ANALYSIS OF THE ATTACK SURFACE OF WINDOWS 10 VIRTUALIZATION-BASED SECURITY

Rafal Wojtczuk
rafal@bromium.com
Agenda

• Short reminder on VBS architecture
• Credential Guard properties and internals
• HV Code Integrity properties and internals
• Hyper-V security/complexity/attack surface
• More details in the whitepaper
Scope

• Most of this research done with W10 1511
• Intel’s hardware (when hw mentioned)
• Mixed original, little-known and well-known content
VBS architecture

- VTL0
  - secret.txt
  - ntosknl.exe

- VTL1
  - Lsalso.exe
  - securekernel.exe

Usermode
Kernelmode

EPT
Hyper-V
Credential Guard architecture

Picture taken from BH2015 Microsoft presentation
Mimikatz fails on CG-protected box
CG scenario 1

• Admins just enabled CG in Group Policy
• No further hardening
• Easy to deploy
Lsalso trustlet, running in VTL1, exposes the above functions via RPC over ALPC port `\RPC Control\LSA_ISO_RPC_SERVER`
NtlmIumProtectCredential

- Input (from lsass.exe): plaintext credentials
- Output (from Lsalso.exe): blob with encrypted credentials
NtlmLumLm20GetNtlm3ChallengeResponse

- Input (from lsass.exe): blob with encrypted credentials + NTLM challenge
- Output (from Lsalso.exe): NTLM response
Scenario 1 properties

- After logon, no cleartext credentials in lsass
- While user is logged in, lsass will auth to remote servers automatically (SSO), for attacker as well
- If attacker collects encrypted blob, he can force Lsalso to auth even after logout (until reboot)
- Demo
Credentials during logon?

- There is still a problem with how the unencrypted credentials are initially delivered to VTL1 (which happens during logon). “rundll32.exe user32.dll,LockWorkStation”.
- If not using smart-card based authentication, then the plaintext credentials can be captured by keylogger and used anywhere, anytime.
- In case of smart-card based authentication, the NTOWF hashes sent by KDC can be captured and reused.
CG scenario 2

• Credential Guard with armor key protection and smartcard-based authentication
• Nontrivial deployment challenge
• Possible to enable without TPM, but in such case no real advantage
Scenario 2 properties

- No more cleartext creds in lsass, ever
- Still, as before, until reboot, attacker can interact with CG and have it perform all SSO-supported authentications for remote resources
- There is no reliable way to deliver “user has logged out, refuse future SSO” message to VTL1
CG properties summary

• Even in the most hardened configuration, once attacker has SYSTEM privileges, they can silently authenticate as logged-in user to remote servers, from the compromised machine, until reboot
• No more classical pass-the-hash – but attackers can adapt and start lateral movement from the same machine, until reboot
• In classical pass-the-hash, one can reuse stolen hashes anytime, from anywhere – thus CG is an improvement
• Again, no hypervisor compromise required for the above attack, just root partition compromise
VBS-enforced code integrity

- Windows 10 can enforce code integrity of usermode binaries, usermode scripts and kernelmode code; the latter via VBS
- We focus on kernelmode case
- The goal – not allow execution of any unsigned code in kernel context, even if the kernel has been compromised
VBS-enforced code integrity

• Basic idea: trusted code (running in VTL1) agrees to grant execute rights in EPT tables of the root partition only for pages storing signed code

• No such page can be both writable and executable
Mixing signed & unsigned code

• Common configuration: unsigned usermode code allowed, unsigned kernelmode denied
• Usermode wants to execute unsigned code at C
  – VTL1 must grant execute right for C in EPT
• Usermode switches to kernelmode, and jumps to C
Kernel HVCI is based on secvisor

• Separate EPT for code originating from signed and unsigned page

• Root partition is configured so that any attempt by unsigned usermode code to enter kernelmode results in vmexit (and EPT flip)
  – IDT, GDT limits set to 0, syscall&sysenter disabled
Kernel HVCI and kernel exploits

- Attackers love arbitrary code running in ring0
- SMEP a problem, but natural bypass:
  - Get ROP capability, then clear CR4.SMEP
  - Or, via write-what-where, clear U/S bit in PT
  - Run your arbitrary code
- Not working with Kernel HVCI!
- Also, cannot hook kernel code, at least not directly
- Data-only exploits, or ROP-only, still fine
Kernel HVCI bypass, MS16-066

