

安全开发者峰会 ZeroMDASH

领域自适应的Agent-Native通用漏洞挖掘工程

C0ss4ck, Swing



目录

从研究方法证据到可恢复的攻击链调查

- 01 工程启发与系统总览**
MDASH、Mantis 与 ZeroMDASH 整体架构
P 03-08
- 02 Context Preparation: 领域上下文准备**
研究员经验、五类研究线索与 Harness × Model 横评
P 09-13
- 03 Threat Modeling: 威胁建模与任务规划**
领域自适应建模、MySQL 示例与扫描任务包
P 14-17
- 04 Investigation: 漏洞调查与攻击链复核**
假设调查、独立复核、原语组合与报告归组
P 18-20
- 05 漏洞案例展示**
KVM 逃逸、Milvus 堆溢出
P 21-38

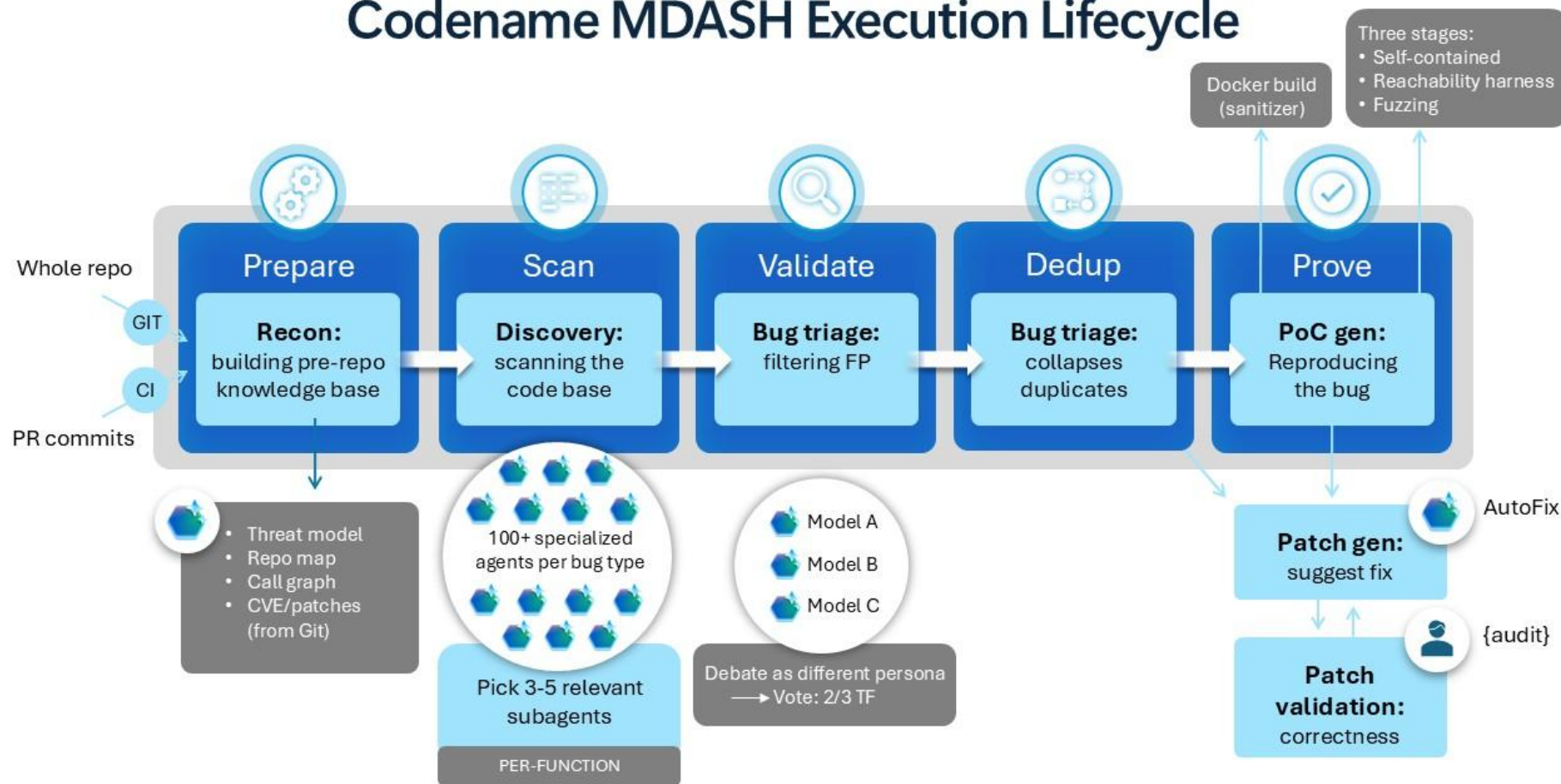
工程启发 与 系统总览





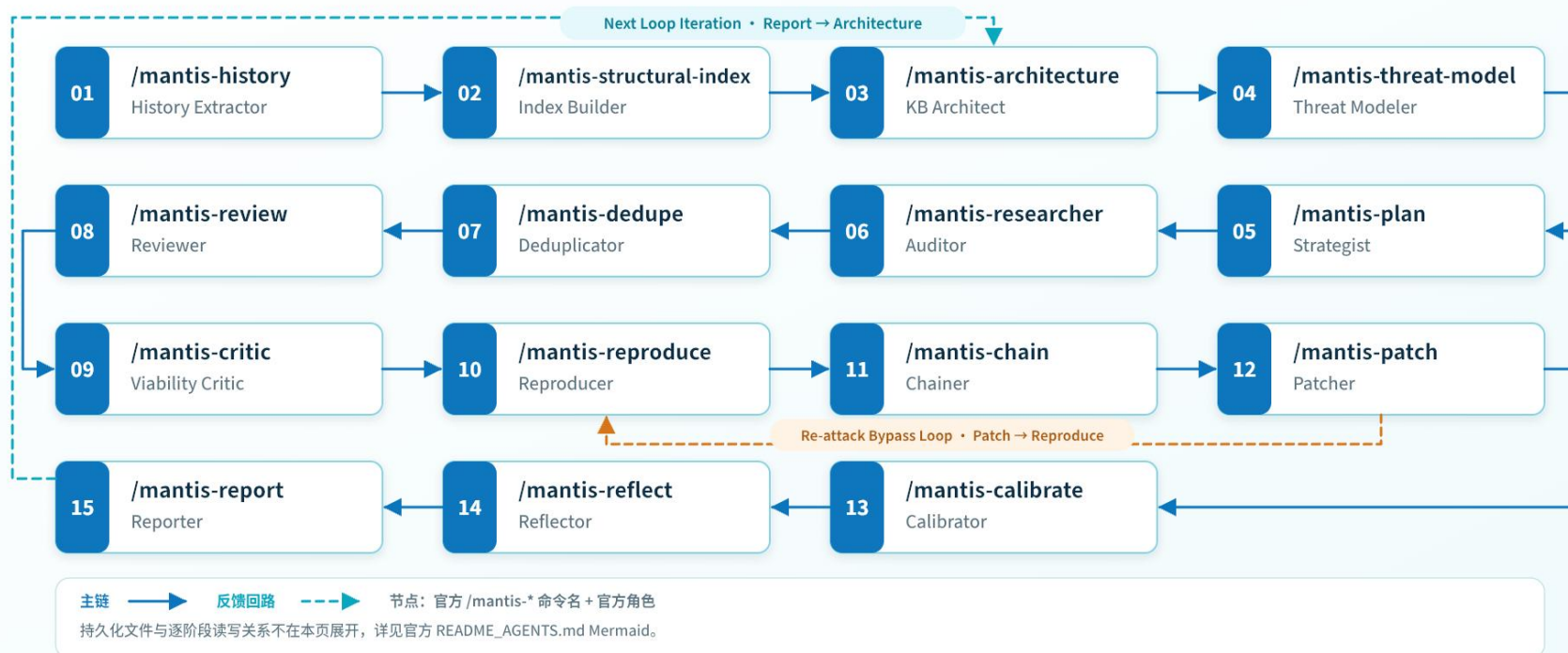
Agentic漏洞挖掘工程: MDASH

Codename MDASH Execution Lifecycle



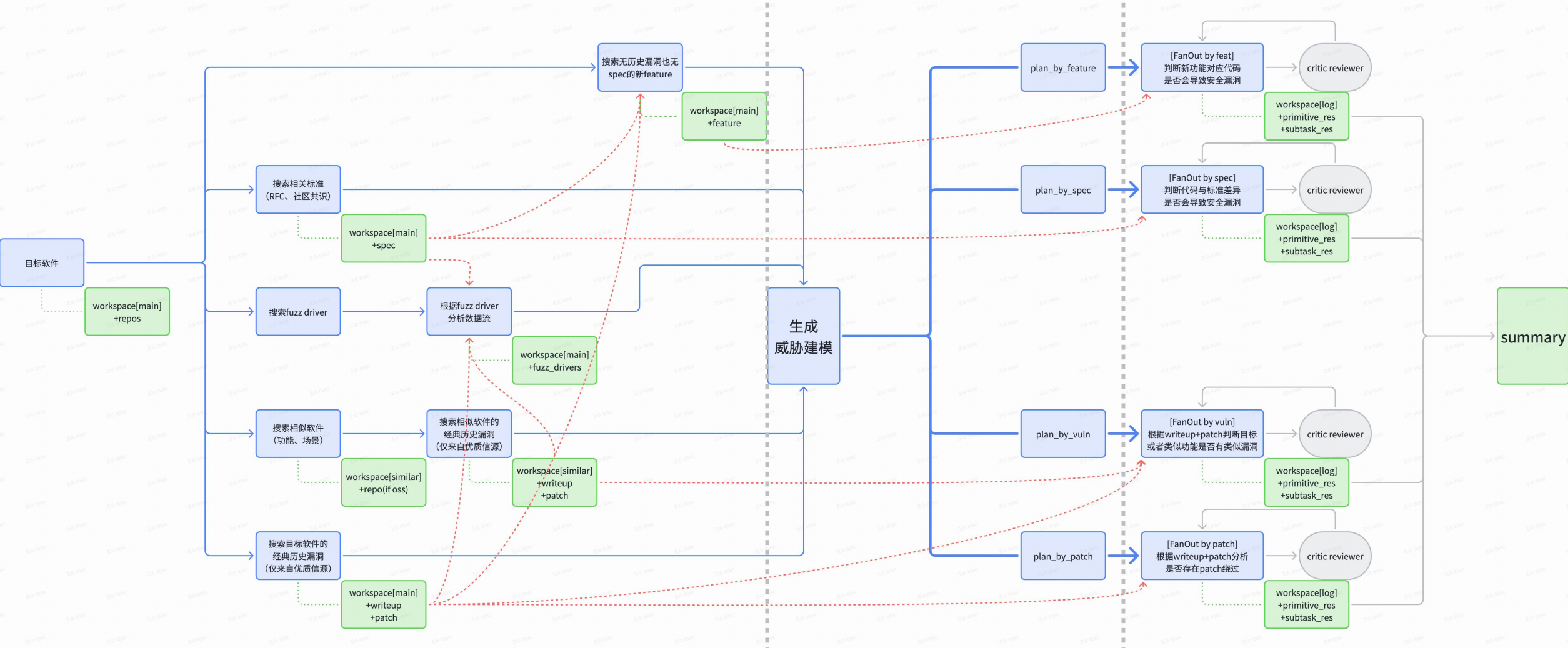
Agentic漏洞挖掘工程：Mantis

Mantis Workflow



来源: github.com/google/mantis • README_AGENTS.md • commit 2b3bbdc (2026-10-10 获取)

