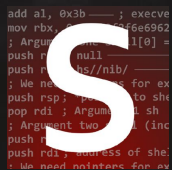


Next-Gen Exploitation:
Exploring the PS5 Security
Landscape



whoami

- @SpecterDev
- Security researcher with a focus on kernel and platform security
- Work on console security as a hobby
- Started with PS4 ~5 years ago
- Also co-host Dayzerosec podcast/media channel
- First time presenter

Agenda

- Where we were (PS4 exploitation)
 - Attack surface, mitigations, post-exploitation
- Where we are now
 - Reduced attack surface
 - Enhanced mitigations
 - Hypervisor-based security
- Hypervisor analysis
- Overview on security co-processor(s)
- Future research and ideas

Notes

- There won't be 0days / non-public bugs
- Mainly a reversing-focused talk with a few exploits & techniques
- We'll focus on kernel, hypervisor, and post-exploitation
 - Mostly x86
 - Virtualization
 - Memory Management
 - Paging

Notes

- For userland stuff, check out theflow's talk from 2022 hardware.io USA
 - [“bd-jb: Blu-ray Disc Java Sandbox Escape”](#)

Where we were

The PlayStation 4



Where we were - PS4 Attack Surface

- Runs a modified FreeBSD 9.0
- Core & networking are mostly untouched
- Some drivers were modified/added
- ~ 150 custom syscalls added
 - No legacy syscalls

Syscall ID	PS4 FW Version	Syscall name	Notes
99	<=1.01?	sys_netcontrol	-
101	<=1.01?	sys_netabort	-
102	<=1.01?	sys_netgetsockinfo	-
113	<=1.01?	sys_socketex	Like existing socket syscall, but with the addition of a name argument.
114	<=1.01?	sys_socketclose	-
125	<=1.01?	sys_netgetliflist	-
141	<=1.01?	sys_kqueueex	-
379	>1.01 <=1.76?	sys_mtypeprotect	-
532	<=1.76?	sys_regmgr_call	-
533	<=1.01?	sys_jitshm_create	Only callable from a jit compiler process, else returns EPERM (0x1)
534	<=1.01?	sys_jitshm_alias	Only callable from a jit compiler/application process, else returns EPERM (0x1)
535	<=1.01?	sys_dl_get_list	Only callable from a debugger, core dump, or syscore process, else returns EPERM (0x1)
536	<=1.01?	sys_dl_get_info	Only callable from a debugger, core dump, or syscore process, else returns EPERM (0x1)

Where we were - PS4 Attack Surface

- Applications are isolated to sandboxed filesystem
- No syscall filtering
 - Ad-hoc permission checking inside syscalls
- Custom syscalls were a source of bugs
 - Most notably the 4.05FW “namedobj” bug disclosed by fail0verflow
 - Type confusion yielding arbitrary free
 - With infoleak could be used to jailbreak

Where we were - PS4 Attack Surface

- Browser/games had access to privileged drivers
 - eBPF
 - [4.50 FW Bug \(qwertyoruiopz\)](#)
 - Set filter & write race condition yielding use-after-free (UAF)
 - [5.05 FW Bug \(qwertyoruiopz\)](#)
 - Set filter race condition yielding double free
 - Raw sockets
 - [6.72 FW Bug \(theflow\)](#)
 - IP6_EXTHDR_CHECK double free

Where we were - PS4 Attack Surface

- Custom drivers were also a source of bugs
 - [9.00 FW Bug “pOOBs4” \(reported by theflow\)](#)
 - Integer truncation yielding heap Out-of-Bounds (OOB) write
 - Exploitable via maliciously formatted USB drive
- Steadily been improving on both PS4 and by extension PS5
 - 6.xx firmwares made huge leaps
 - Various drivers like USB were removed and syscall permission checking was tightened up

Where we were - PS4 Mitigations

- Had weak mitigations even for the time it launched (2013)
- No Address Space Layout Randomization (ASLR) at launch
 - Enabled in 2.xx
- No Supervisor Mode Access/Execution Prevention (SMAP/SMEP) for its lifespan
 - Userland memory access from ring0 is allowed
- JIT memory was accessible to browser due to logic bug until 2.xx FW
 - Didn't even have to ROP < 2.xx

Where we were - PS4 case study “pOOBs4”

244

#1340942

size_t-to-int vulnerability in exFAT leads to memory corruption via malformed USB flash drives

Share:



SUMMARY BY PLAYSTATION



Summary

A heap-based buffer overflow can be triggered by a malformed exFAT USB flash drive.

Vulnerability

The vulnerability is in Sony's exFAT implementation where there is an integer truncation from 64bit to 32bit on a size variable that is used to allocate the up-case table:

Code 381 Bytes

```
1 int UVFAT_readupcasetable(void *unused, void *fileSystem) {
2     ...
3     size_t dataLength = *(size_t*)(upcaseEntry + 24);
4     size_t size = sectorSize + dataLength - 1;
5     size = size - size % sectorSize;
6     uint8_t *data = sceFatfsCreateHeapV1(0, size);
7     ...
8     while (1) {
9         ...
10        UVFAT_ReadDevice(fileSystem, offset, sectorSize, data);
11        ...
12        data += sectorSize;
13        ...
14    }
15 }
```

Namely, `dataLength` and `size` are both 64bit wide, however the `size` argument of `sceFatfsCreateHeapV1()` is 32bit wide:

Where we were - PS4 case study “pOOBs4”

- Gave a one-shot out-of-bounds write into kernel heap
 - Contents were controlled
 - But not flexible
 - From USB filesystem table data

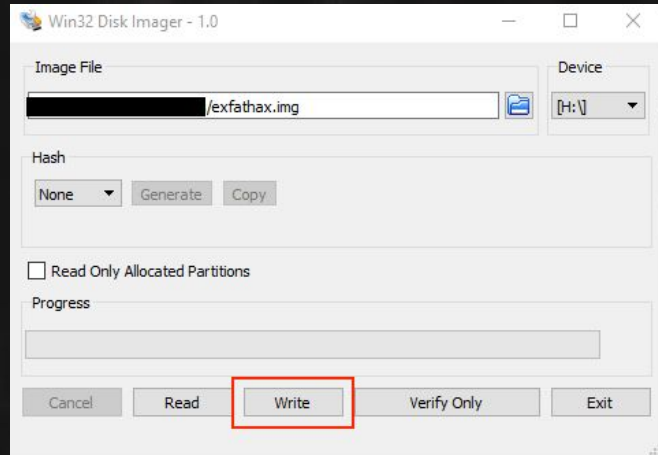
```
2:3050h: 00 00 00 00 03 00 00 00 00 10 00 00 00 00 FF 00 .....ÿ.
2:3060h: 95 03 00 03 16 00 00 00 61 A5 9A 52 61 A5 9A 52  M. ~vĕc~vĕc
```

Variables

Name	Value	Start
struct EXFAT_UPCASE_TABLE UpcaseTable		23040h
struct EXFAT_DIRENTRY_UPCASE_TABLE UpcaseTable...	UpcaseTable (CritPri)	23040h
enum EXFAT_DIRENTRY_TYPE EntryType	UpcaseTable (CritPri)	23040h
UBYTE TypeCode : 5	2h	23040h
UBYTE Benign : 1	0h	23040h
UBYTE Secondary : 1	0h	23040h
UBYTE InUse : 1	1h	23040h
UBYTE _[3]		23041h
DWORD TableChecksum	E619D30Dh	23044h
UBYTE _[12]		23048h
DWORD FirstCluster	3h	23054h
UQUAD DataLength	FF00000001000h	23058h

Where we were - PS4 case study “pOOBs4”

- Full chainable without info disclosure due to weak mitigations
 - No SMAP -> objects can be faked in userspace
 - Don't even need to deal with KASLR
- Code execution achieved with knote function pointer hijack



Where we were - PS4 Post-Exploitation

- After code exec or kernel R/W, nothing stops you from patching whatever you want to run homebrew
 - Secure Access Management Unit (SAMU) handled signing/encryption
 - But the kernel API could be patched to decrypt & verify w/ custom keys
 - System could be used as a decryption oracle for libraries, games, etc.
 - Only certain firmware was protected from this, and mostly irrelevant for homebrew

Where we are

The PlayStation 5



Where we are - PS5 Attack Surface

- Based off FreeBSD 11.0
- Inherits sandboxing improvements made over time to PS4
 - As well as improvements made by FreeBSD to kernel core
- When PS5 was announced, Sony launched an H1 bug bounty
 - Huge step forward for killing easier bugs
- Binaries show evidence of supporting Address Sanitization (ASAN) and fuzzing
- Overall a more mature codebase











Where we are - PS5 Mitigations

- ASLR and SMAP/SMEP have been enforced since launch
- Kernel also has software Control Flow Integrity (CFI)
 - Makes faking objects difficult
 - Also makes code execution post-R/W more annoying
- Bypassable post-R/W on earlier kernels
 - CFI enforce variable was in data segment and writable
 - This was fixed sometime in 3.xx kernels
- Backward-edge control flow is not protected, but...