• Before MS16-066 fix, there are some pages with RWX permission in root partition (kernelmode) EPT

• Likely artifacts of early boot phase

• Attacker can find them by probing each physical page for write and execute, in ring0
Kernel HVCI bypass, MS16-066

C:\Users\testuser\probex>probex 0xf000000 2000000
ntoskrnl.exe at FFFFF802EDA82000
tryexcept at FFFFF802EDE5E292
kthread at FFFFFE000E6C080
stack_base=FFFFD000769C5000 limit=FFFFD000769CB000
starting phys mem probe:
0xf000000-0x1000000 (size 0x1000000): not rwx
0x1000000-0x1015700 (size 0x157000): rwx
0x1015700-0x1100000 (size 0xea9000): not rwx

C:\Users\testuser\probex>
HYPERV-V SECURITY
[Un]usual threat model

• Usual model: hypervisor must be resistant to attacks coming from unprivileged, worker VMs

• Without VBS, root partition is semi-trusted; it can compromise Hyper-V (no big deal) because
  – HvCallDisableHypervisor hypercall
  – Cleartext hiberfile
  – VTd not enabled

• With VBS, the threat comes from the root partition
Necessary support

• Secureboot
  – many vulnerabilities in the past allowing secureboot bypass
• VTd
  – without it, possible to overwrite hypervisor via DMA
• TPM
  – needed to secure S4, see below
Root partition privileges

• Access to privileged hypercalls

• Possible to overlook some dangerous functionality, or e.g. memory corruption bug
Root partition privileges

• Access to almost all physical memory range
  – Without pages allocated for Hyper-V and VTL1
  – Including
    • chipset and PCIe MMIO
    • ACPI NVS
  – LAPIC and VTd bars not accessible
Root partition privileges

• I/O ports: all available except:
• 32, 33 (PCH interrupt controller), 160, 161 (same)
• 0x64, lpc microcontroller (A20 gate)
• 0xcf8, 0xcfc-0xcff – PCI config space
• 0x1804. It is PMBASE+4 == PM1_CNT, it holds the SLP_EN bit, that triggers S3 sleep; see below
Root partition privileges

- MSR – none available directly except:
  - three SYSENTER MSRS
  - fs/gs/shadow gs base
- So, Hyper-V has at least a chance to react properly
Problem 1 – unfiltered MMCFG

- MMCFG is a region of physical address space; access to it results in PCIe config space access
  - Device-specific registers, memory bars locations
- REMAP_LIMIT/REMAP_BASE are locked
- Overlapping RAM with PCIe memory bar does not work
- Anything else interesting we can overlap/cover?
Overlap VTd bars

```
[root@haswell bh16]# cp /sys/firmware/acpi/tables/DMAR .
[root@haswell bh16]# iasl -d DMAR >/dev/null
Loading Acpi table from file DMAR
Acpi Data Table [DMAR] decoded
Formatted output: DMAR.dsl - 5488 bytes
[root@haswell bh16]# grep -i register DMAR.dsl
[038h 0056  8]  Register Base Address : 00000000FED90000
[050h 0080  8]  Register Base Address : 00000000FED91000
[root@haswell bh16]# ./rdmem 0xfed90000
phys memory at 0xfed90000: 0x00000010 0x00000000 0x20660462 0x00c00000
[root@haswell bh16]# ./rdmem 0xfed91000
phys memory at 0xfed91000: 0x00000010 0x00000000 0x20660462 0x00d20080
[root@haswell bh16]# setpci -s 0:2.0 0x10.l
f5800004
[root@haswell bh16]# setpci -s 0:2.0 0x10.l=0xfed90004; ./rdmem 0xfed90000; setpci -s 0:2.0 0x10.l=0xf5800004
phys memory at 0xfed90000: 0x00000000 0x00000000 0x00000000 0x00000000
[root@haswell bh16]# setpci -s 0:2.0 0x10.l=0xfed91004; ./rdmem 0xfed91000; setpci -s 0:2.0 0x10.l=0xf5800004
phys memory at 0xfed91000: 0x00000000 0x00000000 0x00000000 0x00000000
[root@haswell bh16]#
```

But write access hangs the tested platform 😊
Problem 2 – chipset registers

- Some memory-mapped regions, e.g. in MCHBAR, have thousands of registers, most of them undocumented at all
- Are all of them locked? Anything evil can be done?
- I do not know
S3 sleep

