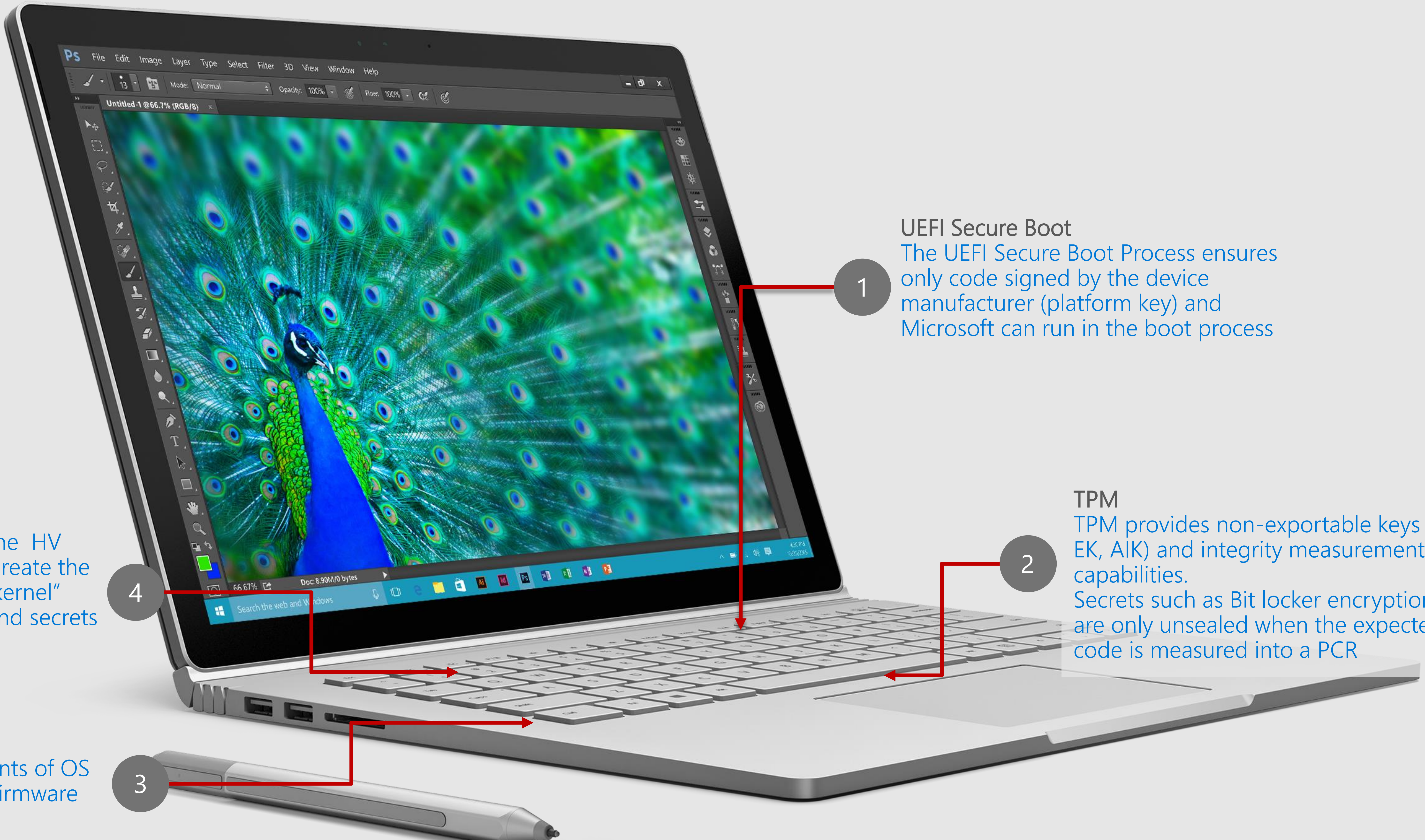
Violations of promises are observable.

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

Trust Challenges

1. Admin Users Have Complete Control
2. Hardware Diversity Creates Inconsistency
3. OS Configuration
4. Security Boundaries
5. Control Plane

Hardware Anchored Trust



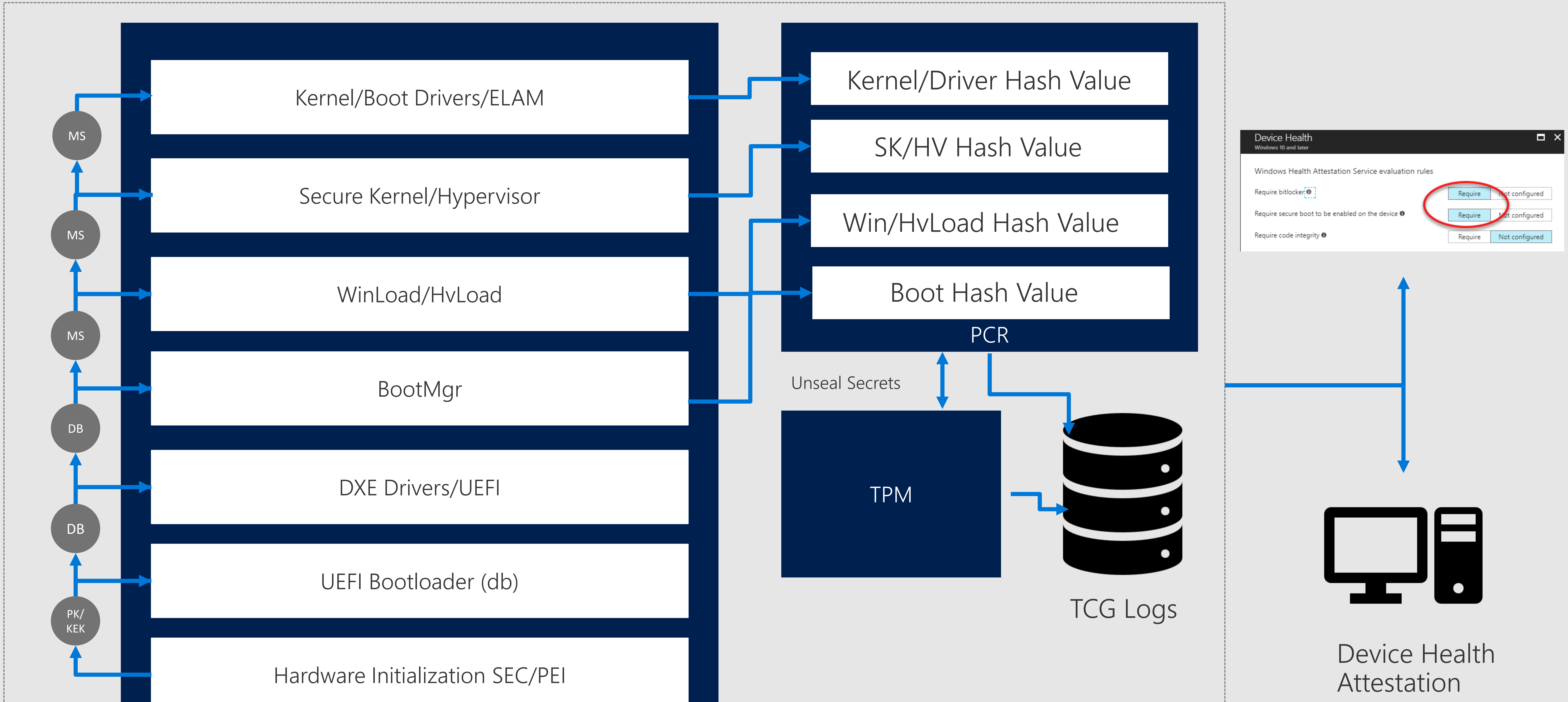
UEFI Secure Boot
The UEFI Secure Boot Process ensures only code signed by the device manufacturer (platform key) and Microsoft can run in the boot process

TPM
TPM provides non-exportable keys (e.g. EK, AIK) and integrity measurement capabilities.
Secrets such as Bit locker encryption key are only unsealed when the expected code is measured into a PCR

Intel VT-x and AMD-v
Virtualization extensions allow the HV launched by the bootloader to create the page table segmented "Secure kernel" which is used to isolated code and secrets from the normal world (VTL0)

Intel TXT and AMD SKINIT
DRTM augments UEFI secure by decoupling integrity measurements of OS and HV components from UEFI firmware

Attesting to Trust



Device Health
Windows 10 and later

Windows Health Attestation Service evaluation rules

Require bitlocker	[✓ Require]	Not configured
Require secure boot to be enabled on the device	[✓ Require]	Not configured
Require code integrity	Require	[✓ Not configured]

See here for additional details

Limitations of current trust model

Secure boot and static root of trust (SRTM)