Where we are - PS5 Mitigations

- eXecute-Only Memory (XOM) / “xotext” mitigate ROP ability
 - Gadgets are difficult to get and may need to be brute forced
 - Hinders reverse engineering efforts pretty significantly
- XOM is enforced both in userspace and kernel






Where we are - PS5 Mitigations

	NX	kASLR	SMAP/SMEP	kCFI	XOM
PS4					
PS5					


Where we are - PS5 case study “pOOBs4”

- SMAP/SMEP
 - We can no longer fake in userspace
- kCFI
 - Have to be careful overwriting function pointers
- Impossible to exploit without separate infoleak
 - Still challenging as USB filesystem would have to be rewritten at runtime to account for kASLR
- The bar for suitable bugs has been raised substantially

Where we are - PS5 case study IPV6 race UAF

711 #826026 **Use-After-Free In IPV6_2292PKTOPTIONS leading To Arbitrary Kernel R/W Primitives** Share:     

TIMELINE

 **theflow0** submitted a report to [PlayStation](#). March 21, 2020(3 years ago)

Summary

Due to missing locks in option `IPV6_2292PKTOPTIONS` of `setsockopt`, it is possible to race and free the `struct ip6_pktopts` buffer, while it is being handled by `ip6_setpktopt`. This structure contains pointers (`ip6po_pktinfo`) that can be hijacked to obtain arbitrary kernel R/W primitives. As a consequence, it is easy to have kernel code execution. This vulnerability is reachable from WebKit sandbox and is available in the latest FW, that is 7.02.

Attachment

Attached is a Proof-Of-Concept that achieves a Local Privilege Escalation on FreeBSD 9 and FreeBSD 12.

Impact

- In conjunction with a WebKit exploit, a fully chained remote attack can be achieved.
- It is possible to steal/manipulate user data.
- Dump and run pirated games.

1 attachment:

F755558: exploit.c

Where we are - PS5 case study IPV6 race UAF

- theflow reported a race condition yielding UAF in IPV6_2292PKTOPTIONS sockopt for INET6 sockets
 - Impacted PS4 <= 7.02 FW
- Fixed before PS5 was released
- *By some miracle, was re-introduced into PS5 3.xx FW via regression*
- Powerful bug that could be used derive infoleak
- Works from 3.00 to 4.51 FW

Where we are - PS5 case study IPV6 race UAF

- theflow implemented this with his bd-j chain he detailed last year
 - Bypassed XOM as native code execution via java was free
- End of 2022 I implemented a (blind to kernel) [exploit in WebKit](#)
 - Webkit ROP gadgets were obtained from an anonymous source who broke PS5 kernel previously
 - Without a dump for at least one FW, likely wouldn't have been possible
 - But gadgets can be bruteforced to port across FW

Where we are - PS5 case study IPV6 race UAF

- PS5 exploit strategy was much the same as PS4
- Infoleak could be derived via overlapping packet info with kqueue
 - Read routing header to dump pointers
- Arbitrary R/W was similarly achievable
 - Set up fake packet info in kernel space with routing header
 - Use packet opts as R/W gadget

Where we are - PS5 case study IPV6 race UAF

- No need to fake objects or execute in userspace
 - Bypass SMAP/SMEP
- Derived infoleak means kASLR can be bypassed immediately
- God-tier bug, shouts to theflow

```
==== Stage 4 - Arbitrary Read/Write =====
[+] Refilled pktopts @ 0x73310163
[+] Found victim sock: 23
[+] Arbitrary kernel read/write should work
[+] Found kqueue .data address: 0xffffffff98048ad3 (found @ i = 0x200)
[+] Found kqueue .data address: 0xffffffff98048ad3 (found @ i = 0x400)
[+] Found kqueue .data address: 0xffffffff98048ad3 (found @ i = 0x600)
[+] Found kqueue .data address: 0xffffffff98048ad3 (found @ i = 0x800)
[+] Found kqueue .data address: 0xffffffff98048ad3 (found @ i = 0xa00)
[+] Found kqueue .data address: 0xffffffff98048ad3 (found @ i = 0xc00)
[+] Found kqueue .data address: 0xffffffff98048ad3 (found @ i = 0xe00)
[+] Test .data read 0xffffffff98048ad3 = 0x420065756575716b
[+] Test .data read 0xffffffff98048adb = 0x206b6e696c206461
[+] Test .data read 0xffffffff98048ae3 = 0x22206e69
[+] PID: 0x4c
[+] Found kernel .data base address: 0xffffffff97d30000
[+] Found allproc: 0xffffffff9a51dcb8
[+] Found proc->p_ucred: 0xffffbabb7b681200
[+] Found proc->p_fd: 0xffffbabb02c427f0
[+] Enabled debug menu
[+] Patched creds
[+] Checking, getuid = 0x0
```

Where we are - PS5 Post-Exploitation

- Post-exploitation is where PS5 looks *really* different from PS4
- Takes advantage of AMD Secure Virtualization (SVM)
 - AMD technology for hardware-backed virtualization
- Hypervisor is a secure monitor and nannies the kernel
 - Intercepts various sensitive actions from the guest kernel
- Basically Virtualization-Based Security for console

Where we are - PS5 Post-Exploitation

- Idea is kernel code integrity cannot be broken without hypervisor bug/bypass
- Kernel code execution is made more difficult
 - XOM cannot be disabled with arbitrary R/W directly
 - Limits gadgets
- Hypervisor is a blackbox
 - Proprietary
 - Unreadable with kernel R/W
- Thanks to [@flat_z](#) I was able to get hypervisor code to study

Hypervisor RE

Reverse Engineering ring-1

```
void hv_vmexit_handler(struct hv_vcpu_int* arg1_1)
```

```
case VMEXIT_MSR
    vmcb_2 = arg1_1->vmcb_ctrl
    uint64_t is_wrmsr = vmcb_2->ctrl.exit_info_1
    if (is_wrmsr == 1)
        struct vmcb_saved_context* gprs = arg1_1->vmcb_context
        if (gprs->rcx.d == MSR_EFER)
            vmcb_2->vmcb_save_state.EFER = zx.q(vmcb_2->vmcb_save_state.RAX.d) | gprs->rdx << 0x20 | 0b10001
            label_ffffffffd9a50884:
            vmcb_2->vmcb_save_state.RIP = vmcb_2->ctrl.next_rip
            goto label_ffffffffd9a4fc44
            label_ffffffffd9a4fbe4:
            vmcb_2->ctrl.event_inj = 0x80000b0d
            goto label_ffffffffd9a4fc44
        if (is_wrmsr != 0)
            goto label_ffffffffd9a4fac6
            goto label_ffffffffd9a4fbe4
case VMEXIT_VMMCALL
    if (vmcb_1->vmcb_save_state.CPL != 0)
        goto err
    enum SCE_HV_VMMCALL_ID vmcall_id = (vmcb_1->vmcb_save_state.RAX).d
    if (vmcall_id > GET_TMR_VIOLATION_ERROR)
        goto err
        int64_t var_58
        int64_t rax_54
        uint64_t rcx_14
        struct vmcb_control_area* rcx_15
        void* rdx_2
        struct vmcb_control_area* rdi_7
        struct vmcb_control_area* r12_2
        switch (vmcall_id)
            case GET_MESSAGE_CONF
                vmcb_ctrl_1 = arg1_1->vmcb_ctrl
                struct vmcb_saved_context* r14_1 = arg1_1->vmcb_context
                uint32_t asid = vmcb_ctrl_1->guest_asid
                bool z_1
                if (0 == data_ffffffffd9ae4a40)
                    data_ffffffffd9ae4a40 = 1
                    z_1 = true
                else
                    data_ffffffffd9ae4a40
                    z_1 = false
                if (z_1 == 0)
                    bool z_2
                    do
                        if (data_ffffffffd9ae4a40 != 0)
                            while (data_ffffffffd9ae4a40 != 0)
                                nop
                        if (0 == data_ffffffffd9ae4a40)
                            data_ffffffffd9ae4a40 = 1
                            z_2 = true
                        else
                            data_ffffffffd9ae4a40
                            z_2 = false
                    while (z_2 == 0)
                char rax_10
                if (asid == 1)
                    rcx_10 = data_ffffffffd9ae4fc4
```

Hypervisor

- x86 FreeBSD kernel runs as guest
- Embedded as part of the kernel on lower firmwares
 - On higher firmwares it's loaded separately but is still similar in function
- **not** bhyve-based, completely custom
- Very small
 - 14 hypercalls < 3.xx FW

Hypervisor

Hypervisor

vmmcalls (VMMCALL_HV_*)

0	GET_MESSAGE_CONF
1	GET_MESSAGE_COUNT
2	START_LOADING_SELF
3	FINISH_LOADING_SELF
4	SET_CPUID_PS4
5	SET_CPUID_PPR
6	IOMMU_SET_GUEST_BUFFERS
7	IOMMU_ENABLE_DEVICE
8	IOMMU_BIND_PASID
9	IOMMU_UNBIND_PASID
0xa	IOMMU_CHECK_CMD_COMPLETION
0xb	IOMMU_CHECK_EVLOG_REGS
0xc	IOMMU_READ_DEVICE_TABLE
0xd	GET_TMR_VIOLATION_ERROR
0xe	VMCLOSURE_INVOCATION (03.00.00.33 and above)
0xf	STARTUP_MP (03.00.00.33 and above)
0x10	DISABLE_STARTUP_MP (03.00.00.33 and above)

Calltree

Current P R

hv_init

Incoming Calls

hv

Outgoing Calls

- hv_unk_2
 - hv_iommu_init_hv_hw
 - hv_vcpu_run
 - hv_vmexit_handler
 - hv_extract_bits
 - hv_vtophys
 - hv_alloc_pages

- hv_unk_3
- hv_vtophys
 - hv_alloc_pages
- hv_iommu_init_hv_hw
- hv_vcpu_run
 - hv_vmexit_handler
 - hv_vtophys
 - hv_alloc_pages
- hv_unk_1

int64_t hv_alloc_pages(int32_t num_pages)

```
int64_t pages = sx.q(num_pages)
int64_t cur_page_ptr_aligned
bool z_1
do
    int64_t cur_page_ptr = g_hv_page_cur
    cur_page_ptr_aligned = (cur_page_ptr + 0xffff) & 0xffffffffffff000
    if ((&hv_page_end - cur_page_ptr_aligned + 0x100b000) s>> 0xc <= pages)
        prepanic()
        sub_ffffffffffd968e730(&data_ffffffffffd9a510c4)
        panic(7, "[HV] hv_alloc_pages failed")
        noreturn
    if (cur_page_ptr == g_hv_page_cur)
        g_hv_page_cur = cur_page_ptr_aligned + (pages << 0xc)
        z_1 = true
    else
        g_hv_page_cur
        z_1 = false
    while (z_1 == 0)
return cur_page_ptr_aligned
```