ZeroMDASH 主图



Context Preparation

领域上下文准备

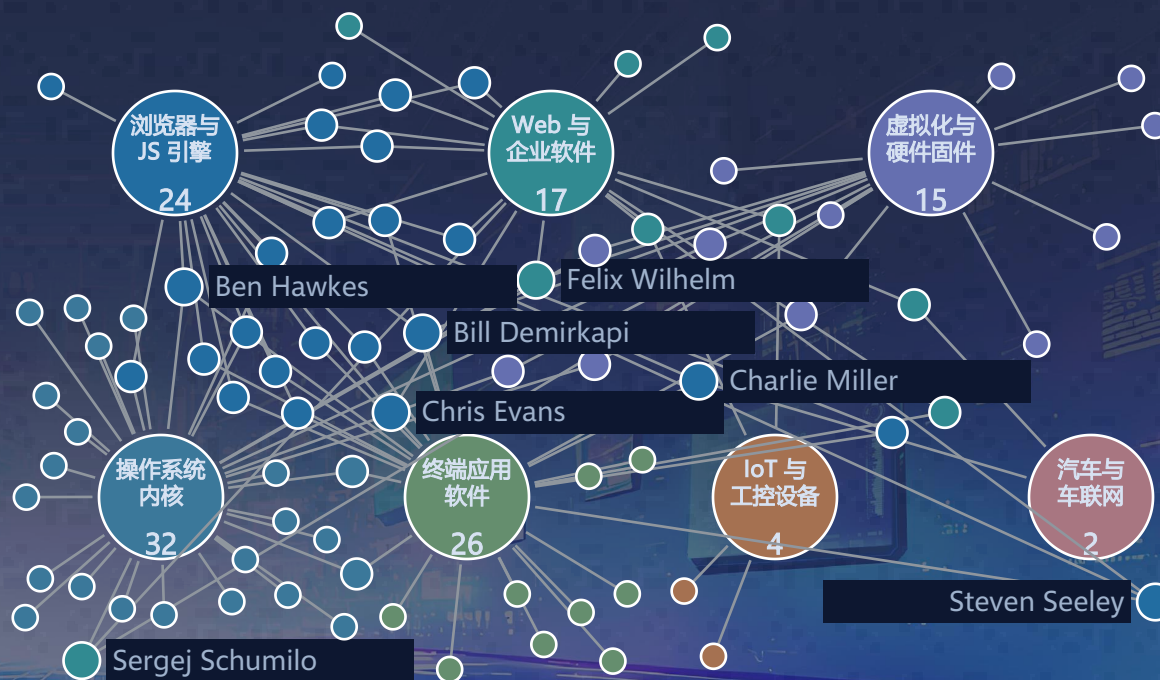
Context Preparation: 安全研究员是怎么挖洞的?