Strong dependency on the security UEFI firmware

Widespread issues with OEM firmware are used to undermine TCB

High-integrity attestation is at boot time

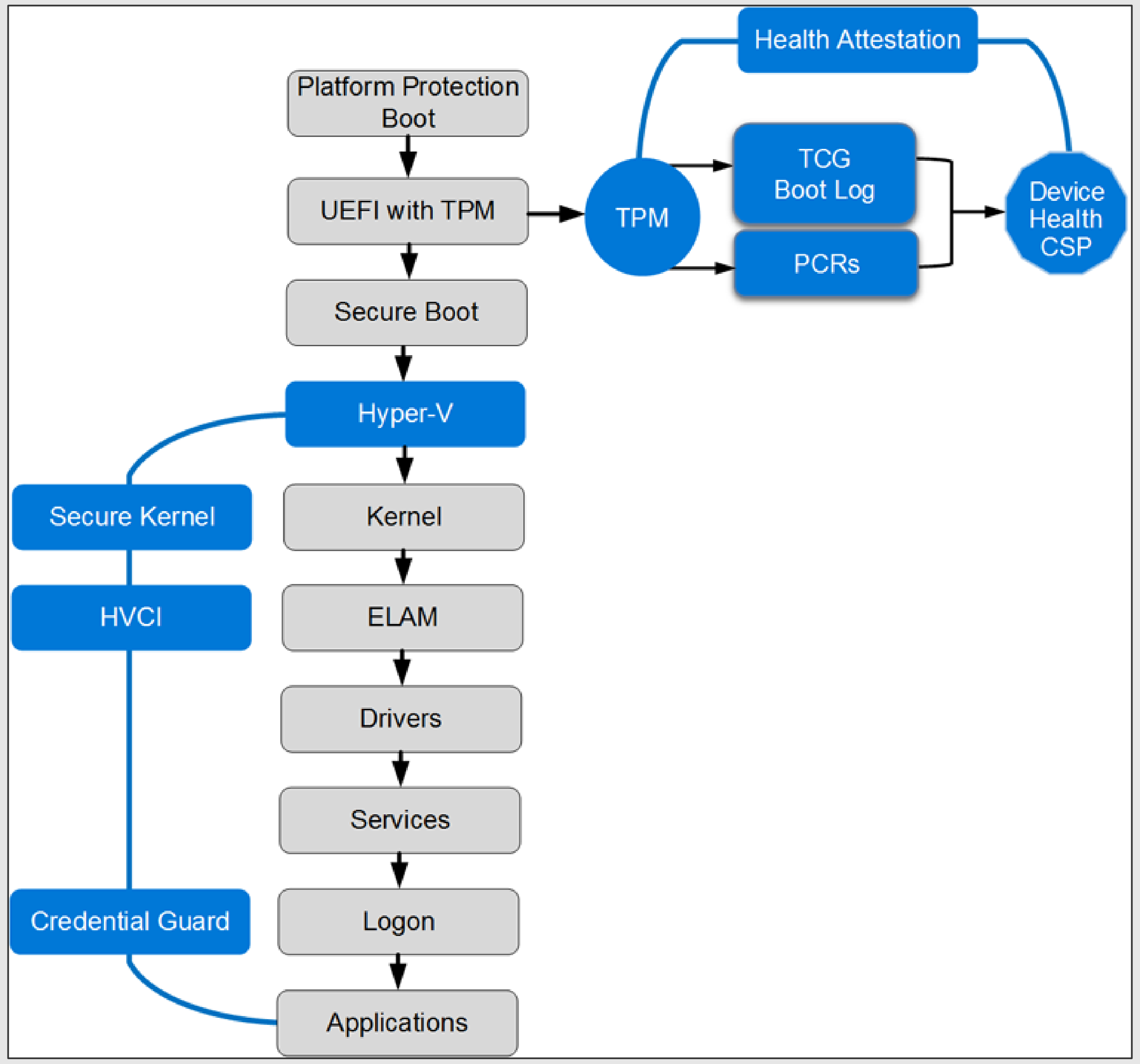
DHA only measures and attests to the integrity of system bring up

Simple runtime attacks do not impact health state

Limited runtime solutions

EDR/AV can identify runtime attacks limited to integrity (software)

PatchGuard/HyperGuard great but not easily extensible



Building a better model

What is System Guard Runtime Attestation?

Platform Tamper Detection for Windows

Spanning device boot to ongoing runtime tampering

Designed for remote assessment of device health

Platform-driven approach to benefit a variety of 3rd party customers and scenarios

Why are we building it?

Challenging to build tamper detection schemes on top of Windows

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

Evolving security approach to keep Windows users and their data safe

How Does it Work?

Boot time attestation assures integrity of Windows & environment where tamper detection sensor runs

Sensor runs in isolated VBS environment constantly monitoring User/Kernel process space for signs of tampering

Relying parties calling into runtime attestation APIs in normal world get integrity protected attestation reports

Relying party backend verifies attestation report to ascertain device tamper status

Improving Boot Trust

System Guard with DRTM

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

"*Assume Breach*" of UEFI and measure it from a hardware-rooted MLE

DRTM + Remote Attestation

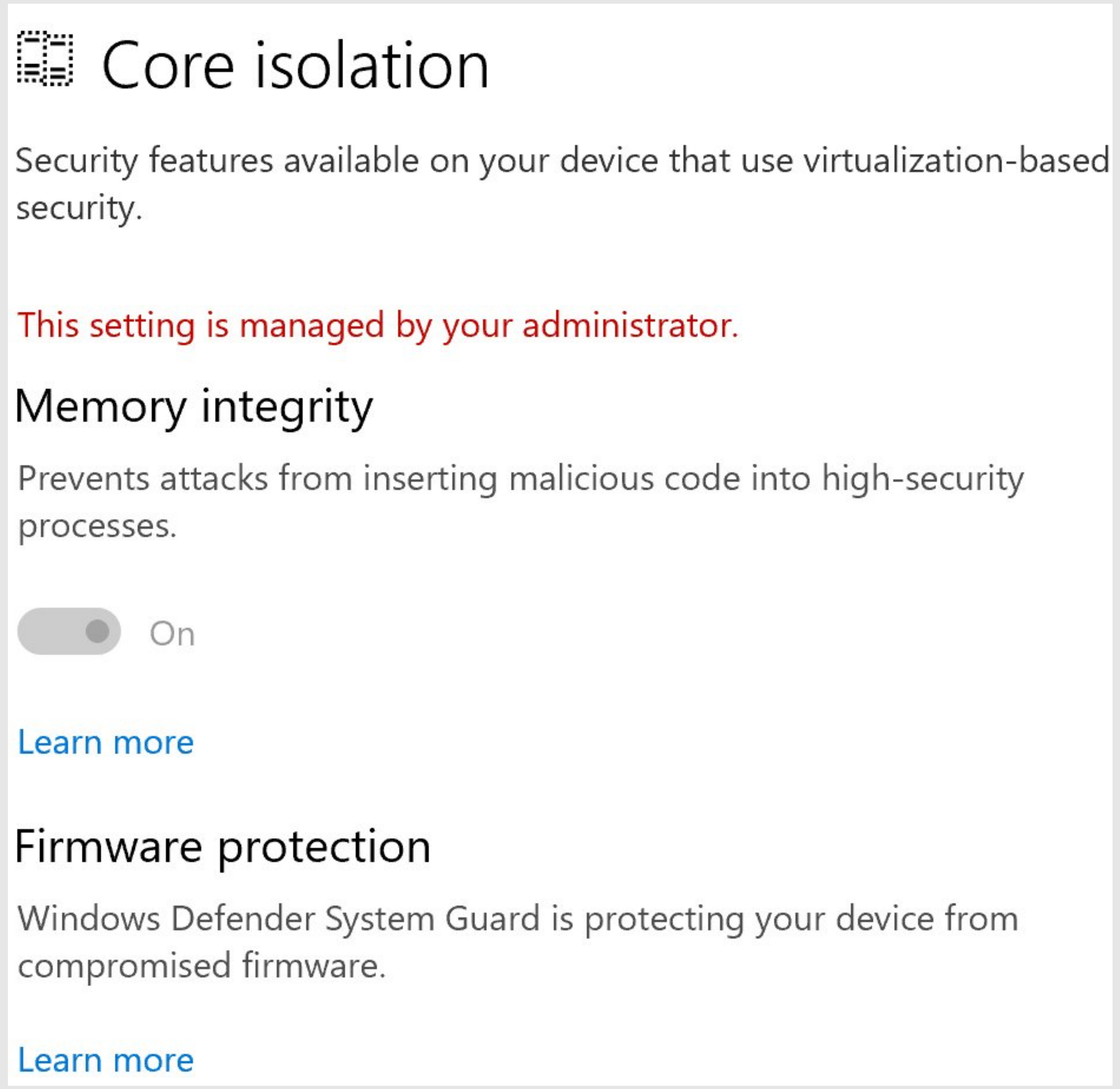
Measurements of key properties available in PCRs

Attest TCB components w/ System Guard runtime attestation + Azure Conditional Access + WDATP

SMM Attacks

Can be used to tamper HV and SK post-MLE

SMM paging protections + attestation on roadmap



Core isolation

Security features available on your device that use virtualization-based security.

This setting is managed by your administrator.

Memory integrity

Prevents attacks from inserting malicious code into high-security processes.