- 4KB pages
- 4107 pages allocatable
- ~16.4MB total HV data pages

Hypervisor

```
void hv_init_pagetables()
{
    int cpuid_field = *data_ffffffffddc4f2d4; // __cpuid(1, temp2);
    int vmcr = __rdmsr(MSR_VM_CR);

    if (((uint8_t) cpuid_field & 0x4) == 0)
        panic(0xF, "[HV] SVM feature is not available\n");

    if (vmcr & VM_CR_SVMDIS)
        panic(0x13, "[HV] SVM feature is disabled by BIOS\n");

    if ((__cpuid(eax: 0x80000008, ecx: 0xC0010114)->ebx & 0x20) == 0)
        panic(0x17, "[HV] NDA feature is not available");

    struct cpuid *cpuid_8000000A_val = __cpuid(eax: 0x8000000A);
    g_svm_features = cpuid_8000000A_val->edx;
    if (cpuid_8000000A_val->ebx <= 1)
        panic(0x1B, "Insufficient ASIDs: %#X", cpuid_8000000a_val->ebx);

    if ((g_svm_features & 0x20E9) != 0x20E9)
        panic(0x1F, "[HV] the processor lacks a required SVM feature: %#x",
            ~(g_svm_features & 0x20E9));

    build_hv_page_tables();
    build_nested_page_tables();
    // ...
    hv_iommu_init_hv_hw();
}
```

- Can't boot w/o SVM
- Uses 'NDA' feature (likely "xotext")
- Two page tables
 - HV pages
 - Nested Page Tables (NPT)

Hypervisor

```
void hv_init_pagetables()
{
    int cpuid_field = *data_ffffffffddc4f2d4; // __cpuid(1, temp2);
    int vmcr = __rdmsr(MSR_VM_CR);

    if (((uint8_t) cpuid_field & 0x4) == 0)
        panic(0xF, "[HV] SVM feature is not available\n");

    if (vmcr & VM_CR_SVMDIS)
        panic(0x13, "[HV] SVM feature is disabled by BIOS\n");

    if ((__cpuid(eax: 0x80000008, ecx: 0xC0010114)->ebx & 0x20) == 0)
        panic(0x17, "[HV] NDA feature is not available");

    struct cpuid *cpuid_8000000A_val = __cpuid(eax: 0x8000000A);
    g_svm_features = cpuid_8000000A_val->edx;
    if (cpuid_8000000A_val->ebx <= 1)
        panic(0x1B, "Insufficient ASIDs: %#X", cpuid_8000000a_val->ebx);

    if ((g_svm_features & 0x20E9) != 0x20E9)
        panic(0x1F, "[HV] the processor lacks a required SVM feature: %#x",
            ~(g_svm_features & 0x20E9));

    build_hv_page_tables();
    build_nested_page_tables();
    // ...
    hv_iommu_init_hv_hw();
}
```

- **Can't boot w/o SVM**
- Uses 'NDA' feature (likely "xotext")
- Two page tables
 - HV pages
 - Nested Page Tables (NPT)

Hypervisor

```
void hv_init_pagetables()
{
    int cpuid_field = *data_ffffffffddc4f2d4; // __cpuid(1, temp2);
    int vmcr = __rdmsr(MSR_VM_CR);

    if (((uint8_t) cpuid_field & 0x4) == 0)
        panic(0xF, "[HV] SVM feature is not available\n");

    if (vmcr & VM_CR_SVMDIS)
        panic(0x13, "[HV] SVM feature is disabled by BIOS\n");

    if ((__cpuid(eax: 0x80000008, ecx: 0xC0010114)->ebx & 0x20) == 0)
        panic(0x17, "[HV] NDA feature is not available");

    struct cpuid *cpuid_8000000A_val = __cpuid(eax: 0x8000000A);
    g_svm_features = cpuid_8000000A_val->edx;
    if (cpuid_8000000A_val->ebx <= 1)
        panic(0x1B, "Insufficient ASIDs: %#X", cpuid_8000000a_val->ebx);

    if ((g_svm_features & 0x20E9) != 0x20E9)
        panic(0x1F, "[HV] the processor lacks a required SVM feature: %#x",
            ~(g_svm_features & 0x20E9));

    build_hv_page_tables();
    build_nested_page_tables();
    // ...
    hv_iommu_init_hv_hw();
}
```

- Can't boot w/o SVM
- **Uses 'NDA' feature (likely "xotext")**
- Two page tables
 - HV pages
 - Nested Page Tables (NPT)

Hypervisor

```
void hv_init_pagetables()
{
    int cpuid_field = *data_ffffffffddc4f2d4; // __cpuid(1, temp2);
    int vmcr = __rdmsr(MSR_VM_CR);

    if (((uint8_t) cpuid_field & 0x4) == 0)
        panic(0xF, "[HV] SVM feature is not available\n");

    if (vmcr & VM_CR_SVMDIS)
        panic(0x13, "[HV] SVM feature is disabled by BIOS\n");

    if ((__cpuid(eax: 0x80000008, ecx: 0xC0010114)->ebx & 0x20) == 0)
        panic(0x17, "[HV] NDA feature is not available");

    struct cpuid *cpuid_8000000A_val = __cpuid(eax: 0x8000000A);
    g_svm_features = cpuid_8000000A_val->edx;
    if (cpuid_8000000A_val->ebx <= 1)
        panic(0x1B, "Insufficient ASIDs: %#X", cpuid_8000000a_val->ebx);

    if ((g_svm_features & 0x20E9) != 0x20E9)
        panic(0x1F, "[HV] the processor lacks a required SVM feature: %#x",
            ~(g_svm_features & 0x20E9));

    build_hv_page_tables();
    build_nested_page_tables();
    // ...
    hv_iommu_init_hv_hw();
}
```

- Can't boot w/o SVM
- Uses 'NDA' feature (likely "xotext")
- **Two page tables**
 - **HV pages**
 - **Nested Page Tables (NPT)**

Hypervisor - Paging

```
inline void build_hv_page_tables()
{
    uint64_t *hv_pml4_table = hv_alloc_pages(1);
    bzero(pml4_table, 0x1000);

    uint64_t pml4_map_pa = pmap_kextract(g_pml4);
    uint64_t *pml4_map = (0xFFFF800000000000 |
                          zx.q(data_ffffffffddc23b54) << PDPSHIFT |
                          zx.q(data_ffffffffddc23b50) << PML4SHIFT) + pml4_map_pa;

    for (int i = 0x100; i < 0x200; i++) {
        uint64_t page_addr = pml4_map[i];

        if (((uint8_t) page_addr & 1) != 0 && i != PML4PML4I) {
            // Recursively generate PDEs and PTEs
            uint64_t pde = hv_generate_pde_pte(
                addr: page_addr & 0xFFFFFFFF000,
                level: 3);

            pde |= PG_V | PG_RW | PG_U;
            hv_pml4_table[i] = pde;
        }
    }

    g_hv_cr3 = hv_pml4_table;
}
```

- Copies FreeBSD page tables
- Everything is mapped as R/W
- Kernel mapping's xotext bit not set

Hypervisor - Paging

```
inline void build_hv_page_tables()
{
    uint64_t *hv_pml4_table = hv_alloc_pages(1);
    bzero(pml4_table, 0x1000);

    uint64_t pml4_map_pa = pmap_kextract(g_pml4);
    uint64_t *pml4_map = (0xFFFF800000000000 |
                          zx.q(data_ffffffffddc23b54) << PDPSHIFT |
                          zx.q(data_ffffffffddc23b50) << PML4SHIFT) + pml4_map_pa;

    for (int i = 0x100; i < 0x200; i++) {
        uint64_t page_addr = pml4_map[i];

        if (((uint8_t) page_addr & 1) != 0 && i != PML4PML4I) {
            // Recursively generate PDEs and PTEs
            uint64_t pde = hv_generate_pde_pte(
                addr: page_addr & 0xFFFFFFFF000,
                level: 3);

            pde |= PG_V | PG_RW | PG_U;
            hv_pml4_table[i] = pde;
        }
    }

    g_hv_cr3 = hv_pml4_table;
}
```

- Copies FreeBSD page tables
- Everything is mapped as R/W
- Kernel mapping's xotext bit not set

Hypervisor - Paging

```
inline void build_hv_page_tables()
{
    uint64_t *hv_pml4_table = hv_alloc_pages(1);
    bzero(pml4_table, 0x1000);

    uint64_t pml4_map_pa = pmap_kextract(g_pml4);
    uint64_t *pml4_map = (0xFFFF800000000000 |
                          zx.q(data_ffffffffddc23b54) << PDPSHIFT |
                          zx.q(data_ffffffffddc23b50) << PML4SHIFT) + pml4_map_pa;

    for (int i = 0x100; i < 0x200; i++) {
        uint64_t page_addr = pml4_map[i];

        if (((uint8_t) page_addr & 1) != 0 && i != PML4PML4I) {
            // Recursively generate PDEs and PTEs
            uint64_t pde = hv_generate_pde_pte(
                addr: page_addr & 0xFFFFFFFF000,
                level: 3);

            pde |= PG_V | PG_RW | PG_U;
            hv_pml4_table[i] = pde;
        }
    }

    g_hv_cr3 = hv_pml4_table;
}
```

- Copies FreeBSD page tables
- **Everything is mapped as R/W**
- **Kernel mapping's xotext bit not set**