研究与复核流程



研究员与领域归属网络

大节点为领域，小节点为研究员



Context Preparation: 研究员样本中的五类常见研究方向

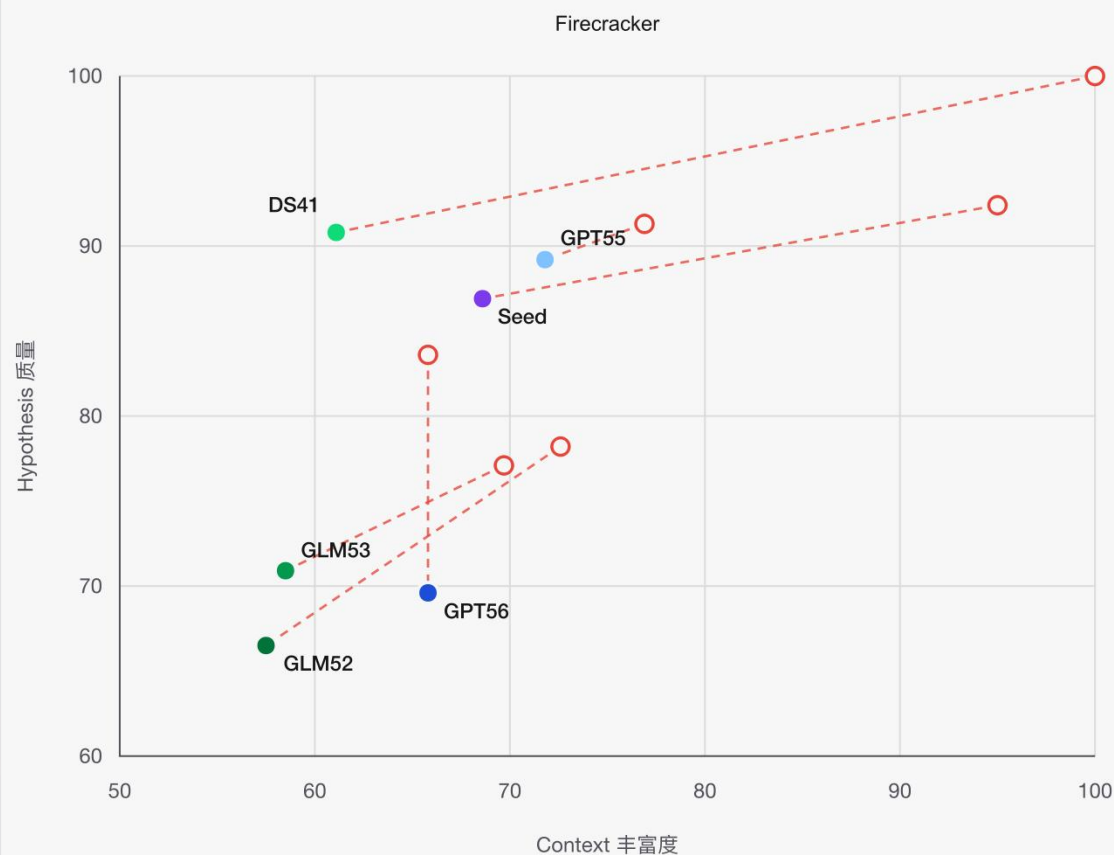
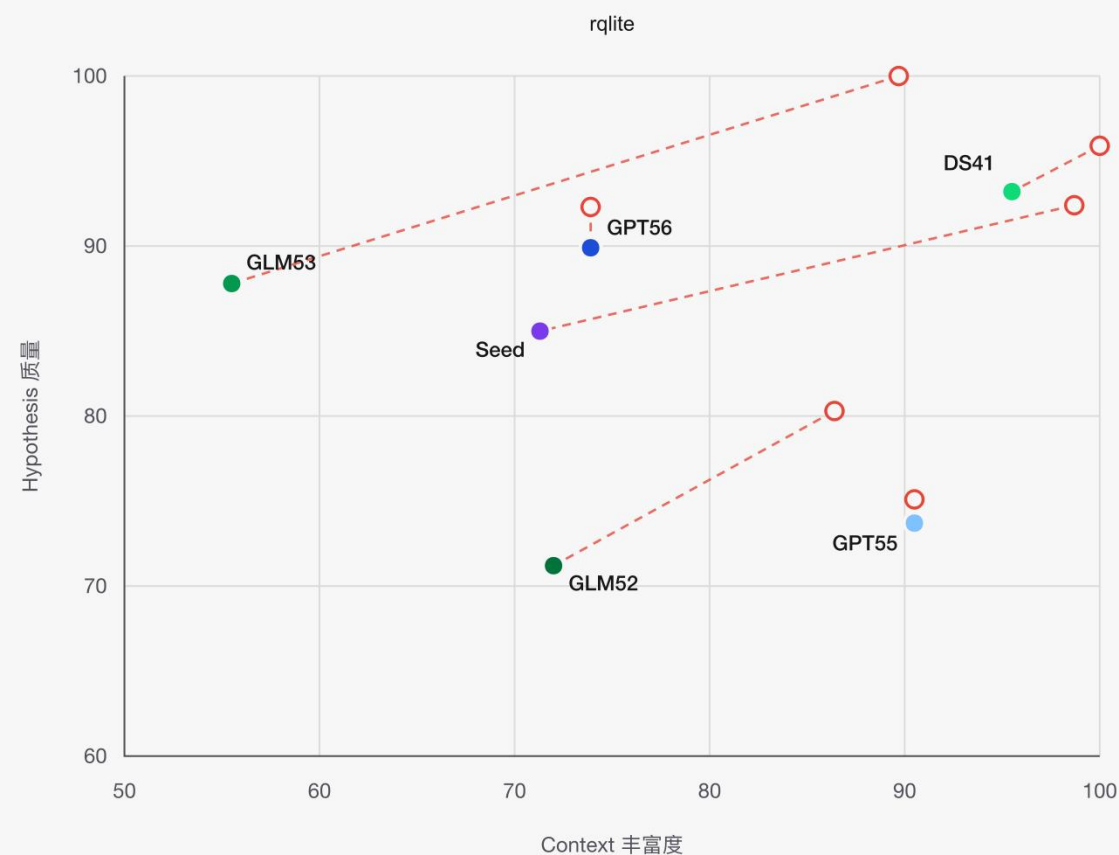
	目标软件 历史漏洞 合计 23 / 78	相似软件 历史漏洞 合计 12 / 78	Fuzz Driver 开发 合计 18 / 78	相关功能 标准 Diff 合计 4 / 78	新 Feature 安全研究 合计 7 / 78
浏览器与 JS 引擎 n = 24	38% 9 / 24	29% 7 / 24	21% 5 / 24	13% 3 / 24	13% 3 / 24
Web 与企业软件 n = 17	41% 7 / 17	35% 6 / 17	29% 5 / 17	18% 3 / 17	18% 3 / 17
虚拟化与服务器硬件/固件 n = 15	20% 3 / 15	20% 3 / 15	27% 4 / 15	0% 0 / 15	13% 2 / 15
操作系统内核 n = 32	34% 11 / 32	6% 2 / 32	25% 8 / 32	0% 0 / 32	13% 4 / 32
终端应用软件 n = 26	31% 8 / 26	19% 5 / 26	15% 4 / 26	4% 1 / 26	8% 2 / 26
IoT 与工控设备 n = 4	25% 1 / 4	25% 1 / 4	25% 1 / 4	0% 0 / 4	0% 0 / 4
汽车与车联网 n = 2	0% 0 / 2	0% 0 / 2	50% 1 / 2	0% 0 / 2	0% 0 / 2