On

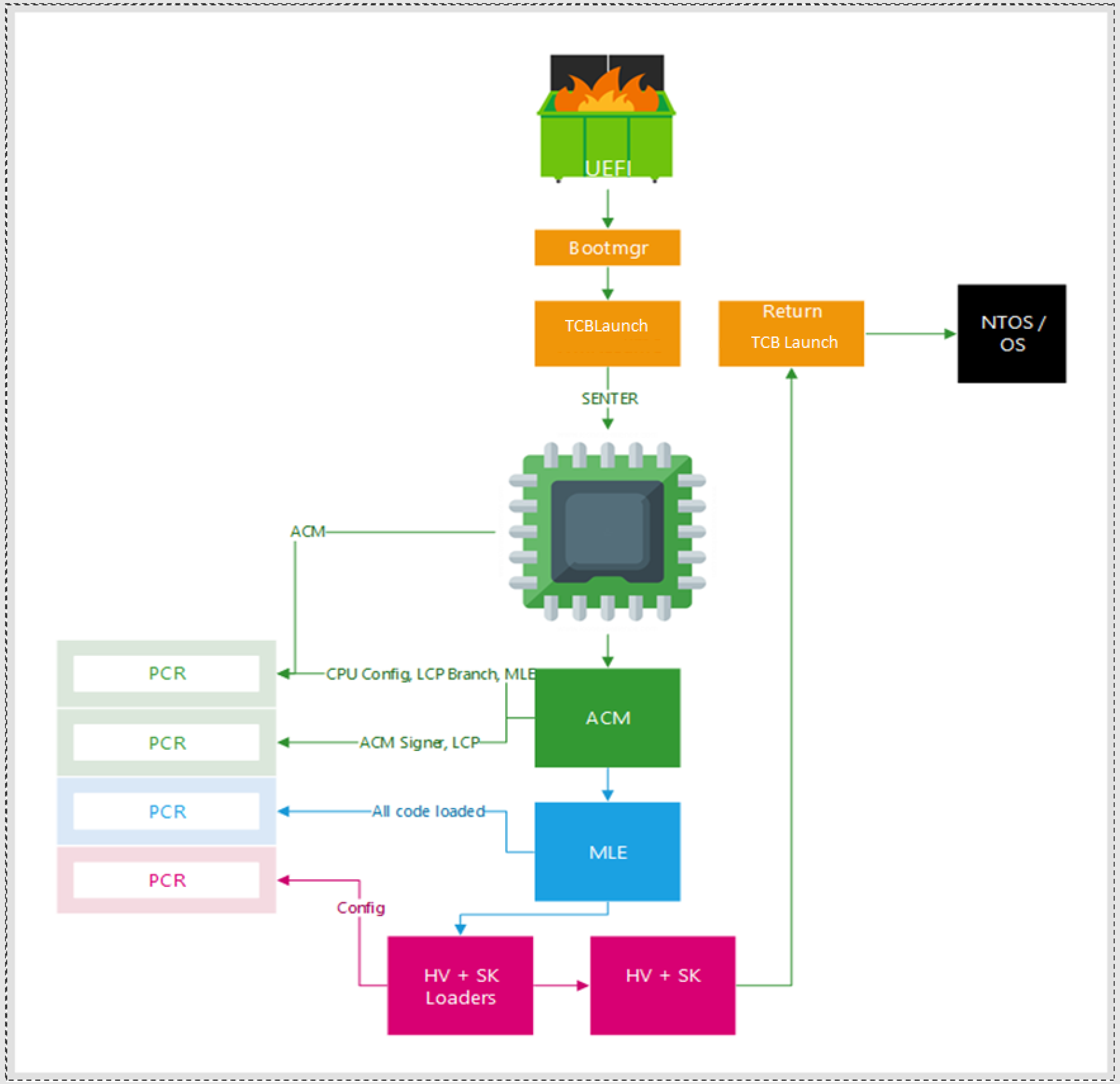
Learn more

Firmware protection

Windows Defender System Guard is protecting your device from compromised firmware.

Learn more

System Guard - DRTM



Regular Launch

Boot manager (**`BootMgr`**) is responsible for initial platform data collection

Parses Secure Boot policy, ACPI/UEFI runtime tables and parts of registry

Initializes BitLocker, displays boot/recovery menu

Launches Windows loader, resume, or other boot app (such as **`MemTest`**)

Windows loader (**`WinLoad`**) does the remainder of the data collection

Constructs large structure (*loader parameter block*) to pass to the kernel

In the case of Hyper-V load, passes data to **`HvLoader`** instead

`HvLoader` initializes Hyper-V, Secure Kernel (for VSM/VBS), then returns

Entire process executes at VTL 0

Secure Boot is the only root chain of trust – TPM used only for BitLocker

Secure Launch – SINIT & ACM

Boot manager now checks for Intel TXT support & enablement

If present, TXT heap is allocated, GETSEC initialized, MTRRs configured

`TxtLoadAcmModule` will start the SINIT Authenticated Code Module

Authenticated Code Module provided by Intel for each CPU SKU

Lives in **`\Windows\System32\acm.bin`**, or more likely, in **`TcbRes.wim`**

WIM contains a VID/PID pair directory and matching ACM (**`KBL_SINIT`**)

Parsed and loaded by **`TxtLoadAcmFromPackage`**

ACM validates chipset registers, PCH settings, PCI MMIO space

If SINIT is successful, we are now in TXT mode and MLE can be loaded

Otherwise, TXT launch fails, and the machine boots in *degraded mode*

Measured Launch Environment

MLE validates OS-sensitive data structures and execution environment

Specific settings that Hyper-V may be susceptible to, MSR values, etc…

Transferring execution to the MLE

`BlArchLaunchMle` copies data into the **`TRANSIT`** section of **`TcbLaunch`**

`TcbLaunch` contains the MLE embedded as the "DOS Stub" part of the PE header (the real PE binary has an **`e_lfanew`** PE offset of 12KB+!)

After MLE execution

Calls the first ordinal of **`TcbLaunch`** (**`BlSlEntryPoint`**) which validates the input data and parameters of the **`TcbLaunch`** Boot Application itself

Eventually calls the application entry point (**`TcbMain`**) of the real PE file

TCB Launcher Flow

`TcbLaunch` serves as export module containing the Boot Library API

OS boot logic now lives in **`TcbLoader.dll`** which has two exports **`TcbLoadEntry`** used for boot flow, **`TcbResumeEntry`** for S5 resume flow

Loading Hyper-V is done by **`HvLoader.dll`**, compiled as dynamic library

This factoring allows **`TcbLoader`** & **`HvLoader`** to share an address space

State, memory and APIs can thus be global to both loaders – as if they were drivers running within the same kernel

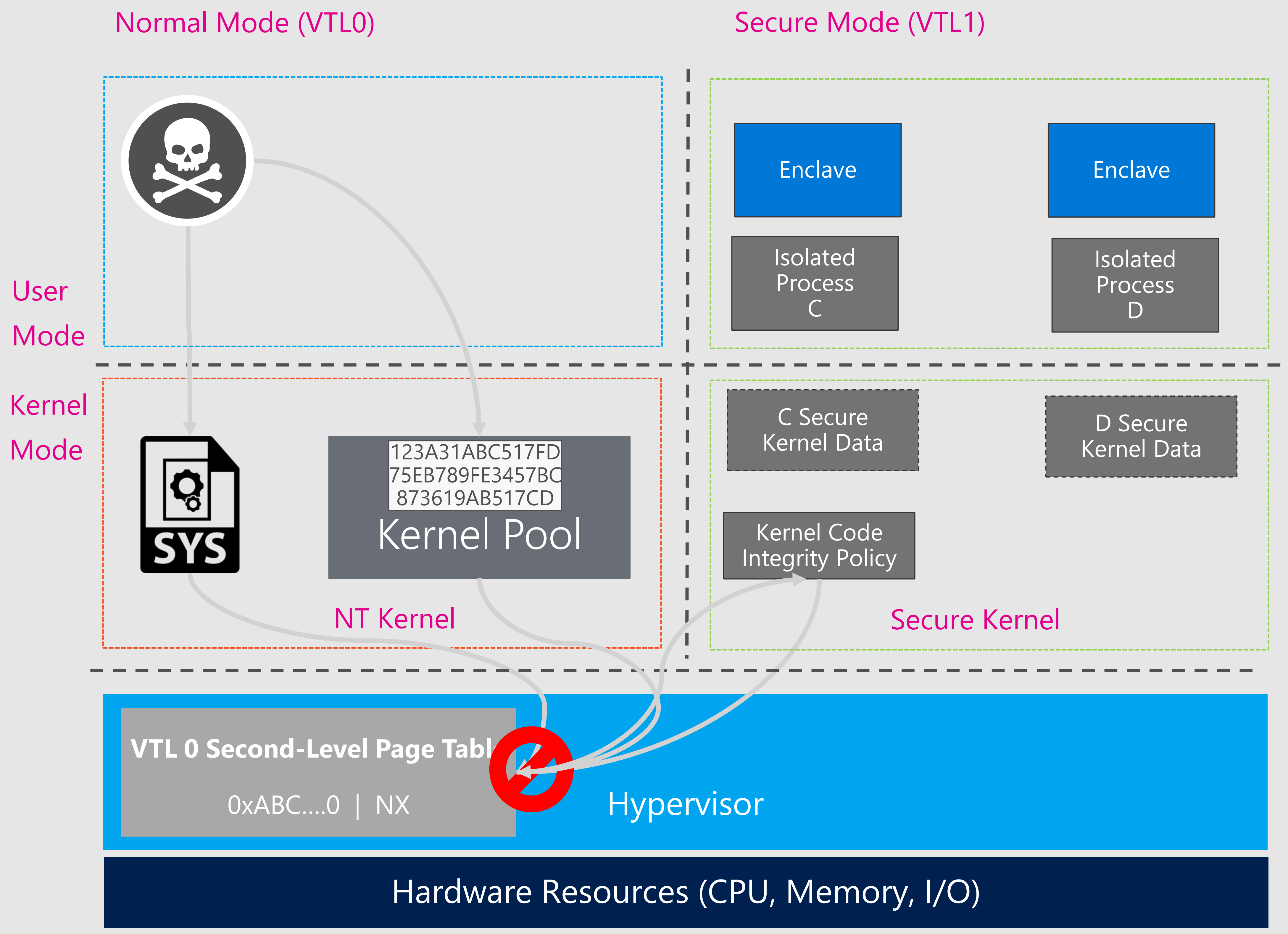
`HvLoader` factored to handle both TXT vs. non-TXT boot

`BlGetExecutionEnvironment` used to determine memory ownership

Multiple validation stages were added to all pointers and registers in ACPI tables and all other ephemeral transfer structures

Virtual Secure Mode

Leverages nested page tables managed by VTL1 to eliminate W^X memory in VTL0 kernel-mode



SLAT Used to Enforce RX-Only

HVCI running in SK validates code pages

If valid, set GPA bits to `R=1 W=0 KMX=UMX=1`

Mode-Based Execute (MBE) Control

Extended-Extended Page Tables (EPT)