Hypervisor - Nested Paging

```
for (int pdp_index = 0; pdp_index < 0x200; pdp_index++) {
    // [...] PDP setup
    uint64_t *npt_pdr_table = hv_alloc_pages(1);
    npt_pdp_table[pdp_index] = hv_vtophys(npt_pdr_table) | PG_V | PG_RW | PG_U;

    for (int pdr_index = 0; pdr_index < 0x200; i++) {
        // [...] PDR setup
        for (int pte_index = 0; pte_index < 0x200000; pte_index += 0x1000) {
            uint64_t pte_bits = pte_index | pdr;
            uint64_t pte;

            if (pte_bits < hv_page_start || pte_bits >= r15_3) {
                if (&kernel_hv_load_addr - rax_14 > pte_bits ||
                    pte_bits >= &hv_page_end - rax_14) {
                    // user pages
                    pte = 0;

                    if ((pte_bits & 0x7fffffffffff8000) != 0xfdd80000) {
                        pte = pte_bits | PG_V | PG_RW | PG_U; // user, r/w, paged
                    }
                } else {
                    // kernel text pages
                    pte = pte_bits | kernel_text_page_bits;
                }
            } else {
                // don't map hv pages to guest
                pte = 0;
            }

            npt_pte_table[i / 0x1000] = pte;
        }
    }
}

g_npt_cr3 = hv_vtophys(npt_pml4_table);
}
```

- HV pages not mapped
- Kernel text is nested
- Enforces XOM on guest kernel

Hypervisor - Nested Paging

```
for (int pdp_index = 0; pdp_index < 0x200; pdp_index++) {
    // [...] PDP setup
    uint64_t *npt_pdr_table = hv_alloc_pages(1);
    npt_pdp_table[pdp_index] = hv_vtophys(npt_pdr_table) | PG_V | PG_RW | PG_U;

    for (int pdr_index = 0; pdr_index < 0x200; i++) {
        // [...] PDR setup
        for (int pte_index = 0; pte_index < 0x200000; pte_index += 0x1000) {
            uint64_t pte_bits = pte_index | pdr;
            uint64_t pte;

            if (pte_bits < hv_page_start || pte_bits >= r15_3) {
                if (&kernel_hv_load_addr - rax_14 > pte_bits ||
                    pte_bits >= &hv_page_end - rax_14) {
                    // user pages
                    pte = 0;

                    if ((pte_bits & 0x7fffffffffff8000) != 0xfdd80000) {
                        pte = pte_bits | PG_V | PG_RW | PG_U; // user, r/w, paged
                    }
                } else {
                    // kernel text pages
                    pte = pte_bits | kernel_text_page_bits;
                }
            } else {
                // don't map hv pages to guest
                pte = 0;
            }

            npt_pte_table[i / 0x1000] = pte;
        }
    }
}

g_npt_cr3 = hv_vtophys(npt_pml4_table);
}
```

- HV pages not mapped
- Kernel text is nested
- Enforces XOM on guest kernel

Hypervisor - Nested Paging

```
for (int pdp_index = 0; pdp_index < 0x200; pdp_index++) {
    // [...] PDP setup
    uint64_t *npt_pdr_table = hv_alloc_pages(1);
    npt_pdp_table[pdp_index] = hv_vtophys(npt_pdr_table) | PG_V | PG_RW | PG_U;

    for (int pdr_index = 0; pdr_index < 0x200; i++) {
        // [...] PDR setup
        for (int pte_index = 0; pte_index < 0x200000; pte_index += 0x1000) {
            uint64_t pte_bits = pte_index | pdr;
            uint64_t pte;

            if (pte_bits < hv_page_start || pte_bits >= r15_3) {
                if (&kernel_hv_load_addr - rax_14 > pte_bits ||
                    pte_bits >= &hv_page_end - rax_14) {
                    // user pages
                    pte = 0;

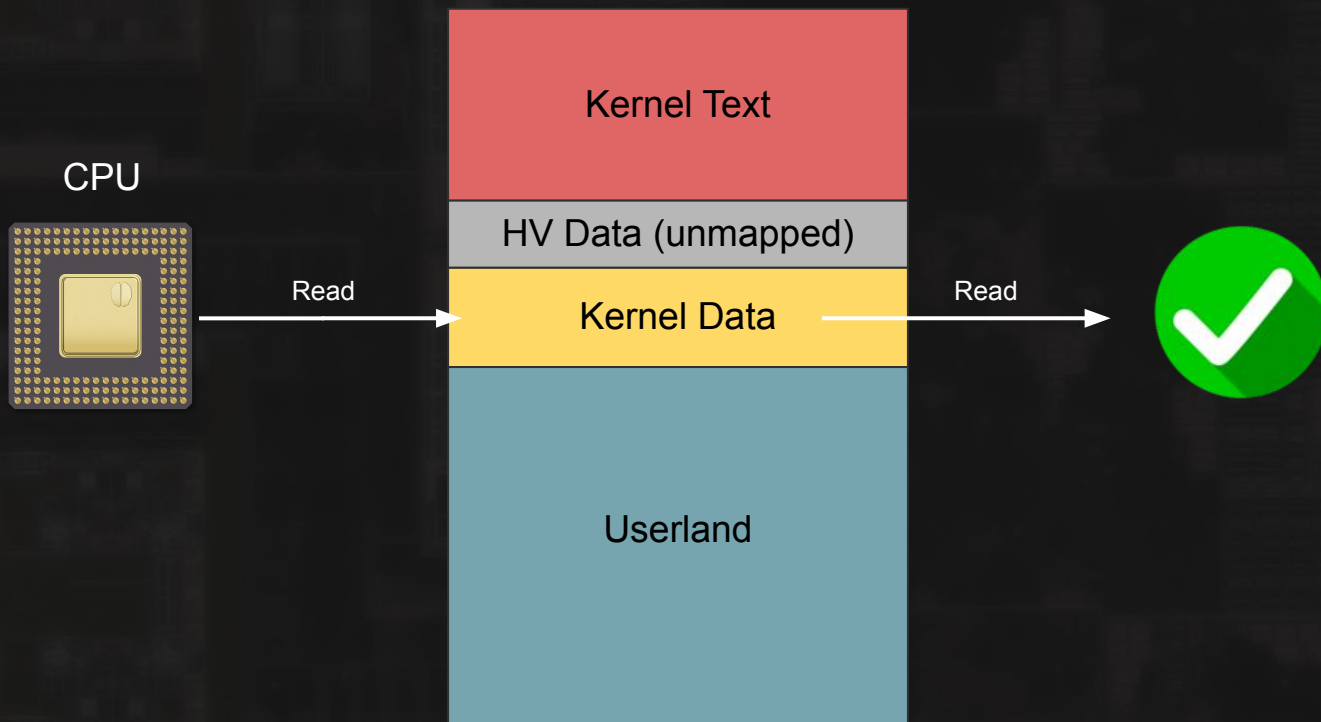
                    if ((pte_bits & 0x7fffffffffff8000) != 0xfdd80000) {
                        pte = pte_bits | PG_V | PG_RW | PG_U; // user, r/w, paged
                    }
                } else {
                    // kernel text pages
                    pte = pte_bits | kernel_text_page_bits;
                }
            } else {
                // don't map hv pages to guest
                pte = 0;
            }

            npt_pte_table[i / 0x1000] = pte;
        }
    }
}

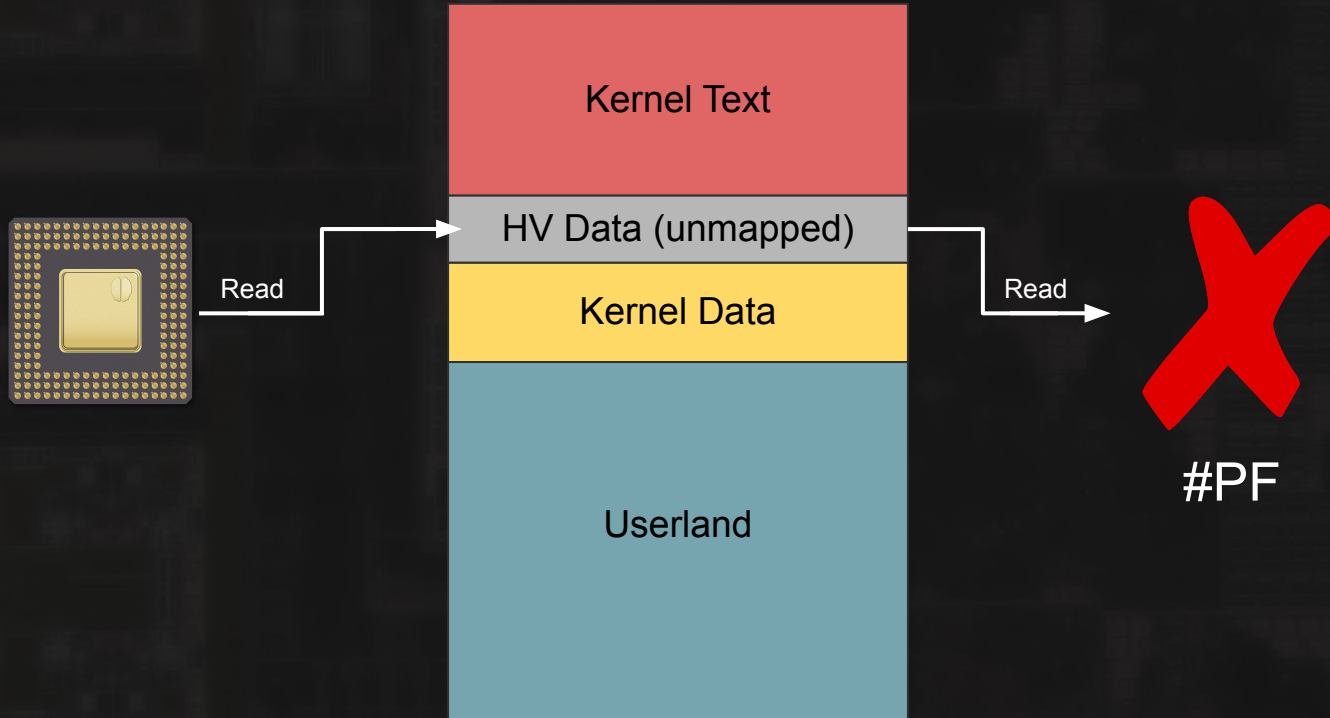
g_npt_cr3 = hv_vtophys(npt_pml4_table);
}
```

