VBS in Windows

Virtualization-Based Security (VBS) Architecture

A narrative explainer of VBS: the problem it solves, why Microsoft introduced it, how it works, and where to learn more.

The issue: attackers reached the kernel

Over the past decade, well resourced attackers moved “down the stack,” regularly achieving or simulating kernel mode access. Common themes:

- Signed driver abuse and BYOVD: vulnerable but legitimately signed drivers are loaded to gain arbitrary kernel read/write.
- Direct LSASS/credential theft: secrets scraped from memory even when user mode defenses exist.
- Kernel tampering: patching code integrity decisions, disabling security features from ring-0.
- DMA attacks: devices reading host memory directly, bypassing CPU page tables.

Traditional assumptions “the kernel is trusted” is no longer held. Protecting secrets inside the same trust boundary as potentially compromised kernel code became untenable.

Why Microsoft introduced VBS (and why now)

VBS arrived when several preconditions aligned:

- Hardware maturity: SLAT became ubiquitous (Intel EPT, AMD NPT), enabling a hypervisor to own a second translation layer and strictly isolate memory from the guest OS kernel.
- I/O memory protection: VT-d/IOMMU widely available to constrain DMA so devices cannot touch protected pages.
- Secure boot chain adoption: UEFI + Secure Boot provided a verifiable foundation for launching a minimal hypervisor and secure components early in boot.
- Ecosystem readiness: driver signing, Windows Defender platform, and enterprise management (GPO, MDM) made broad deployment feasible.
- Threat reality: frequent BYOVD incidents and post-exploitation frameworks targeting the kernel demanded isolation stronger than ring-based privilege alone.

The result: a small, hypervisor-enforced “secure world” alongside normal Windows, so that even if the normal kernel is compromised, high-value secrets and policy remain protected.

How VBS works (at a glance)

- Virtual Trust Levels (VTLs)
  - VTL0: Normal world for Windows kernel and user processes.
  - VTL1: Secure world hosting the Secure Kernel and trustlets.
- Memory isolation
  - SLAT provides two-stage translation: guest virtual → guest physical → host physical.
  - The hypervisor owns the second stage (EPT/NPT), mapping VTL1 so VTL0 cannot read or write it.
  - IOMMU prevents devices from DMA-accessing protected regions.
- Secure Kernel and trustlets
  - A minimal Secure Kernel offers primitives to isolated “trustlet” processes (for example, LSA isolation).
  - VTL0 receives decisions or opaque handles, not raw secrets.
- Hyper-V hypervisor
  - Type-1 hypervisor that enforces VTL boundaries, controls second-level page tables, and mediates interrupts and mappings.

What VBS enables in Windows

- Credential Guard: stores sensitive credential material in VTL1 to blunt LSASS scraping and similar attacks.
- HVCI (Hypervisor-Enforced Code Integrity): protects code-integrity state and enforces signed, untampered kernel code.
- Application Guard: isolates untrusted browser sessions using the same virtualization primitives.
- Kernel Data Protection (KDP): marks select kernel memory as hypervisor-enforced read-only at runtime.
- HyperGuard and related hardening: invariants checked with hypervisor assistance.

Benefits, trade-offs, and limits

- Benefits
  - Hardware-backed isolation against kernel memory scraping and many BYOVD-style tampering attempts.
  - DMA resilience for protected regions via IOMMU.
- Trade-offs
  - Some overhead, typically modest on modern CPUs with SLAT and IOMMU.

- Incompatibility with certain legacy or non-conformant drivers, especially under HVCI.

- Limits
  - Does not prevent all user-ñ mode theft or vulnerability classes.
  - Trustlets must themselves be secure, and misconfiguration can weaken guarantees.

Quick start: checking and enabling

Architecture diagrams

1) High-level VBS trust separation

```
+--------------------------+        +---------------------------+
|        VTL0              |        |           VTL1            |
|  Normal Windows world    |        |    Secure world (SK)      |
|                          |  calls |  Secure Kernel +          |
|  User processes          +------->+  Trustlets (e.g., LSAiso) |
|  Windows kernel (ntoskrnl)|       |                           |
|                          |        |  Secrets, CI state        |
|  Drivers                 |        |  are kept here            |
+--------------------------+        +---------------------------+
                ^                              ^
                |                              |
                |        Hyper-V enforces VTL boundaries
                +---------------[ Hyper-V ]---------------------+

```

2) Memory isolation with SLAT and IOMMU

```
Guest virtual (GV)  -->  Guest physical (GP)  -->  Host physical (HP)
       |                         |                        |
       |  (guest page tables)    |  (EPT/NPT controlled |  (real RAM)
       |                         |   by Hyper-V)        |

VTL1 pages mapped so VTL0 has:  no read / no write / no execute.
DMA attempts -> IOMMU/VT-d remapping blocks access to VTL1 regions.

```

3) Data flows at a glance

```
[App in VTL0]
    |
    v   (syscall/IOCTL)
[Windows kernel in VTL0]  ---->  [Secure service API boundary]
                                        |
                                        v
                           [Trustlet in VTL1]
                                |  (operate on secrets/policy)
                                v
                    [Decision/Handle back to VTL0]

Note: VTL0 receives only results/handles, not raw secrets.

```

```
13
```

4) Feature mapping to VBS primitives

```
Credential Guard  -> Trustlet in VTL1 holds credential material
HVCI              -> CI policy/state protected by VTL1 + Hyper-V
KDP               -> Hypervisor-enforced read-only kernel regions
WDAG              -> Isolation using virtualization primitives

```

- System Information (msinfo32): confirm “Virtualization-based security” and “Device Guard” status.
- Windows Security → Device security → Core isolation details.
- Enterprise: enable Credential Guard and HVCI via Group Policy or MDM profiles.

References

- Virtualization-based security (VBS) overview — Microsoft Docs: https://learn.microsoft.com/windows/security/identity-protection/credential-guard/virtualization-based-security
- Credential Guard — Microsoft Docs: Credential Guard overview
- HVCI (Memory integrity) — Microsoft Docs: https://learn.microsoft.com/windows/security/application-security/application-control/user-mode-code-integrity/memory-integrity
- Kernel Data Protection — Microsoft Docs: https://learn.microsoft.com/windows-hardware/drivers/bringup/kernel-data-protection
- IOMMU/VT-d for DMA protection — Microsoft Docs: https://learn.microsoft.com/windows/security/information-protection/kernel-dma-protection/overview