XU for user pages

XS for supervisor pages

KMX and **UMX** become separate hardware bits

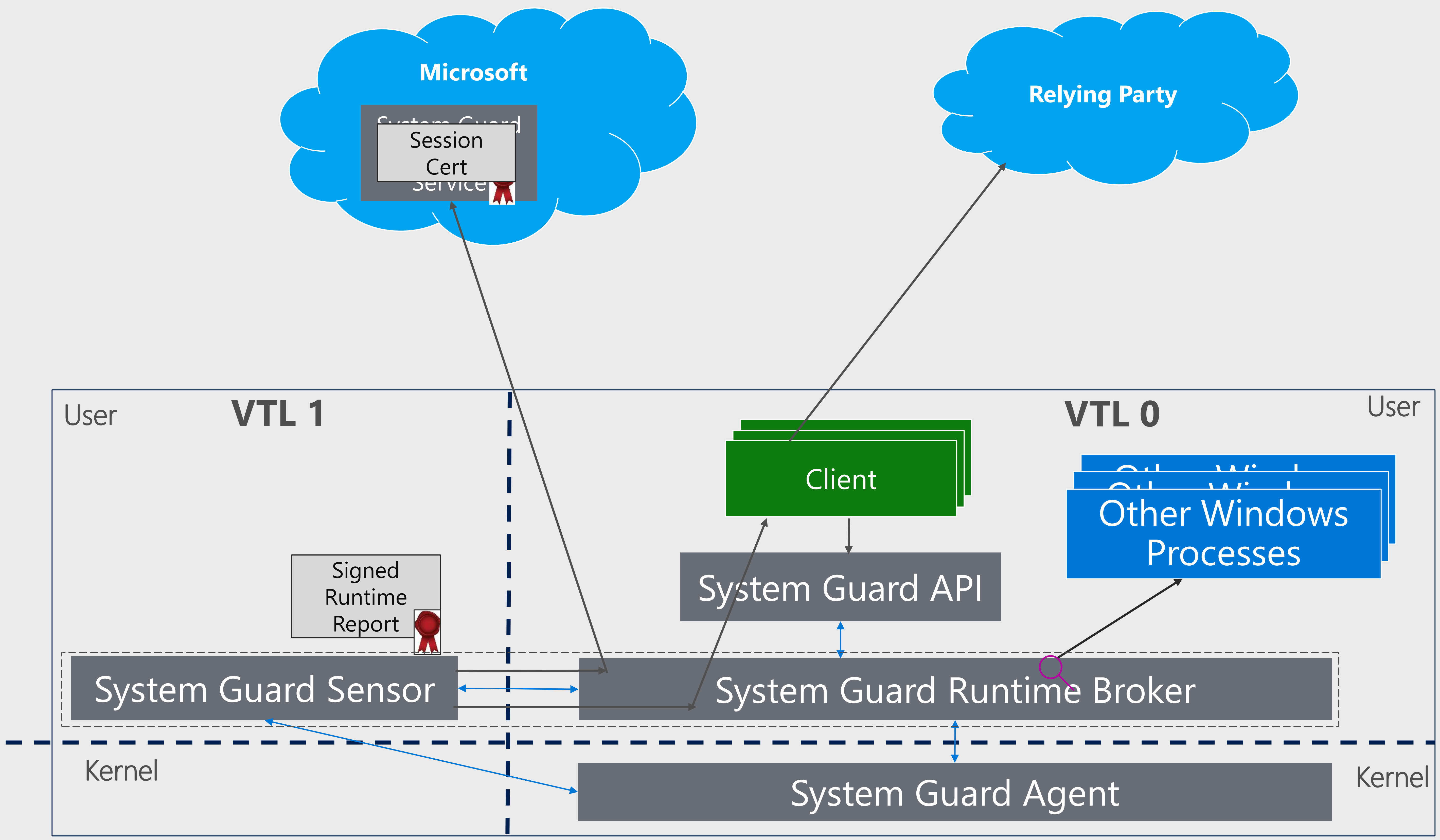
VSM Enclaves

Extensible development model for VSM

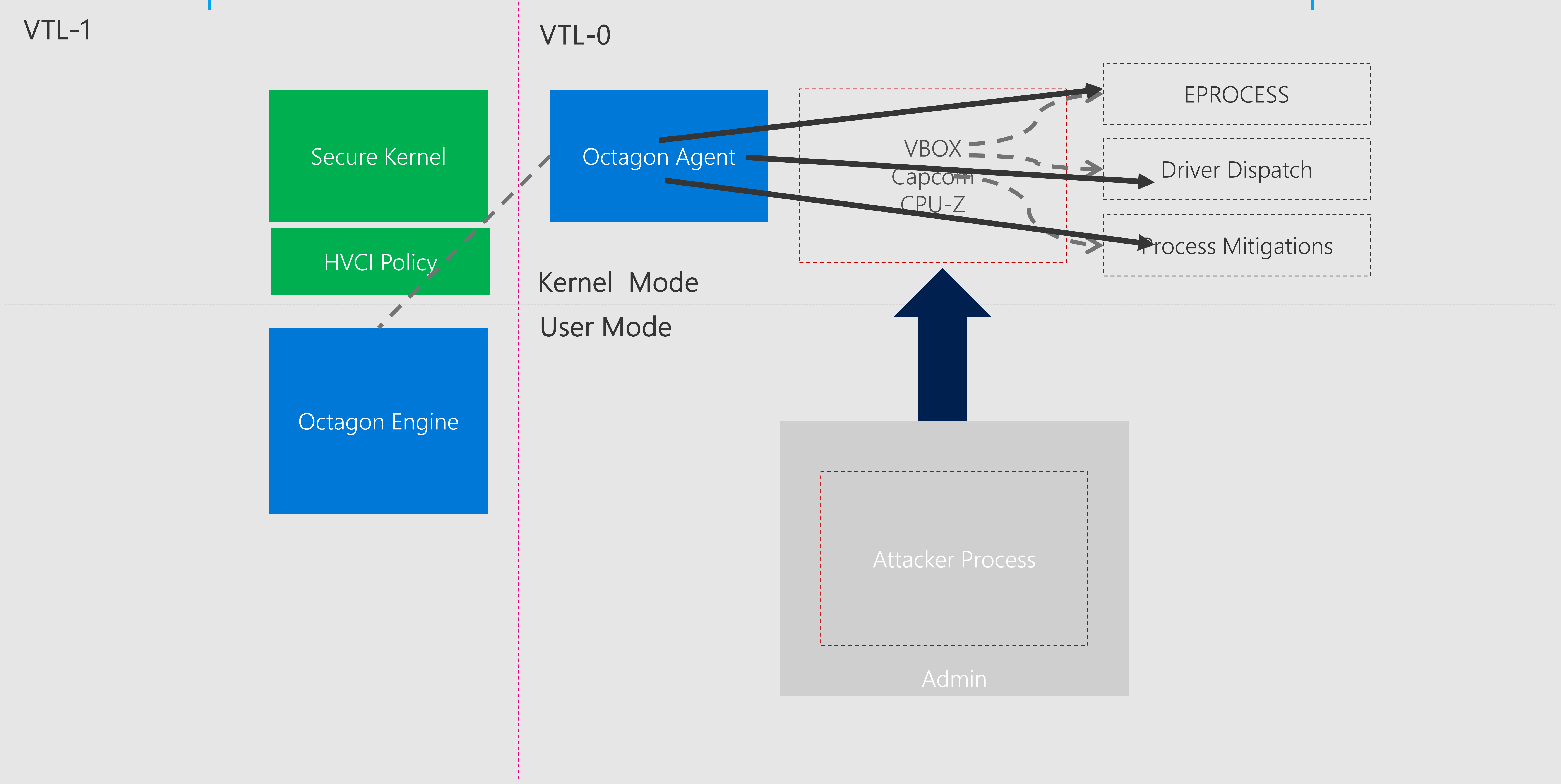
Supports VSM/SGX API documented here

Requires code to be signed by Microsoft

System Guard Runtime Attestation



Example: Failed Attestation via Data Corruption



Octagon Internals

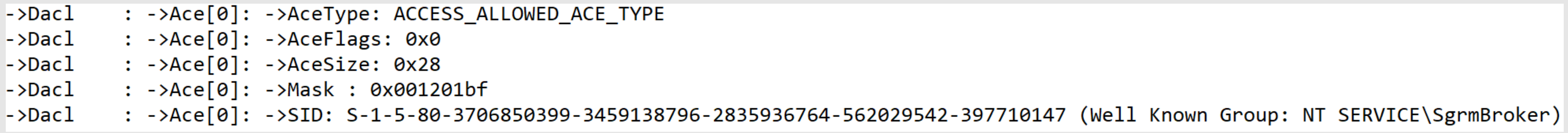
Activation

Agent Driver Component

`SgrmAgent.sys` loads early as a boot time driver – must win potential race against earlier name-squatting driver

Registers Kernel Extension Host to consume internal kernel API access for race-free process enumeration and thread CONTEXT acquisition

Creates `\Device\MSSGRMAGENTSYS` tied to "`SgrmBroker`" Service SID:



```
->Dacl    : ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[0]: ->AceFlags: 0x0
->Dacl    : ->Ace[0]: ->AceSize: 0x28
->Dacl    : ->Ace[0]: ->Mask : 0x001201bf
->Dacl    : ->Ace[0]: ->SID: S-1-5-80-3706850399-3459138796-2835936764-562029542-397710147 (Well Known Group: NT SERVICE\SgrmBroker)
```

But – `IRP_MJ_CREATE` dispatcher only allows handle to be opened by WinTCB Protected Process (Level 0x62) and restricts to a single owner per boot

Activation

Broker Activation

`SgrmBroker.exe` ("System Guard Runtime Monitor Broker") loads as delayed started service, opens handle to Agent, and sends `INIT IOCTL (0x9C402480)`

This returns initial kernel image information (kernel base, size, checksum, timestamp, full path) which will be used to initialize the Assertion Engine