- HV pages not mapped
- Kernel text is nested
- **Enforces XOM on guest kernel**

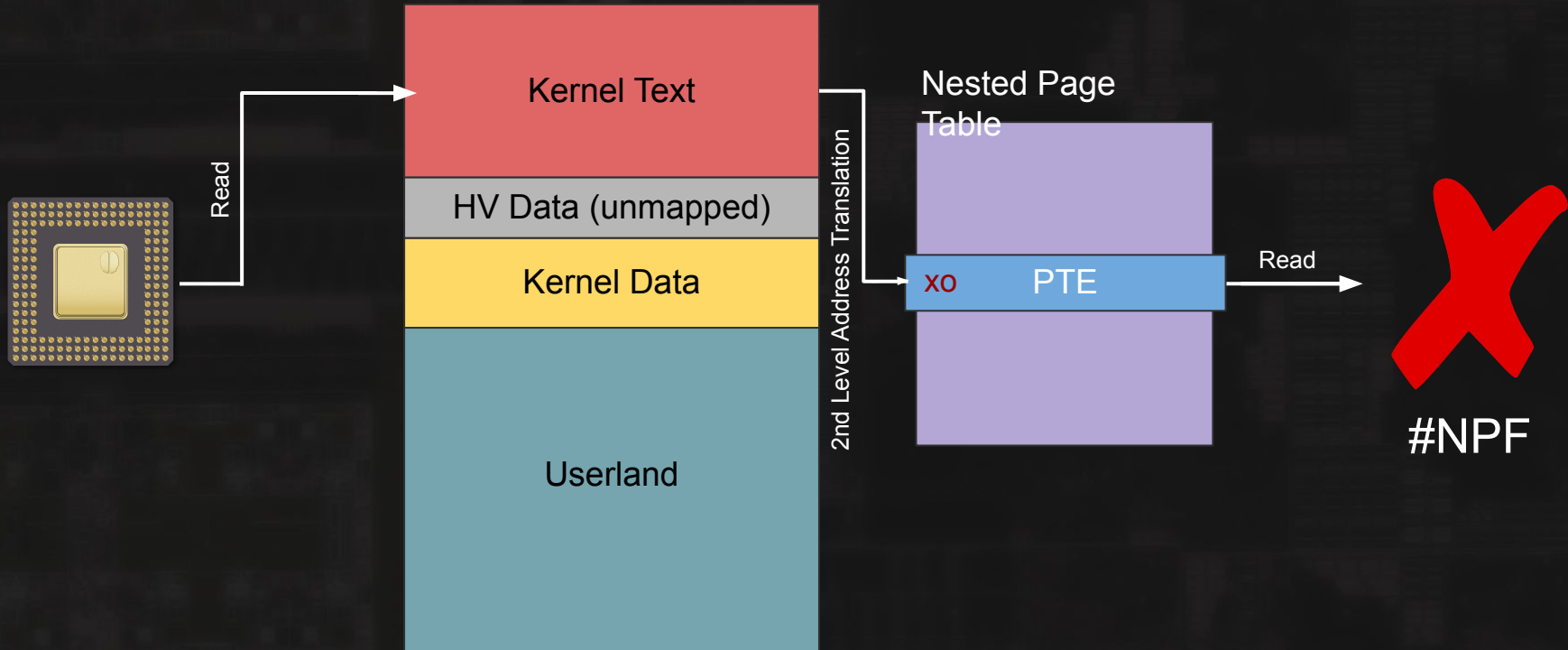
Hypervisor - Guest Kernel Reads



Hypervisor - Guest Kernel Reads



Hypervisor - Guest Kernel Reads



Hypervisor - VM Setup

```
void hv_iommu_init_hv_hw()
{
    int cpuid = *(gsbase + 0x34);
    if (cpuid > 0x10)
        panic(0xB, "cpuid %d HV_NCPUS %d", cpuid, 0x10);

    // ...
    hv_init_msrs();
    struct vmcb *vmcb = hv_init_vmcb();

    if (cpuid == 0) {
        hv_init_iommu();
    }

    hv_vcpu_run(vmcb); // noreturn, HV main loop
}
```

- Initializes Machine State Registers (MSRs)
- Virtual Machine Control Block (VMCB) Setup
- IOMMU Init
 - Skip this for today
 - Mostly for M.2 SSD
- HV main loop (intercepts)

Hypervisor - VM Setup

```
void hv_iommu_init_hv_hw()
{
    int cpuid = *(gsbase + 0x34);
    if (cpuid > 0x10)
        panic(0xB, "cpuid %d HV_NCPUS %d", cpuid, 0x10);

    // ...
    hv_init_msrs();
    struct vmcb *vmcb = hv_init_vmcb();

    if (cpuid == 0) {
        hv_init_iommu();
    }

    hv_vcpu_run(vmcb); // noreturn, HV main loop
}
```

- **Initializes Machine State Registers (MSRs)**
- **Virtual Machine Control Block (VMCB) Setup**
- IOMMU Init
 - Skip this for today
 - Mostly for M.2 SSD
- HV main loop (intercepts)

Hypervisor - VM Setup

```
void hv_iommu_init_hv_hw()
{
    int cpuid = *(gsbase + 0x34);
    if (cpuid > 0x10)
        panic(0xB, "cpuid %d HV_NCPUS %d", cpuid, 0x10);

    // ...
    hv_init_msrs();
    struct vmcb *vmcb = hv_init_vmcb();

    if (cpuid == 0) {
        hv_init_iommu();
    }

    hv_vcpu_run(vmcb); // noreturn, HV main loop
}
```

- Initializes Machine State Registers (MSRs)
- Virtual Machine Control Block (VMCB) Setup
- **IOMMU Init**
 - **Skip this for today**
 - **Mostly for M.2 SSD**
- HV main loop (intercepts)

Hypervisor - VM Setup

```
void hv_iommu_init_hv_hw()
{
    int cpuid = *(gsbase + 0x34);
    if (cpuid > 0x10)
        panic(0xB, "cpuid %d HV_NCPUS %d", cpuid, 0x10);

    // ...
    hv_init_msrs();
    struct vmcb *vmcb = hv_init_vmcb();

    if (cpuid == 0) {
        hv_init_iommu();
    }

    hv_vcpu_run(vmcb); // noreturn, HV main loop
}
```

- Initializes Machine State Registers (MSRs)
- Virtual Machine Control Block (VMCB) Setup
- IOMMU Init
 - Skip this for today
 - Mostly for M.2 SSD
- **HV main loop (intercepts)**

Hypervisor - VM Setup

```
inline hv_init_msrs()
{
    // Enable SVM and XOM
    uint32_t efer = __rdmsr(MSR_EFER);
    __wrmsr(MSR_EFER, efer | 0b1000100000000000); // 12 = svm enable, 16 = xotext

    // Core performance boost custom feature
    uint32_t cpb_dis = __rdmsr(MSR_CPB_DIS);
    __wrmsr(MSR_CPB_DIS, cpb_dis | 0b1000000000000000); // 18 = custom bit

    // Initialize host state save area
    uint64_t hsave_area = hv_alloc_pages(1);
    bzero(hsave_area, 0x1000);
    // ...
    __wrmsr(hv_vtophys(vaddr: hsave_area), MSR_VM_HSAVE_PA);
}
```

- Enables SVM and XOM
- Sets up host save state area

Hypervisor - VM Setup

```
inline hv_init_msrs()
{
    // Enable SVM and XOM
    uint32_t efer = __rdmsr(MSR_EFER);
    __wrmsr(MSR_EFER, efer | 0b1000100000000000); // 12 = svm enable, 16 = xotext

    // Core performance boost custom feature
    uint32_t cpb_dis = __rdmsr(MSR_CPB_DIS);
    __wrmsr(MSR_CPB_DIS, cpb_dis | 0b1000000000000000); // 18 = custom bit

    // Initialize host state save area
    uint64_t hsave_area = hv_alloc_pages(1);
    bzero(hsave_area, 0x1000);
    // ...
    __wrmsr(hv_vtophys(vaddr: hsave_area), MSR_VM_HSAVE_PA);
}
```

- **Enables SVM and XOM**
- Sets up host save state area

Hypervisor - VM Setup

```
inline hv_init_msrs()
{
    // Enable SVM and XOM
    uint32_t efer = __rdmsr(MSR_EFER);
    __wrmsr(MSR_EFER, efer | 0b1000100000000000); // 12 = svm enable, 16 = xotext

    // Core performance boost custom feature
    uint32_t cpb_dis = __rdmsr(MSR_CPB_DIS);
    __wrmsr(MSR_CPB_DIS, cpb_dis | 0b1000000000000000); // 18 = custom bit

    // Initialize host state save area
    uint64_t hsave_area = hv_alloc_pages(1);
    bzero(hsave_area, 0x1000);
    // ...
    __wrmsr(hv_vtophys(vaddr: hsave_area), MSR_VM_HSAVE_PA);
}
```