• S3 is fragile from security POV
• Boot script hijack vulnerability from 2014 could be used to take control over the hypervisor
  – likely all firmware makes were affected
• More potential attacks via S3 thinkable (see the whitepaper)
S4 sleep

• S4 is even more fragile from security POV
• Need to protect integrity of hiberfile
• With VBS, it is encrypted
• Need to keep the key secret
• If TPM available, the key is sealed to TPM
• If no TPM, then the key is cleartext in UEFI variable
S4 is insecure without TPM
SMM

- SMM is highly-privileged mode of CPU, unrestricted by hypervisor.
- Usually, firmware vendors pack quite some services in SMM; they can be invoked by write to I/O port 0xb2.
- A lot of bugs in SMM found recently.
SMM code tends to be buggy
SMM

• It is well-known that SMM vulnerability can be used to compromise a hypervisor in runtime
  – BTW, secureboot as well
• VBS allows direct access to I/O port 0xb2, as well as to ACPI NVS
• Intel researchers demoed searching VTL1 memory for password hashes
SMM abuse example

testuser@DESKTOP-W10 /c/Users/testuser/projects/bh16
$ tasklist | grep -i iso
LsaIso.exe 812 Services 0 3,108 K

testuser@DESKTOP-W10 /c/Users/testuser/projects/bh16
$ ./smm31.exe scan 0 2>/dev/null >scan.txt

testuser@DESKTOP-W10 /c/Users/testuser/projects/bh16
$ tail -4 scan.txt
VMCS at 0x8d4d000: host: rip=FFFFFF8000809011E cr3=000000000007C9000 ept=0000000006A7201E
guest: rip=FFFFFFFFFD05000 cr3=000000000001AB000 cr4=00000000001526F8
VMCS at 0x8d4f000: host: rip=FFFFFF8000809011E cr3=000000000007C9000 ept=0000000006A7901E
guest: rip=FFFFFF7800003035 cr3=0000000000121B000 cr4=0000000000026F8

testuser@DESKTOP-W10 /c/Users/testuser/projects/bh16
$ wc -l scan.txt
32 scan.txt
SMM abuse example

testuser@DESKTOP-W10 /c/Users/testuser/projects/bh16

$ cat hyperhook_cpuid.c
#include <stdint.h>
#include "regsh.h"
#include "vmx.h"

void hook_c(struct regs * regs)
{
    if (do_vmread(VM_EXIT_REASON) == EXIT_REASON_CPUID &&
        regs->r13 == 0xAABBCCDDAABBCCDDULL) {
        regs->r13 = 0x1122334411223344ULL;
    }
}
VMCS at 0x01fb000: host: rip=FFFF8000809011E cr3=0000000007C9000 ept=0000000006A7201E

wmk: level 0 ptbase 0000000007C9000 pte 0000000007C7D023
wmk: level 1 ptbase 0000000007C7D000 pte 0000000007CF023
wmk: level 2 ptbase 0000000007CF000 pte 0000000040001A1
va FFFFF8000809011E -> pa 00000000409011E
4C8B4828244C8948

before cpuid: r13=AABBCDDAABBCDD; after cpuid: r13=AABBCDDAABBCDD

now=4C8B4828244C8948 write 4C8B48FFFF6FEDDE9 pbase=0000000040000000
shsize=0x2000
shellcode written to phys 0000000040000000

before cpuid: r13=AABBCDDAABBCDD; after cpuid: r13=1122334411223344
Summary

• Despite its limited scope, VBS is useful
• A lot of effort by MS to make it as secure as possible; still, unusual attack surface
• VTd, TPM strictly necessary (with secureboot)
• SMM vulnerabilities the greatest threat
Questions?
Extra slides: Non-VBS-specific threats

• CPU erratas
• Rowhammer
• Flashable discrete hardware
VTL1 attack surface

- RPC services implemented in LsaIso (including RPC demarshalling code)
- 48 services implemented in securekernel\!IumInvokeSecureService (called by nt! HvlpEnterIumSecureMode)
- VTL1 extensively calls into VTL0 to use some services – need to sanitize all responses
Other funny chipset capabilities

• E.g. chipset can program DRAM SPD
• Capability locked by sane BIOS

Picture taken from Wikipedia article on Serial Presence Detect