This Engine runs wrapped as a DLL – `SgrmEnclave_secure.dll` for the VSM Enclave which can both execute the assertions and perform system attestation, and SgrmEnclave.dll which is a 'Shim' for assertion-only usage

Finally, registers an RPC Endpoint ("`SgrmRpc`") to provide Client API

Agent Driver

Mailbox Interface

SgrmAgent provides a "Mailbox" `IOCTL` (`0x9c402484`) for requesting:

- User-mode read-only mapping of any piece of kernel-mode VA
- Race-free, referenced access to `EPROCESS`, `ETHREAD`, `DRIVER_OBJECT`
- Suspend/resume of any thread
- User-mode copy of any physical RAM, kernel VA, or MMIO region
- Map/unmap of any file
- User-mode memory region bounds and mapped file name, if any
- User-mode `CONTEXT` of any thread
- Read-only `MSR` access
- Internal structure symbol offset lookup by name

Agent Driver

Looked up Symbols

```
EPROCESS: (User)DirectoryTableBase, Peb, Token, Protection,
SequenceNumber, CreateTime, SignatureLevel, MitigationFlags,
SectionSignatureLevel, (InheritedFrom)UniqueProcessId, Flags2

TOKEN: TokenSource, Privileges, IntegrityLevelIndex,
UserAndGroups, Flags

DRIVER_OBJECT: DriverStart/Size/Name/Unload/StartIo,
MajorFunction[], and DEVICE_OBJECT's DriverObject
```

These lookups can provide hints as to what type of data the runtime attestation engine is looking out for – and possibly ignoring

System Guard Runtime Client API

Provide an Extensible Mechanism for WDATP & 3rd-party AM

Communicates with `SYSTEM`, Local Administrator, and WDATP Service SID

Internal RS4 API has a generic "Report" API and internal registration for WDATP

```
//Application requests an attestation session from System Guard
DeviceAttestationSession session = await
DeviceAttestationManager.CreateSessionAsync(RPID, Nonce);

//Application obtains session report signed by System Guard
Attestation Service
String jwtSessionReport = await session.GetSessionReportAsJwtAsync();

//Application can request the session X.509 certificate to verify the
session report
//and obtain the key to validate the runtime report signature
Certificate sessionCertificate = await
session.GetSessionCertificateAsync();

//Application obtains runtime report signed by System Guard (called
multiple times per session)
String jwtRuntimeReport = await session.GetRuntimeReportAsJwtAsync();
```

New RS5 API provides dedicated `GetSessionReport`, `GetSessionCertificate` and `GetRuntimeReport` functions that can be consumed by 3rd party AM to both validate runtime assertions, as well as that they were generated by a Secure Launch'ed VSM Enclave (or not!)

Octagon Script Model

LUA Engine

Assertions should be rapidly deployable (preferably through Cloud) to react to real-time threats in the wild, while at the same time provide safety and security while potentially dangerous/untrusted Ring 0 data is being parsed

Running Octagon in Ring 3 mitigates the largest privilege escalation worries, but compiled assertions in a language with pointers and a full memory access model could still result in theoretical VTL1 compromise through constructed VTL 0 kernel data

Thus, the Assertion Engine is a full-blown LUA Script Interpreter – providing acceptable performance, good security, and a rapidly iteratable language commonly used in other AM tools (Windows Defender)

LUA<->Broker Assist Wrapper API

How Does LUA Talk to Win32/NT?

When the Assertion Engine is spun up, the `EngHostInitialize` export is called, passing in `g_BrokerAssistCallbackTbl` as an array of "assist" APIs

These are made available through an `AssistWrapper` class which uses the Win32 `CallEnclave` API to *exit* the enclave and execute the required assist

Most of these assists map 1:1 to the IOCTLs which are provided by the Agent, but some don't : `GetSystemInformation/Time`, `NotifyAssertion/LuaFailure`, `NotifyEngine`, `SetTimer`, `TraceLog`

These can simply use the regular Win32/TPM API exposed to user mode

Broker<->LUA Host Wrapper API

How Does the Client API -> Broker Execute LUA?

The Enclave DLL exports `EngHostNotify/DispatchThread/Initialize/Shutdown` for internal use. During initialization, LUA globals can also be set, such as `GLOB_KERNEL_IMAGE_BASE`

`EngHostGetReport` (RS4), `EngHostGetRuntimeReport` (RS5) call into the LUA functions (such as "`Notify`" or "`ExecuteNext`")

`EngHostGetSessionCertificate/Report` use VTL1 Enclave API (`EnclaveGetAttestationReport`) to get VSM-signed proof that the assertions came from trusted system. `EngHostCreate/DestroyAttestationClient` and `EngHostAttest` talk to relying party to submit the attestation data

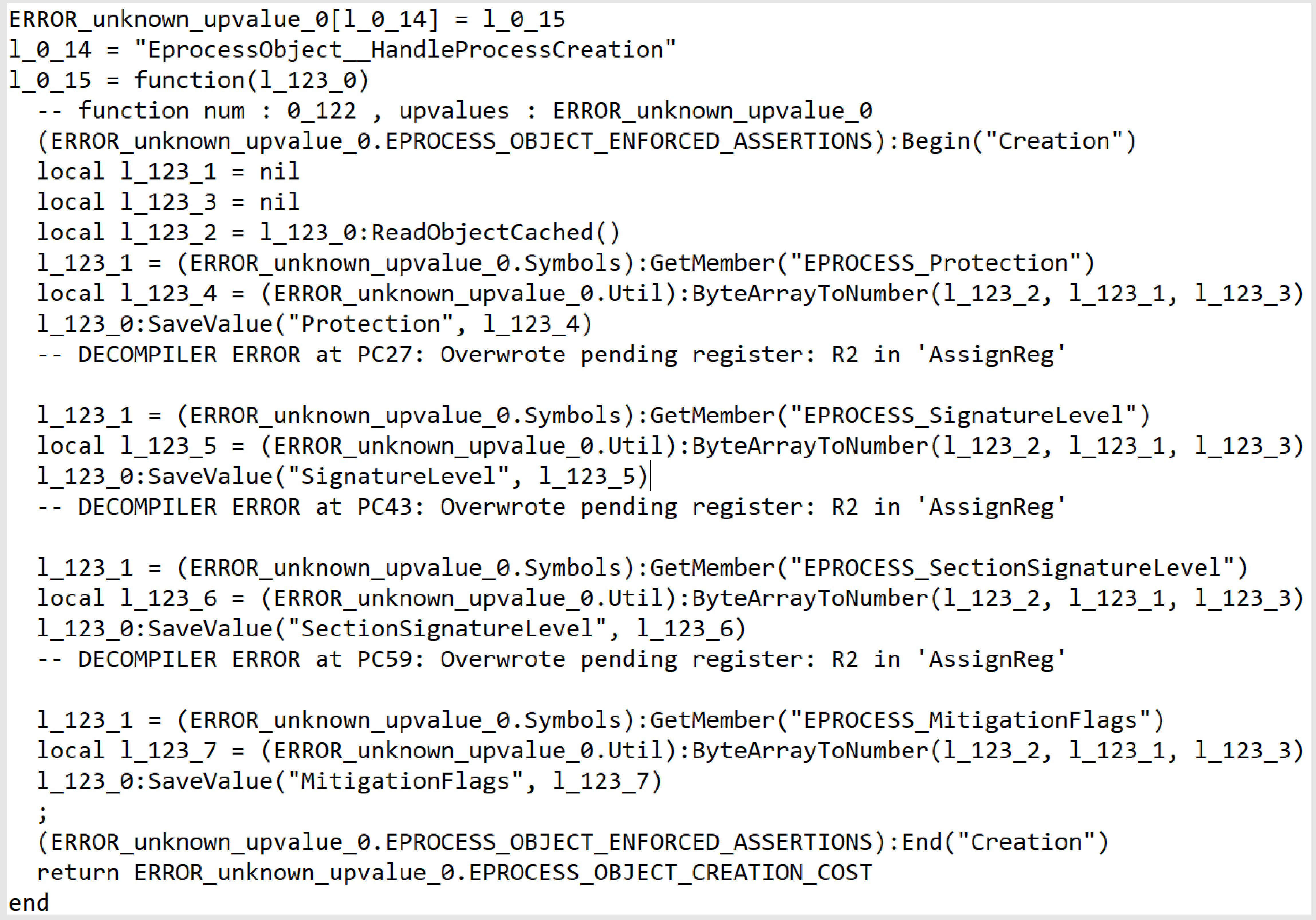
Decompiling the LUA Script

Until Cloud Deployed Scripts Arrived…

The entire LUA script bytecode is in the `.luaseg` PE section of `SgrmEnclave`

No modern, well-supported LUA 5.3.4+ decompiler exists (and MS has used custom engine before)