- Enables SVM and XOM
- **Sets up host save state area**

Hypervisor - VM Setup

```
inline struct vmcb *hv_init_vmcb()
{
    struct vmcb *vmcb = hv_alloc_pages(1);
    *(uint64_t *) (offset + 0xffffffffd9ae1398) = vmcb;
    bzero(vmcb, 0x1000);

    // Intercept CR4
    vmcb->ctrl.vec_0_cr = 0b100000000000000000000000;

    // Intercept select CR0 writes, INVD, MSR_PROT
    vmcb->ctrl.vec_3 = 0b100000100000000000000000100000;

    // Intercept VMRUN, VMCALL, VMLoad, VMSAVE, STGI, CLGI, SKINIT
    // Optionally intercept RDPRU if not devkit
    vmcb->ctrl.vec_4 = (sceKernelIsDevKit() == 0) << 0xE | 0b11111111;

    vmcb->ctrl.iopm_base_pa = hv_vtophys(&g_iopm);
    vmcb->ctrl.msrpm_base_pa = hv_vtophys(&g_msrpm);
    vmcb->ctrl.guest_asid = 0x1; // ASID
    vmcb->ctrl.tlb_control = 0x3; // Flush TLB entries
    vmcb->enable_misc_vector |= 0b1001; // Enable nested paging + GMET

    vmcb->ctrl.npt_cr3 = g_npt_cr3; // Setup in hv_build_nested_page_tables()
    // [...] other CRs/registers mostly just copied
    return vmcb;
}
```

- Intercepts Control Register + MSR Writes
 - CR0, CR4
- Supports Hypercalls
- Enables Nested Paging / SLAT
- Enables Guest Mode Execute Trap (GMET)

Hypervisor - VM Setup

```
inline struct vmcb *hv_init_vmcb()
{
    struct vmcb *vmcb = hv_alloc_pages(1);
    *(uint64_t *) (offset + 0xffffffffd9ae1398) = vmcb;
    bzero(vmcb, 0x1000);

    // Intercept CR4
    vmcb->ctrl.vec_0_cr = 0b100000000000000000000000;

    // Intercept select CR0 writes, INVD, MSR_PROT
    vmcb->ctrl.vec_3 = 0b1000001000000000000000000100000;

    // Intercept VMRUN, VMCALL, VMLoad, VMSAVE, STGI, CLGI, SKINIT
    // Optionally intercept RDPRU if not devkit
    vmcb->ctrl.vec_4 = (sceKernelIsDevKit() == 0) << 0xE | 0b11111111;

    vmcb->ctrl.iopm_base_pa = hv_vtophys(&g_iopm);
    vmcb->ctrl.msrpm_base_pa = hv_vtophys(&g_msrpm);
    vmcb->ctrl.guest_asid = 0x1; // ASID
    vmcb->ctrl.tlb_control = 0x3; // Flush TLB entries
    vmcb->enable_misc_vector |= 0b1001; // Enable nested paging + GMET

    vmcb->ctrl.npt_cr3 = g_npt_cr3; // Setup in hv_build_nested_page_tables()
    // [...] other CRs/registers mostly just copied
    return vmcb;
}
```

- **Intercepts Control Register + MSR Writes**
 - **CR0, CR4**
- Supports Hypercalls
- Enables Nested Paging / SLAT
- Enables Guest Mode Execute Trap (GMET)

Hypervisor - VM Setup

```
inline struct vmcb *hv_init_vmcb()
{
    struct vmcb *vmcb = hv_alloc_pages(1);
    *(uint64_t *) (offset + 0xffffffffd9ae1398) = vmcb;
    bzero(vmcb, 0x1000);

    // Intercept CR4
    vmcb->ctrl.vec_0_cr = 0b100000000000000000000000;

    // Intercept select CR0 writes, INVD, MSR_PROT
    vmcb->ctrl.vec_3 = 0b100000100000000000000000100000;

    // Intercept VMRUN, VMCALL, VMLoad, VMSAVE, STGI, CLGI, SKINIT
    // Optionally intercept RDPRU if not devkit
    vmcb->ctrl.vec_4 = (sceKernelIsDevKit() == 0) << 0xE | 0b11111111;

    vmcb->ctrl.iopm_base_pa = hv_vtophys(&g_iopm);
    vmcb->ctrl.msrpm_base_pa = hv_vtophys(&g_msrpm);
    vmcb->ctrl.guest_asid = 0x1; // ASID
    vmcb->ctrl.tlb_control = 0x3; // Flush TLB entries

    vmcb->enable_misc_vector |= 0b1001; // Enable nested paging + GMET

    vmcb->ctrl.npt_cr3 = g_npt_cr3; // Setup in hv_build_nested_page_tables()
    // [...] Other CRs/registers mostly just copied
    return vmcb;
}
```

- Intercepts Control Register + MSR Writes
 - CR0, CR4
- Supports Hypercalls
- **Enables Nested Paging / SLAT**
- **Enables Guest Mode Execute Trap (GMET)**

Hypervisor - Guest Mode Execute Trap

15.25.13 Guest Mode Execute Trap Extension

The Guest Mode Execute Trap (GMET) extension allows a hypervisor to cause nested page faults on attempts by a guest to execute code at CPL0, 1 or 2 from pages designated by the hypervisor. The presence of the GMET extension is indicated by CPUID Fn8000_000A EDX[17]=1. The GMET mode is selected for a targeted guest by setting bit 3 of VMCB offset 090h to 1. For processors that don't support GMET this bit is ignored.

On GMET capable processors, when this bit is set to 1 on a VMRUN, the processor changes how the U/S bit in the nested page table is interpreted. The NX bit still prohibits execution of code at any privilege level when set to 1. However, with GMET enabled and the effective NX bit =0, if the effective U/S bit =1 and the page is being accessed for execution at CPL0, 1 or 2, a nested page fault #VMEXIT(NPF) is generated. If the effective NX bit =0 and the effective U/S bit =0 then the ...

<https://www.amd.com/system/files/TechDocs/24593.pdf>

Hypervisor - Guest Mode Execute Trap

15.25.13 Guest Mode Execute Trap Extension

The Guest Mode Execute Trap (GMET) extension allows a hypervisor to cause nested page faults on attempts by a guest to execute code at CPL0, 1 or 2 from pages designated by the hypervisor. The presence of the GMET extension is indicated by CPUID Fn8000_000A EDX[17]=1. The GMET mode is selected for a targeted guest by setting bit 3 of VMCB offset 090h to 1. For processors that don't support GMET this bit is ignored.

On GMET capable processors, when this bit is set to 1 on a VMRUN, the processor changes how the U/S bit in the nested page table is interpreted. The NX bit still prohibits execution of code at any privilege level when set to 1. However, with GMET enabled and the effective NX bit =0, if the effective U/S bit =1 and the page is being accessed for execution at CPL0, 1 or 2, a nested page fault #VMEXIT(NPF) is generated. If the effective NX bit =0 and the effective U/S bit =0 then the ...

<https://www.amd.com/system/files/TechDocs/24593.pdf>

Hypervisor - Guest Mode Execute Trap

Table 15-21. GMET Page Configuration

nPT NX Bit	nPT U/S Bit	Guest User-Mode Code	Guest Supervisor-Mode Code
1	X	No Execute	No Execute
0	1	Execute	No Execute
0	0	Execute	Execute

- We can't execute user pages in kernel context
 - Even if SMEP is disabled/bypassed
- Makes it harder to subvert kernel control flow

Hypervisor - Intercepts

- Intercepts obvious attack

Hypervisor - Intercepts

- Intercepts obvious attack
- **Protected MSRs**

```
1 void hv_vmexit_handler(struct hv_vcpu_int *vcpu)
2 {
3     struct vmcb *vmcb = vcpu->vmcb_ctrl;
4     uint32_t exit_code = (uint32_t) vmcb->ctrl.exit_code;
5
6     switch (exit_code) {
7         // ...
8
9         case VMEXIT_MSR:
10            int is_wrmsr = vmcb->ctrl.exit_info1;
11            if (is_wrmsr) {
12                struct vmcb_saved_context *gprs = vcpu->vmcb_context;
13
14                if (gprs->rcx == MSR_EFER) {
15                    vmcb->save_state.EFER = vmcb->save_state.RAX |
16                    gprs->RDX << 0x20 |
17                    0b100011000000000000;
18                    vmcb->save_state.RIP = vmcb->ctrl.next_rip;
19                    return;
20                }
21            } else {
22                vmcb->ctrl.event_inj = 0x80000306;
23                return;
24            }
25
26            vmcb->ctrl.event_inj = 0x80000B0D;
27            return;
28
29            // ...
30        }
31    }
```

Hypervisor - Intercepts

- Intercepts obvious attack
- **Protected MSRs**
 - Extended Features (EFER) is masked

```
1 void hv_vmexit_handler(struct hv_vcpu_int *vcpu)
2 {
3     struct vmcb *vmcb = vcpu->vmcb_ctrl;
4     uint32_t exit_code = (uint32_t) vmcb->ctrl.exit_code;
5
6     switch (exit_code) {
7         // ...
8
9         case VMEXIT_MSR:
10            int is_wrmsr = vmcb->ctrl.exit_info1;
11            if (is_wrmsr) {
12                struct vmcb_saved_context *gprs = vcpu->vmcb_context;
13
14                if (gprs->rcx == MSR_EFER) {
15                    vmcb->save_state.EFER = vmcb->save_state.RAX |
16                        gprs->RDX << 0x20 |
17                        0b100011000000000000;
18                    = vmcb->ctrl.next_rip;
19                }
20            }
21        }
22    }
```