Context Preparation: 为什么是这五类方向作 Context ?

方向类型	代表案例	核心查证问题	在 ZeroMDASH 中的下游任务
目标软件历史漏洞	regreSSHion (CVE-2024-6387) : 2006 修复, 2020 重构移除保护, 2024 回归	哪次重构删除或绕过了哪条安全不变量?	search_target_history → plan_by_vuln_patch; patch-manifest 含 repo 内验证的 commit SHA, 生成 missing / bypass 复核子任务
相似软件历史漏洞	0.0.0.0 Day: 三家浏览器初始均未识别 0.0.0.0 为本地目标	同一信任边界在目标软件实现里由哪段代码把守?	search_similar_software → plan_by_vuln_similar; 同类实现对照清单。迁移的是安全问题, 不是漏洞结论
规范与实现 Diff	MadeYouReset (CVE-2025-8671) : HTTP/2 层已关闭, 后端任务继续消耗资源	规范承诺的安全语义, 其实现在每一条代码路径上都一致满足吗?	search_spec → plan_by_spec: 把规范条款逐条转成核查任务, 对照实现逐项验证
Fuzz Driver 开发	OSS-Fuzz-Gen × OpenSSL (CVE-2024-9143) : 发现成立, 但是产品路径不可达 (Low)	harness 能生成的输入, 与产品真实入口可达的输入, 差距有多大?	search_fuzz_driver; 靠 fuzz driver 给 analyze_attack_surface 节点指引已明确可达的攻击面
新功能安全研究	MSG_OOB × CVE-2025-38236: 默认开启的新 flag + 沙箱放行, renderer 到内核提权	新功能默认开启或放行了什么能力, 打破了哪些既有安全假设?	search_new_feature → plan_by_feature: 缺少规范约束、fuzz driver覆盖、历史漏洞研究的新功能或成新攻击面

五类产物汇聚为统一任务上下文 (证据来源 / 代码锚点 / 攻击者前提 / 安全不变量 / 验证任务 / 优先级信号), 进入 Threat Modeling 与 Investigation 阶段