But with some hacks… some of the logic can be exposed



```
ERROR_unknown_upvalue_0[l_0_14] = l_0_15
l_0_14 = "EprocessObject__HandleProcessCreation"
l_0_15 = function(l_123_0)
  -- function num : 0_122 , upvalues : ERROR_unknown_upvalue_0
  (ERROR_unknown_upvalue_0.EPROCESS_OBJECT_ENFORCED_ASSERTIONS):Begin("Creation")
  local l_123_1 = nil
  local l_123_3 = nil
  local l_123_2 = l_123_0:ReadObjectCached()
  l_123_1 = (ERROR_unknown_upvalue_0.Symbols):GetMember("EPROCESS_Protection")
  local l_123_4 = (ERROR_unknown_upvalue_0.Util):ByteArrayToNumber(l_123_2, l_123_1, l_123_3)
  l_123_0:SaveValue("Protection", l_123_4)
  -- DECOMPILER ERROR at PC27: Overwrote pending register: R2 in 'AssignReg'

  l_123_1 = (ERROR_unknown_upvalue_0.Symbols):GetMember("EPROCESS_SignatureLevel")
  local l_123_5 = (ERROR_unknown_upvalue_0.Util):ByteArrayToNumber(l_123_2, l_123_1, l_123_3)
  l_123_0:SaveValue("SignatureLevel", l_123_5)
  -- DECOMPILER ERROR at PC43: Overwrote pending register: R2 in 'AssignReg'

  l_123_1 = (ERROR_unknown_upvalue_0.Symbols):GetMember("EPROCESS_SectionSignatureLevel")
  local l_123_6 = (ERROR_unknown_upvalue_0.Util):ByteArrayToNumber(l_123_2, l_123_1, l_123_3)
  l_123_0:SaveValue("SectionSignatureLevel", l_123_6)
  -- DECOMPILER ERROR at PC59: Overwrote pending register: R2 in 'AssignReg'

  l_123_1 = (ERROR_unknown_upvalue_0.Symbols):GetMember("EPROCESS_MitigationFlags")
  local l_123_7 = (ERROR_unknown_upvalue_0.Util):ByteArrayToNumber(l_123_2, l_123_1, l_123_3)
  l_123_0:SaveValue("MitigationFlags", l_123_7)
  ;
  (ERROR_unknown_upvalue_0.EPROCESS_OBJECT_ENFORCED_ASSERTIONS):End("Creation")
  return ERROR_unknown_upvalue_0.EPROCESS_OBJECT_CREATION_COST
end
```

Decompiling the Cloud-Deployed LUA Script

In RS5 Builds 17704 and later…

The LUA script is now deployed by cloud, dropped in `\Windows\System32\Sgrm\SgrmAssertions.bin` with a catalog file (`SgrmAssertions.cat`) containing its hash and the Isolated User Mode EKU (1.3.6.1.4.1.311.10.3.37)

File is validated & checked against catalog hash by the enclave, not broker.
It contains a LUA_FILE_HEADER followed by the LUA Bytecode

```
▪ UCHAR  TLVEmpty[6];          // {01, 00, 00, 00, 00, 00}
▪ USHORT Version;              // LUA_FILE_HEADER_FORMAT_VERSION (1)
▪ UINT   SVN;                  // >= 1
▪ USHORT MajorScriptVersion;   // MAJOR_SCRIPT_VERSION (1)
▪ USHORT MinorScriptVersion;   // MINOR_SCRIPT_VERSION (1)
▪ USHORT MajorEngineVersion;   // MAJOR_ENGINE_VERSION (1)
▪ USHORT MinorEngineVersion;   // MINOR_ENGINE_VERSION (1)
```

Current LUA Assertions

Subject to Change & Rapid Iteration

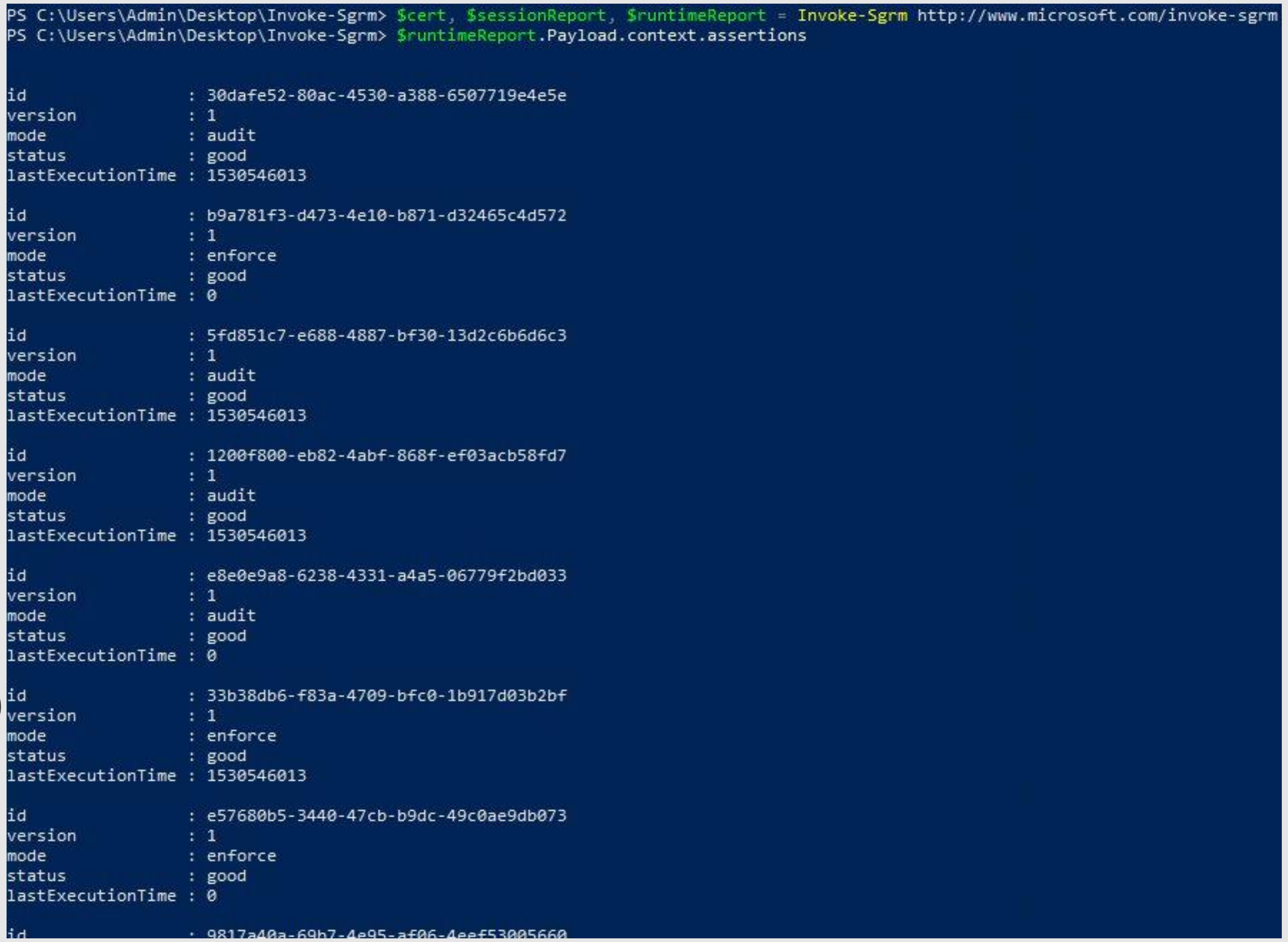
- Process Image Base memory contents & integrity
- Code Integrity for `MsSense.exe`, `MsMpEng.exe` & `SgrmBroker.exe`
- Driver Dispatch Routines for `MsSecFlt`, `SgrmAgent`, `WdFilter`
- Device Objects for `HackSysExtremeVulnerableDriver`, `mimidrv`, `Htsysm`
- `EPROCESS` Data Modification (Mitigations, Token, Protection Level)
- Primary `TOKEN` Data Modification (Groups, IL) and `SYSTEM` Steal/Swap
- Firmware (specifically, SMRAM Unlock on AMD Systems w/ special MSR)

Consider these a useful "tech demo" for the initial implementation as obviously the checks are nowhere exhaustive yet – the magic behind the technology isn't what hardcoded LUA script runs – it's the engine around it

Current LUA Assertion GUIDs

Encoded in Runtime Report

- FIRMWARE_ENFORCEMENT/AUDITMODE_GUID
  5fd851c7-e688-4887-bf30-13d2c6b6d6c3
  b9a781f3-d473-4e10-b871-d32465c4d572
- DRIVEROBJECT_DISPATCHES_GUID
  0efb8b25-8b47-4993-8a44-69e4b732c105
- BLACKLISTED_DEVICES_GUID
  9817a40a-69b7-4e95-af06-4eef53005660
- PRIMARY_TOKEN_ENFORCEMENT/AUDIT_GUID
  33b38db6-f83a-4709-bfc0-1b917d03b2bf
  1200f800-eb82-4abf-868f-ef03acb58fd7
- EPROCESS_OBJECT_ENFORCEMENT/AUDIT_GUID
  e57680b5-3440-47cb-b9dc-49c0ae9db073
  e8e0e9a8-6238-4331-a4a5-06779f2bd033
- CODE_INTEGRITY_GUID
  30dafe52-80ac-4530-a388-6507719e4e5e



```
PS C:\Users\Admin\Desktop\Invoke-Sgrm> $cert, $sessionReport, $runtimeReport = Invoke-Sgrm http://www.microsoft.com/invoke-sgrm
PS C:\Users\Admin\Desktop\Invoke-Sgrm> $runtimeReport.Payload.context.assertions


id                : 30dafe52-80ac-4530-a388-6507719e4e5e
version           : 1
mode              : audit
status            : good
lastExecutionTime : 1530546013

id                : b9a781f3-d473-4e10-b871-d32465c4d572
version           : 1
mode              : enforce
status            : good
lastExecutionTime : 0

id                : 5fd851c7-e688-4887-bf30-13d2c6b6d6c3
version           : 1
mode              : audit
status            : good
lastExecutionTime : 1530546013

id                : 1200f800-eb82-4abf-868f-ef03acb58fd7
version           : 1
mode              : audit
status            : good
lastExecutionTime : 1530546013

id                : e8e0e9a8-6238-4331-a4a5-06779f2bd033
version           : 1
mode              : audit
status            : good
lastExecutionTime : 0

id                : 33b38db6-f83a-4709-bfc0-1b917d03b2bf
version           : 1
mode              : enforce
status            : good
lastExecutionTime : 1530546013

id                : e57680b5-3440-47cb-b9dc-49c0ae9db073
version           : 1
mode              : enforce
status            : good
lastExecutionTime : 0

id                : 9817a40a-69b7-4e95-af06-4eef53005660
```

Cloud Communications

Broker for HTTP Network Communications

Connecting to an HTTPS server is not only impossible from a Protected Process, it is also a bad idea

But Session Reports require connecting to sgrm.microsoft.com to send an HTTP POST request with the attestation (sent to `/v1.0/Attestation`)

Hence, a custom sandboxed process (`SgrmLpac.exe`) is used, which exposes an RPC interface that is used to connect to the service – it runs as a Low Privilege App Container (LPAC), having barely any attack surface

Nit: provides an interesting "living off the land" HTTP POST RPC API…

Octagon Threat Model

Enclave Technologies

Enclave models

Windows supports two enclave types: Intel SGX and Microsoft VSM

Intel SGX

Fully encrypted memory and bus traffic, resilient to SMM (but not ME)
but requires special UEFI enablement, Intel Sky Lake+ SKU, AESM service

Microsoft VSM

Susceptible to runtime SMM attacks and bus snooping/RAM analysis
but runs on any CPU platform with hardware virtualization support

Physical attacker with a quantum laser bit flipper is out of scope

SMM attacks are a realistic attack vector, however. What else is there?