Bits	Mnemonic	Description	Access type
63:22	Reserved		MBZ
21	AIBRSE	Automatic IBRS Enable	R/W
20	UAIE	Upper Address Ignore Enable	R/W
19	Reserved		MBZ
18	INTWB	Interruptible WBINVD/WBNOINVD enable	R/W
17	MCOMMIT	Enable MCOMMIT instruction	R/W
16	Reserved	NDA xotext	MBZ
15	TCE	Translation Cache Extension	R/W
14	FFXSR	Fast FXSAVE/FXRSTOR	R/W
13	LMSLE	Long Mode Segment Limit Enable	R/W
12	SVME	Secure Virtual Machine Enable	R/W
11	NXE	No-Execute Enable	R/W
10	LMA	Long Mode Active	R/W
9	Reserved		MBZ
8	LME	Long Mode Enable	R/W
7:1	Reserved		RAZ
0	SCE	System Call Extensions	R/W

Hypervisor - Intercepts

- Intercepts obvious attack
- **Protected MSRs**
 - Other protected regs are forbidden

```
1 void hv_vmexit_handler(struct hv_vcpu_int *vcpu)
2 {
3     struct vmcb *vmcb = vcpu->vmcb_ctrl;
4     uint32_t exit_code = (uint32_t) vmcb->ctrl.exit_code;
5
6     switch (exit_code) {
7         // ...
8
9         case VMEXIT_MSR:
10            int is_wrmsr = vmcb->ctrl.exit_info1;
11            if (is_wrmsr) {
12                struct vmcb_saved_context *gprs = vcpu->vmcb_context;
13
14                if (gprs->rcx == MSR_EFER) {
15                    vmcb->save_state.EFER = vmcb->save_state.RAX |
16                                            gprs->RDX << 0x20 |
17                                            0b100011000000000000;
18                    vmcb->save_state.RIP = vmcb->ctrl.next_rip;
19                    return;
20                }
21            } else {
22                vmcb->ctrl.event_inj = 0x80000306;
23                return;
24            }
25
26            vmcb->ctrl.event_inj = 0x80000B0D;
27            return;
28
29            // ...
30        }
31    }
```

#GP Exception

Hypervisor - Intercepts

- Intercepts obvious attack
- **Protected MSRs**
 - Other protected regs are forbidden
 - Most MSRs are protected

```
16889 MSR c00101f8 protected: READ & WRITE
16890 MSR c00101f9 protected: READ & WRITE
16891 MSR c00101fa protected: READ & WRITE
16892 MSR c00101fb protected: READ & WRITE
16893 MSR c00101fc protected: READ & WRITE
16894 MSR c00101fd protected: READ & WRITE
16895 MSR c00101fe protected: READ & WRITE
16896 MSR c00101ff protected: READ & WRITE
16897 MSR c0010200 is NOT protected
16898 MSR c0010201 is NOT protected
16899 MSR c0010202 is NOT protected
16900 MSR c0010203 is NOT protected
16901 MSR c0010204 is NOT protected
16902 MSR c0010205 is NOT protected
16903 MSR c0010206 is NOT protected
16904 MSR c0010207 is NOT protected
16905 MSR c0010208 is NOT protected
16906 MSR c0010209 is NOT protected
16907 MSR c001020a is NOT protected
16908 MSR c001020b is NOT protected
16909 MSR c001020c protected: READ & WRITE
16910 MSR c001020d protected: READ & WRITE
16911 MSR c001020e protected: READ & WRITE
16912 MSR c001020f protected: READ & WRITE
16913 MSR c0010210 protected: READ & WRITE
16914 MSR c0010211 protected: READ & WRITE
16915 MSR c0010212 protected: READ & WRITE
16916 MSR c0010213 protected: READ & WRITE
16917 MSR c0010214 protected: READ & WRITE
16918 MSR c0010215 protected: READ & WRITE
```

Hypervisor - Intercepts

- Intercepts obvious attack
- Protected MSRs
- **CR0 Write is filtered**

```
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;
    }
}
```

Hypervisor - Intercepts

- Intercepts obvious attack
- Protected MSRs
- **CR0 Write is filtered**

```
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;
    }
}
```

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

Hypervisor - Intercepts

- Intercepts obvious attack
- Protected MSRs
- CR0 Write is filtered
- **CR4 Write is also filtered**

```
case VMEXIT_CR4_WRITE:
    vmcb = (struct vmcb *) vcpu->vmcb_ctrl;
    uint32_t *reg = hv_get_reg(vcpu, vmcb->ctrl.exit_info_1);
    uint32_t changed_bits = vmcb->vmcb_save_state.CR4 ^ reg[0];

    if ((changed_bits & 0b1100000000000000000001) != 0) {
        vcpu->vmcb_ctrl->event_inj = 0x2BAD000080000B0D;
        return;
    }

    // ...
    vmcb_ctrl_1->vmcb_save_state.CR4 = reg[0];
    break;
```

Hypervisor - Intercepts

- Intercepts obvious attack
- Protected MSRs
- CR0 Write is filtered
- **CR4 Write is also filtered**

```
case VMEXIT_CR4_WRITE:  
    vmcb = (struct vmcb *) vcpu->vmcb_ctrl;  
    uint32_t *reg = hv_get_reg(vcpu, vmcb->ctrl.exit_info_1);  
    uint32_t changed_bits = vmcb->vmcb_save_state.CR4 ^ reg[0];  
  
    if ((changed_bits & 0b1100000000000000000001) != 0) {  
        vcpu->vmcb_ctrl->event_inj = 0x2BAD000080000B0D;  
        return;  
    }  
}
```

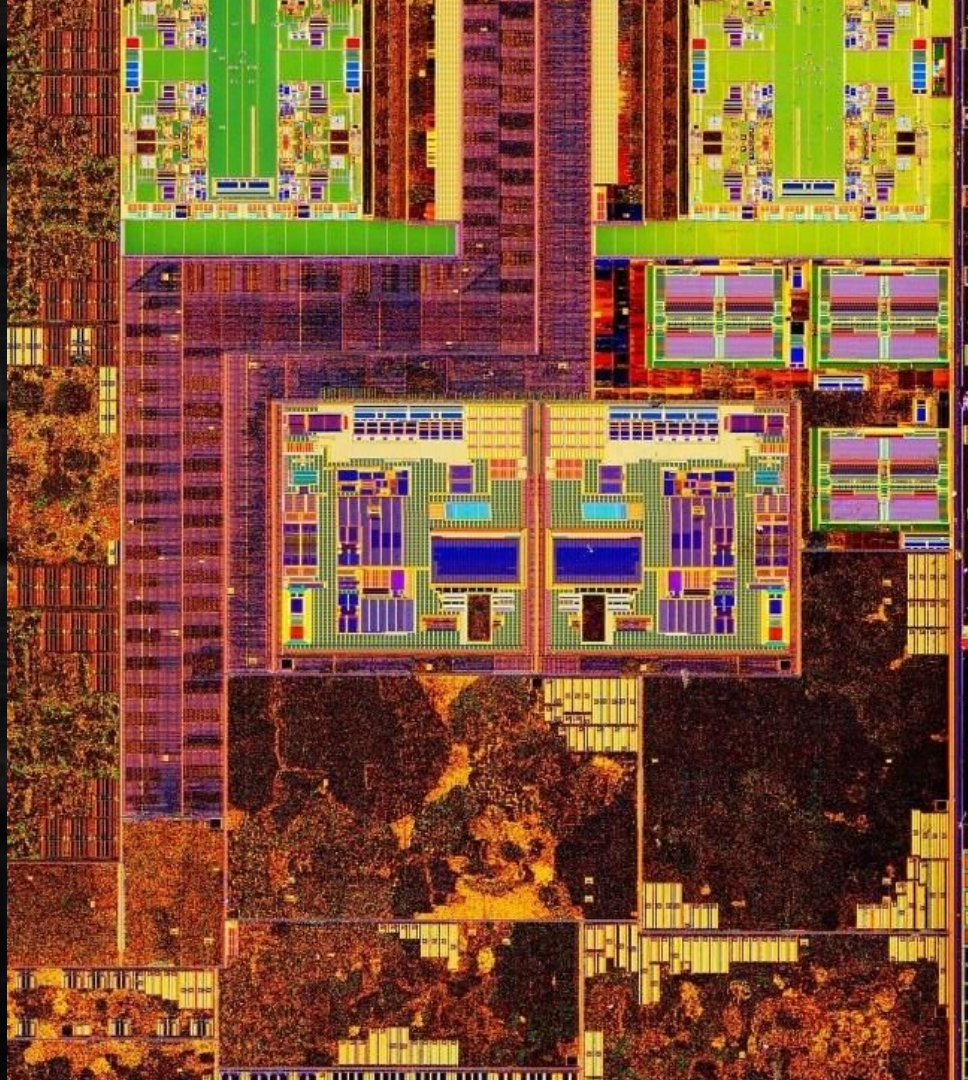
Bits	Mnemonic	Description	Access Type
63:24	Reserved		MBZ
23	CET	Control-flow Enforcement Technology	R/W
22	PKE	Protection Key Enable	R/W
21	SMAP	Supervisor Mode Access Protection	R/W
20	SMEP	Supervisor Mode Execution Prevention	R/W
19	Reserved		MBZ
18	OSXSAVE	XSAVE and Processor Extended States Enable Bit	R/W
17	PCIDE	Process Context Identifier Enable	R/W
16	FSGSBASE	Enable RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE instructions	R/W
15:13	Reserved		MBZ
12	LA57	5-Level Paging Enable	R/W
11	UMIP	User Mode Instruction Prevention	R/W
10	OSXMMEXCPT	Operating System Unmasked Exception Support	R/W
9	OSFXSR	Operating System FXSAVE/FXRSTOR Support	R/W
8	PCE	Performance-Monitoring Counter Enable	R/W
7	PGE	Page-Global Enable	R/W
6	MCE	Machine Check Enable	R/W
5	PAE	Physical-Address Extension	R/W
4	PSE	Page Size Extensions	R/W
3	DE	Debugging Extensions	R/W
2	TSD	Time Stamp Disable	R/W
1	PVI	Protected-Mode Virtual Interrupts	R/W
0	VME	Virtual-8086 Mode Extensions	R/W