Attack Surface

TPM assumed trusted

Reliance on it for reports, attestation, secret sealing through PCRs

Hypervisor assumed trusted

VTL1 security and secrets vulnerable to Hyper-V bugs and VSM bugs

SMM assumed trusted

Unmeasured SMM runtime components, privileged SMI handlers, access outside SMRAM-backed communication buffers can mess with TXT

Management Engine assumed trusted

TXT guarantees and IOMMU boundaries vulnerable to ME vulnerabilities

Limitations

Vulnerable to race conditions

Any runtime assertion technology will be limited by an attacker restoring normal/expected state within the timing window of the assertion (already an issue with PatchGuard today

Privileged runtime VTL 0 attacker can theoretically supply ‘fake’ data

Measured data for runtime report comes through VTL 0 / Ring 3 broker process and provided by VTL 0 / Ring 0 driver

DRTM and runtime assertions can only measure/verify ‘known’ unknowns

A magic "God Mode" MSR won't be measured if nobody knows it exists. Similarly, data-only corruption techniques and effects must be known

Defense-in-Depth

Fast I/O Dispatch with probe & locked MDLs reduces race conditions

Memory-based assertions to detect data attacks are backed by Memory Descriptor Lists mapped into user mode, simulating a shared data buffer

Ring 3 Broker runs as Windows TCB-level protected process

Minimizes risks of user-mode-only admin attacks, requiring either a protected process bypass or kernel-driven attack

Self-assertion and validation of Octagon's own components

These checks validate that Octagon components haven't been compromised, forcing a successful bypass to have to win a Time-of-Check-To-Time-of-Use (TOCTTOU) race

Future Improvements

Cloud-provisioned scripts with encryption and/or obfuscation

Would create information asymmetry for attackers wishing to fake assertion (must know every possible assertion – a single one leads to detection) – this model has proven successful for PatchGuard

Hypervisor-backed intercepts and guarantees

Gathering data (or enhancing existing data) through hypervisor could reduce reliance on VTL 0 truth provenance

Improve measurement, signing and attestation guarantees of platform

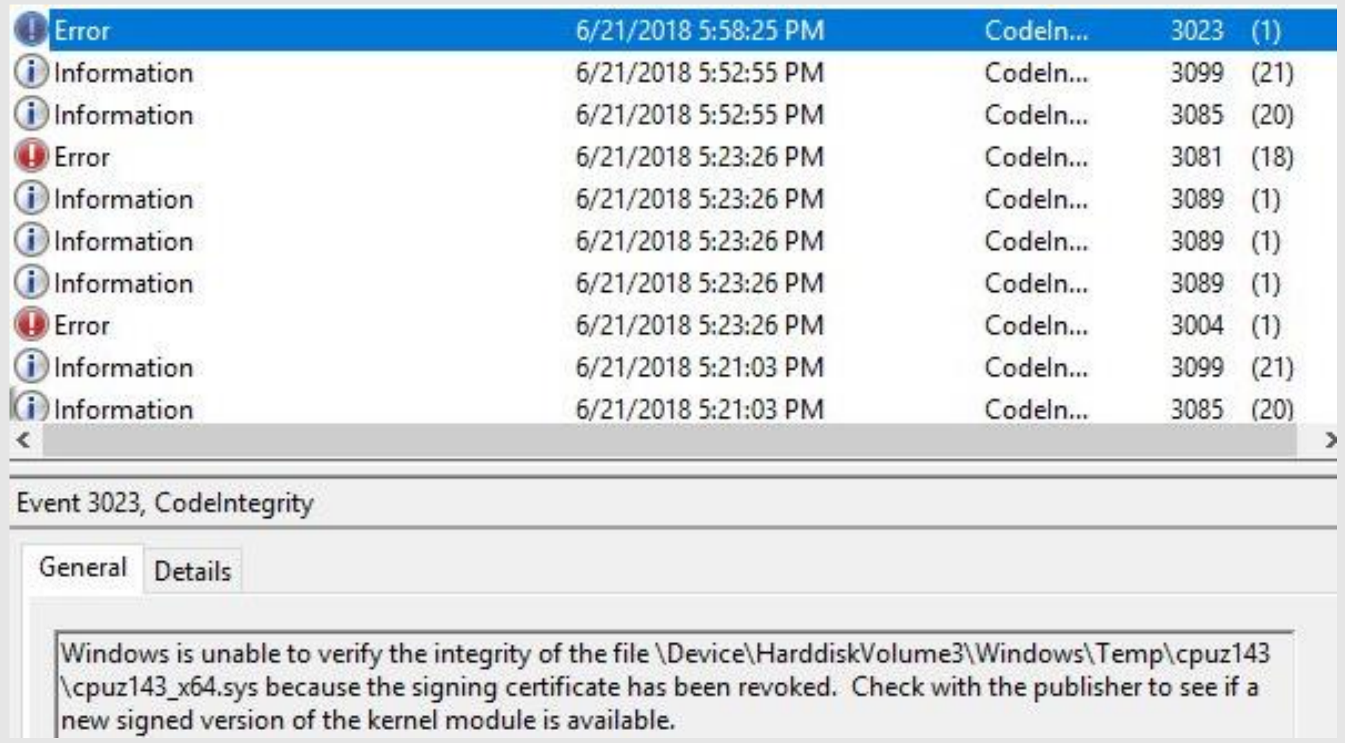
Vendors continuously innovating with stronger guarantees at firmware and chipset level

Improving Kernel Security

VBS enabled by default in all Windows SKUs

Needs Kaby Lake+ due to perf reasons (MBEC)

Activates Kernel CFG, DMA Protection, etc..



```
Error          6/21/2018 5:58:25 PM   CodeIn...   3023  (1)
Information    6/21/2018 5:52:55 PM   CodeIn...   3099  (21)
Information    6/21/2018 5:52:55 PM   CodeIn...   3085  (20)
Error          6/21/2018 5:23:26 PM   CodeIn...   3081  (18)
Information    6/21/2018 5:23:26 PM   CodeIn...   3089  (1)
Information    6/21/2018 5:23:26 PM   CodeIn...   3089  (1)
Information    6/21/2018 5:23:26 PM   CodeIn...   3089  (1)
Error          6/21/2018 5:23:26 PM   CodeIn...   3004  (1)
Information    6/21/2018 5:21:03 PM   CodeIn...   3099  (21)
Information    6/21/2018 5:21:03 PM   CodeIn...   3085  (20)
```

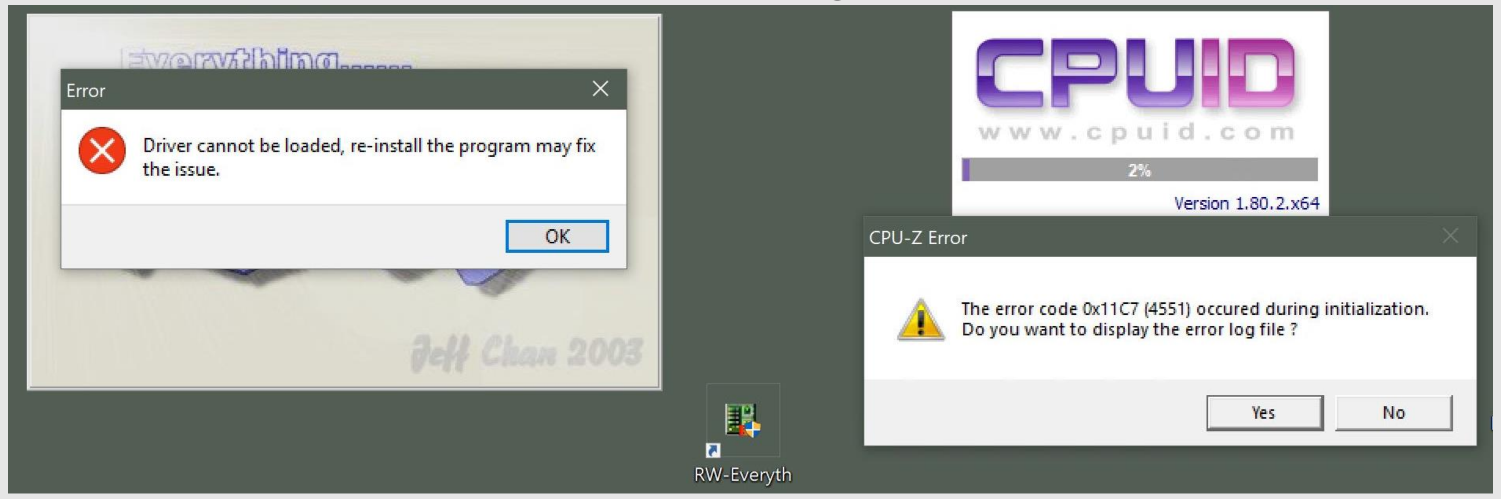
Event 3023, CodeIntegrity

Windows is unable to verify the integrity of the file \Device\HarddiskVolume3\Windows\Temp\cpuz143\cpuz143_x64.sys because the signing certificate has been revoked. Check with the publisher to see if a new signed version of the kernel module is available.