Hypervisor (Security)

- Intercepts various other hypercalls
- Most are IOMMU related
 - SET_GUEST_BUFFERS
 - ENABLE_DEVICE
 - (UN)BIND_PASID
 - Etc.
- Other misc. hypercalls
 - TMR_VIOLATION_ERROR for Trusted Memory Region
 - SET_CPUID_(PS4/PPR) for PS4 game emulation

Encryption & Signing

Brief Overview



Encryption & Signing

- PS5 has code signing
- Applications & libraries are encrypted on disk
- Most of these files are known as “Signed ELF’s” (SELF’s)
 - Only the security co-processor has keys
 - Decrypted & Verified via kernel API
 - Much more relevant to the PS5 security model than PS4
- Used to be handled by SAMU
- Superseded by AMD Platform Security Processor (PSP/AMD-SP)

Encryption & Signing - PSP + DRM

- PS5 runs two secure kernels

Encryption & Signing - PSP + DRM

- PS5 runs two secure kernels
- **Sony Secure OS (MP0)**
 - Secure modules, SAMU port

Secure Modules

Filename	Service
80021000	authmgr
80021001	kms
80021002	pup
80021003	pfs
80021004	driveauth

```
int64_t psp_send_mbox(int32_t arg1, int64_t arg2, int64_t arg3)
{
    int64_t rax = stack_cookie
    int64_t rax_2
    if (arg1 > 0xa || (arg1 <= 0xa && not(test_bit(0x642, arg1))))
        sub_ffffffffff93d8450("%s: invalid tmid %d\n")
        rax_2 = 0x16
    if (arg1 <= 0xa && test_bit(0x642, arg1))
        char const* const rdi
        if (arg2.w != 0)
            rdi = "%s: unaligned addr 0x%lX\n"
        if (arg2.w == 0 && arg3.w != 0)
            rdi = "%s: should be multiple of 64KiB 0x%lX\n"
        if (arg2.w != 0 || (arg2.w == 0 && arg3.w != 0))
            sub_ffffffffff93d8450(rdi)
            rax_2 = 0x16
        if (arg2.w == 0 && arg3.w == 0)
            void var_78
            bzero(&var_78, 0x40)
            int32_t var_60 = 0x20
            int32_t var_58_1 = arg1
            int32_t var_54_1 = 0
            int64_t var_50_1 = arg2
            int64_t var_48_1 = arg3
            void* r12_2 = 0xffff800000000000 | zx.q(data_ffffffffffddc23b54) << 0x1e | zx.q(data_
            if ((*r12_2 - 0xfdc10570) & 0x40000000) != 0)
                sub_ffffffffff93d8450("Recovery Flag detected, ignore the command(Cmd 0x%X)\n")
                label_ffffffffff9228d0d:
                sub_ffffffffff93d8450("%s: Command error\n")
                label_ffffffffff9228d12:
                rax_2 = 5
            else
                rax_2 = 0
    return rax_2
}
```

Encryption & Signing - PSP + DRM

- PS5 runs two secure kernels
- Sony Secure OS (MP0)
 - Secure modules, SAMU port
- **AMD Secure OS (MP3)**
 - **PlayReady 3000 DRM**
 - **Trusted Execution Environment**
 - **Supports hotloading of Trusted Applets (TAs)**

```
fffffffdae08313 mp3_system_suspend_handler
fffffffdae0de1a W:\\Build\\J01183702\\sys\\internal\\modules\\mp3\\mp3-psp-tee.c
fffffffdae19132 mp3_enable -> false\n
fffffffdae214c0 MP3 PLAYER
fffffffdae32b5f MP3 Player*
fffffffdae337c5 mp3_ioctl_get_debug_token
fffffffdae4d0d1 mp3_scanin_unblock2_handler
fffffffdae68570 mp3_process_suspend_handler
fffffffdae70ced mp3_process_exit_handler
fffffffdae7a40f mp3_tmr_enc_handler
fffffffdae7a423 failed to signal mp3 to suspend\n
fffffffdae95a9e mp3_scanin_unblock_handler
fffffffdae97b6c MP3 Decoder
fffffffdaea7756 mp3_tmr_free_handler
```

IOCTL #	Name	Notes
0xC010B403	TEE_IOCTL_INVOKE	Invoke commands
0xC010B408	TEE_IOCTL_DLM_GET_DEBUG_TOKEN	-
0xC028B409	TEE_IOCTL_DLM_START_TA_DEBUG	-
0xC110B40A	TEE_IOCTL_DLM_FETCH_DEBUG_STRING	-
0x8008B40B	TEE_IOCTL_DLM_STOP_TA_DEBUG	-
0x8008B40B	TEE_IOCTL_INIT_ASD	-
0xC038B40C	TEE_SHM_REGISTER	-
0xC004B40E	TEE_SHM_RELEASE	-
0xC004B40F	TEE_SET_TIMEOUT	-

Encryption & Signing - “A53”

- Additional security co-processor built into the PS5 SoC
 - Dual core AARCH64 co-processor
- Kernel refers to it as “A53” and/or “mp4”
 - Likely a Cortex-A53
- Don’t know much about it yet
- Seems unique to PS5/Xbox SX
- Moves secure stuff from x86

```
fffffffdae04729 mode: A53 IO Controller enabled\n
fffffffdae04a51 a53mm_context.restore_priv_state == 0
fffffffdae04a77 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_privilege_mode.c:552
fffffffdae04ac6 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_privilege_mode.c:1452
fffffffdae04b16 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_privilege_mode.c:1861
fffffffdae0600d a53mm_context.proc_suspend_count + sc->user_suspend_count + sc->system_suspend_count == a
fffffffdae06089 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_util.c:549
fffffffdae060ce W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_util.c:65
fffffffdae06417 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_debug.c:270
fffffffdae06668 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_command.c(%d) a53mm is
fffffffdae067d5 a53mm_context.icr_taskqueue != NULL
fffffffdae06809 [A53MM] %s: a53mm app rings are not suspended\n
fffffffdae06838 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm.c:725
fffffffdae06878 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm.c:955
fffffffdae08079 a53io_scf
fffffffdae089a7 W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_page_table.c:564
fffffffdae097dc W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_input_ring.c(%d) a53mm
fffffffdae0983b W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_input_ring.c:308
fffffffdae09bec W:\\Build\\J01183702\\sys\\mp4\\x86\\kernel\\modules\\a53mm\\a53mm_event.c(%d) AMPR EVENT
fffffffdae0bf06 a53ioc_set_key2
fffffffdae0cd48 A53IO oring_lock
```

a53mm

IOCTL #	Name	Notes
0xC030AC03	A53MM_CALL_INDIRECT_BUFFER	-
0xC038AC06	A53MM_GET_PARAM *	-
0xC018AC10	A53MM_GET_USAGE_STATS_DATA	-
0xC030AC02	A53MM_GIVE_DIRECT_MEM_TO_MAPPER	-
0xC018AC0E	A53MM_INTERNAL_LOCK_UNLOCK_MAPPER_MEMORY	-
0xC018AC12	A53MM_INTERNAL_TEST_PAGE_MIGRATION	-
0xC018AC0B	A53MM_MAPPER_QUERY_PA	-

Future Research & Ideas

Recap and where to go from
here



Future Research & Ideas - Data-Only Attacks

- Hypervisor essentially limits us to data-only attacks
- But control of data is still powerful
 - We can't patch/hook code
 - ... but we can hook data
- Instead of patching PSP kernel API, we can try hijacking the mailbox
 - Spoof responses to load our own code
 - Haven't had time to try this yet, but in theory should work
- Might not be path of least resistance...

Future Research & Ideas - Hypervisor

- Guest has a lot of potential vectors for VM escape
- Obvious and easy ones are out
 - Control regs, EFER, page tables
- But less obvious vectors can be explored
 - MSRs
 - IOMMU / HW attack + IOMMU hypercalls
 - Features / extended instructions HV doesn't consider
 - Memory Mapped I/O (MMIO)
- These kinds of bugs definitely exist :)

Future Research & Ideas - Hypervisor

- The hypervisor can't protect *everything*
- Trade-offs
 - Move more to HV = more attack surface
 - Also high performance penalty
- HV is completely in-house
 - Less audited
 - Less mature
 - But a unique albeit formidable challenge is XOM

Conclusions

Conclusions

- Sony has stepped up
 - Attack surface reductions
 - Exploit mitigations
 - Hypervisor-backed security model
- System mods no longer as easy as Userland + Kernel chain
- Secure element and hardware security actually matters now
- Most of this security is banked on the hypervisor

Conclusions

- While the hypervisor is small, securing it is still hard
- Complex implementations force an ever-expanding scope
- Console exploitation isn't dead, just more interesting
- But it's more costly in terms of time
- More steps are needed
 - More burn potential

Conclusions

- If this all sounds interesting to you, I run a discord server for PS5 research
 - Still lots to do!
 - Feel free to hit me up on discord or twitter with thoughts or questions
 - PS5 R&D discord: discord.gg/kbrzGuH3F6
 - My handles:
 - Discord: [specter#0666](#)
 - Twitter: [@SpecterDev](#)
 - If you do work on stuff, document on psdevwiki.com
- Hopefully some cool stuff goes public soon

Thanks

- Flatz: providing a kernel dump to reverse + other knowledge
- Daax: answering all my annoying HV questions
- Misc. others from Reverse Engineering discord
 - discord.gg/rtfm
- zi: nitpicking slides
- Hardwear.io for giving me the opportunity to speak
- All of you for listening

Fin