Blocking malicious, vulnerable, abused drivers

With HVCI, drivers that are known sources of abuse blocked via blacklist

Future is to enable more secure signing model (WHQL, EV) by default



Error

Driver cannot be loaded, re-install the program may fix the issue.

CPU-Z Error

The error code 0x11C7 (4551) occured during initialization.
Do you want to display the error log file ?

Read-only kernel memory

HyperGuard enforces memory/MSRs as RO

Looking at expanding to protect current attestation points

Improving Boot Security

System Guard with DRTM

Hardware support for DRTM and measured boot available in WIP

OEM support for TXT is coming in future devices

SMM Isolation and Attestation

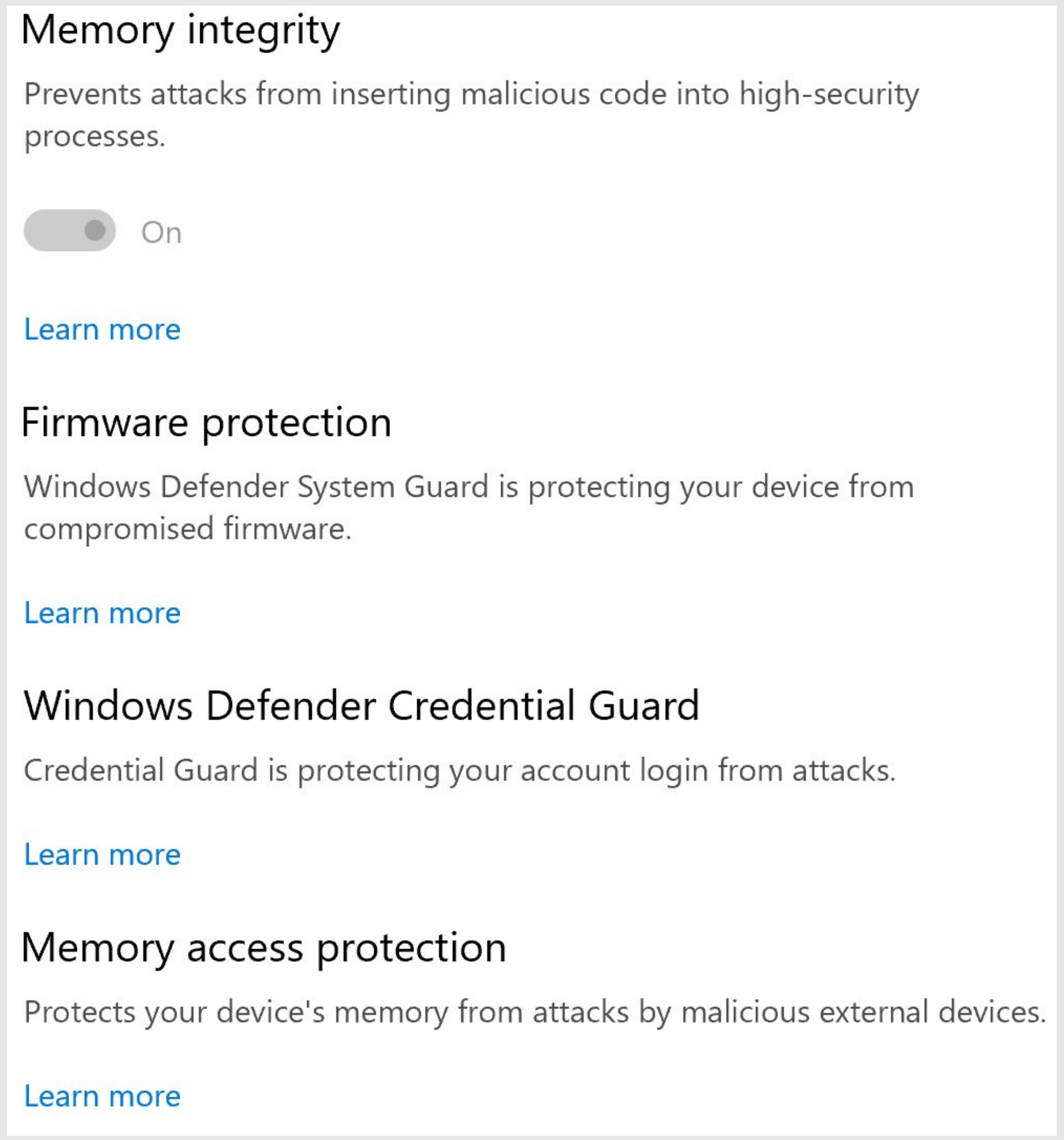
SMM is a key attack vector for TXT and VBS

Working with IHVs to use page table protection against SMM attacks targeting OS & HV memory

Ability to attest memory protection is available

Early boot DMA protection

DMA-r, Early boot IOMMU, and BME Clear are all supported in WIP



Memory integrity

Prevents attacks from inserting malicious code into high-security processes.

On

Learn more

Firmware protection

Windows Defender System Guard is protecting your device from compromised firmware.

Learn more

Windows Defender Credential Guard

Credential Guard is protecting your account login from attacks.

Learn more

Memory access protection

Protects your device's memory from attacks by malicious external devices.

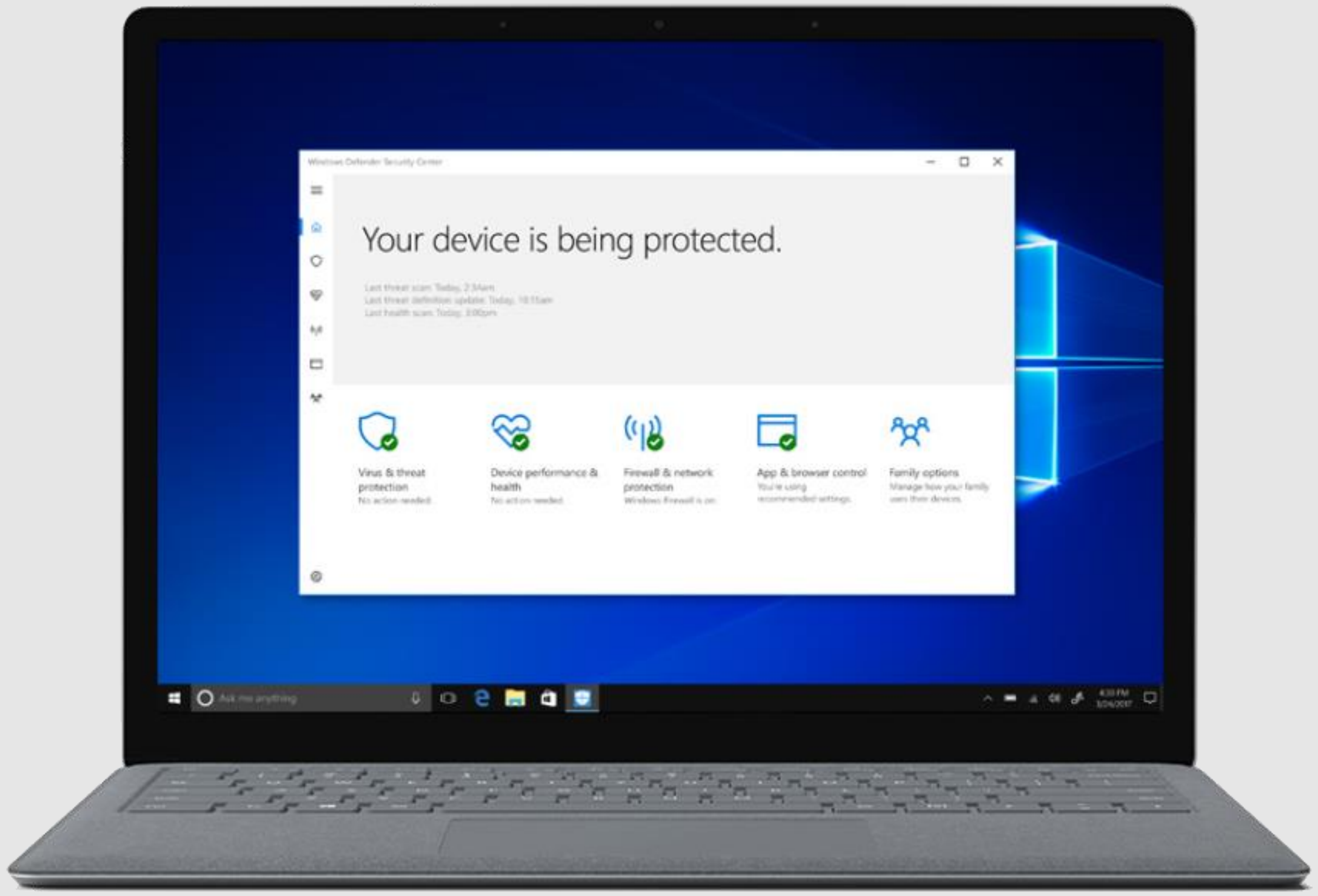
Learn more

Wrap up

Windows security promises are increasing

System Guard Runtime Attestation is critical to providing transparency to users & cloud services

Available now in 1803/1809 with a public attestation API coming soon


Your device is being protected.
Virus & threat protection
Device performance & health
Firewall & network protection
App & browser control
Family options

Aspirational security promises are the guiding principles for security investments