Context Preparation: 不同 Harness + Model 下的横评



harness+model

GLM52 · Claude Code + GLM-5.2

GLM53 · Claude Code + GLM-5.3-Flash

GPT55 · Codex + GPT-5.5

GPT56 · Codex + GPT-5.6-sol

DS41 · dsh + DeepSeek-V4.1-Flash

Seed · TraeX + Seed-Evolving

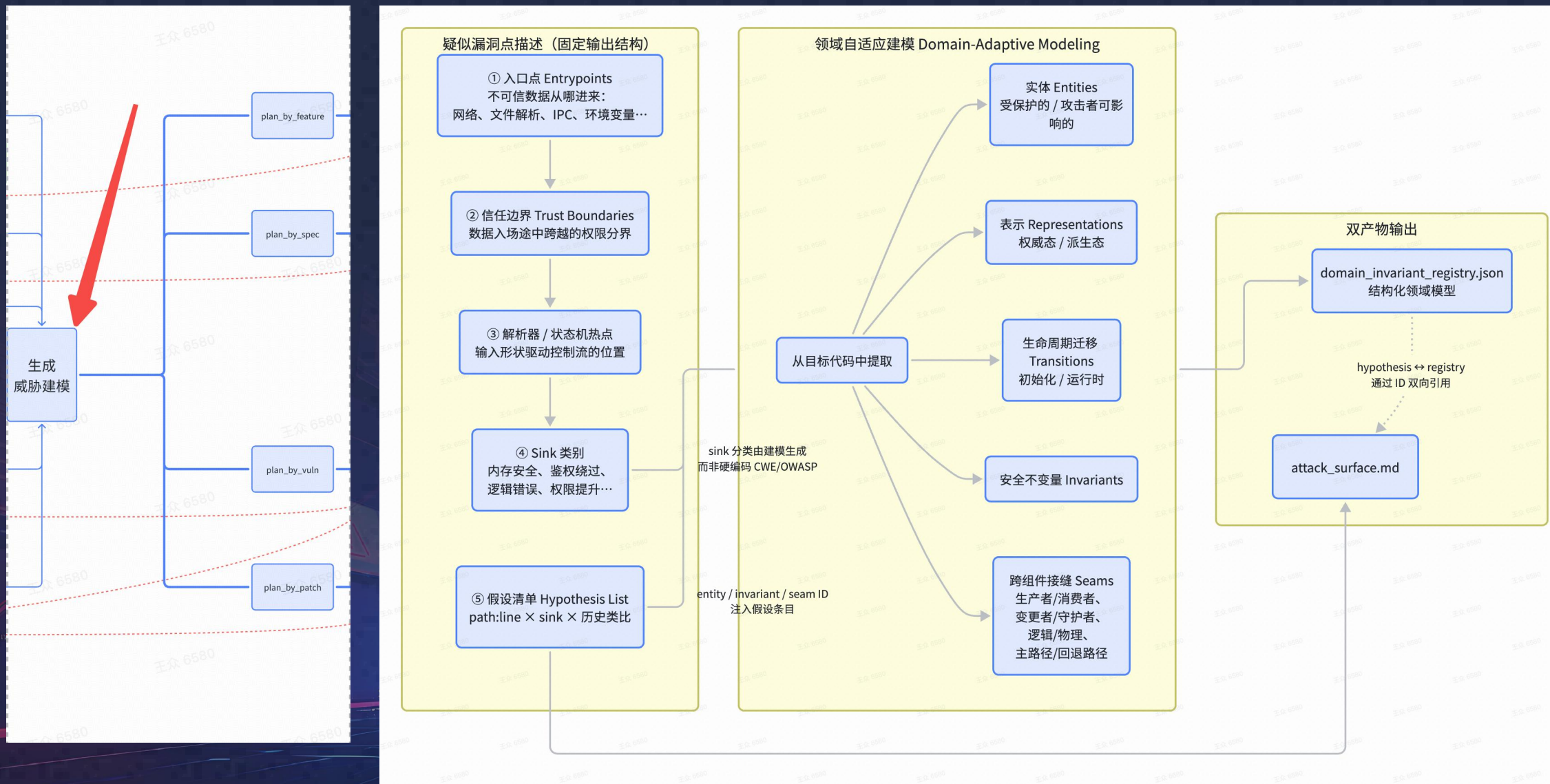
编码 ○ 幻觉审计前得分

- - -> 幻觉扣分方向

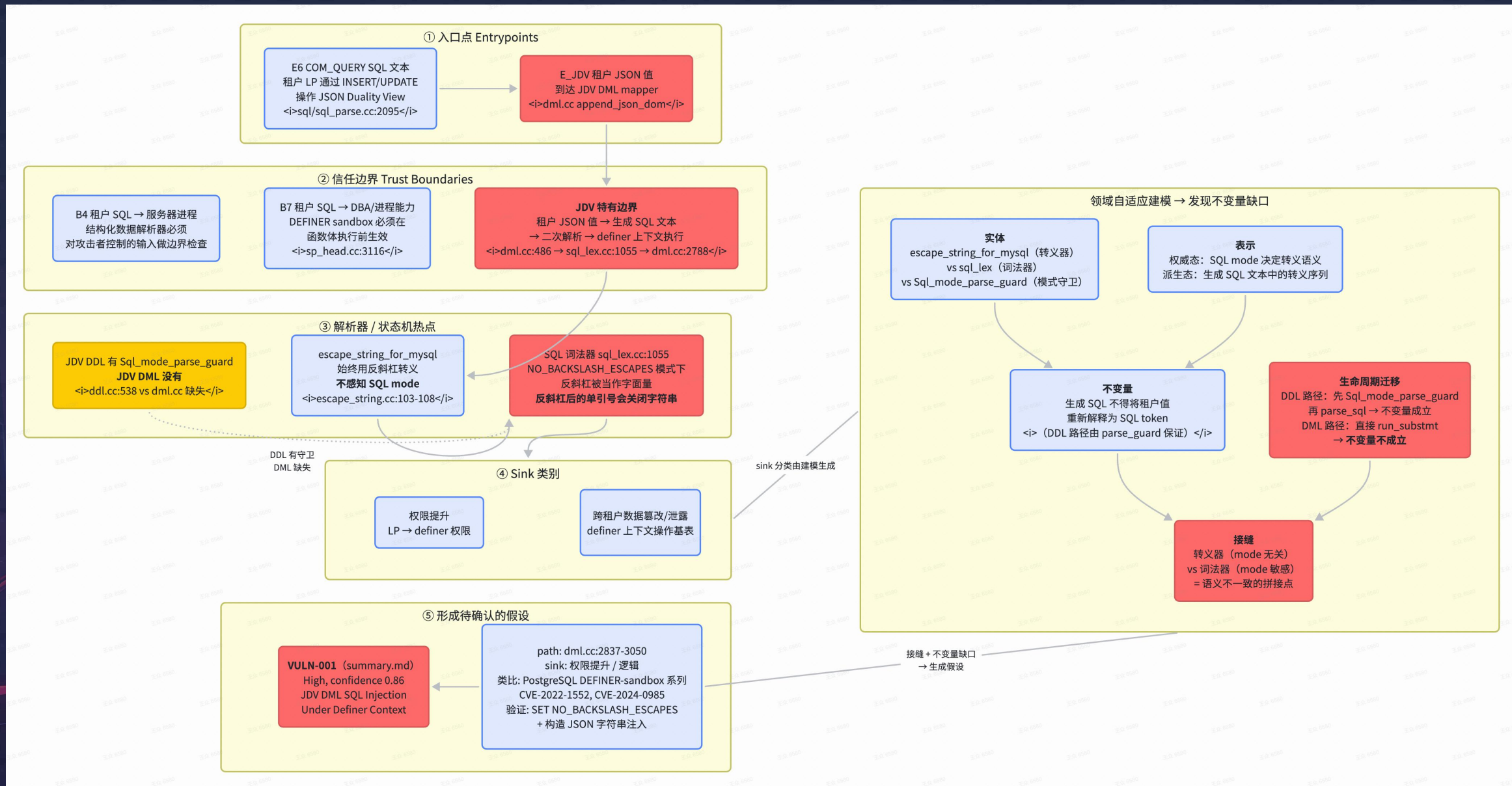
Threat Modeling

威胁建模 与 任务规划

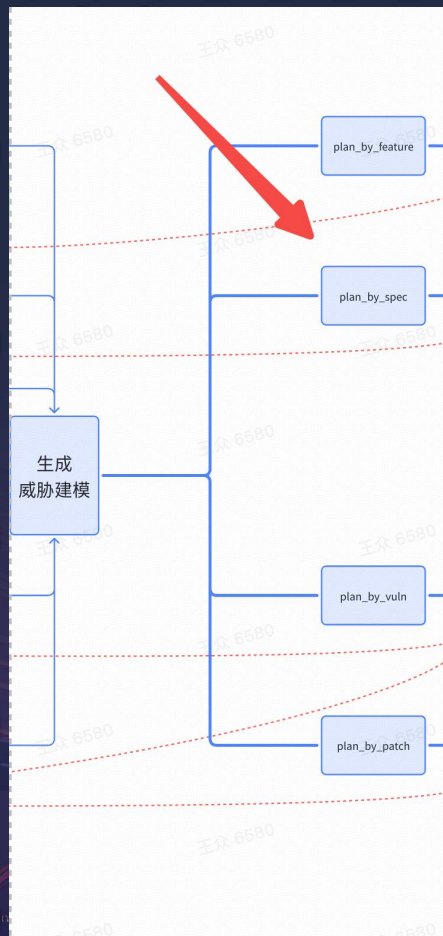
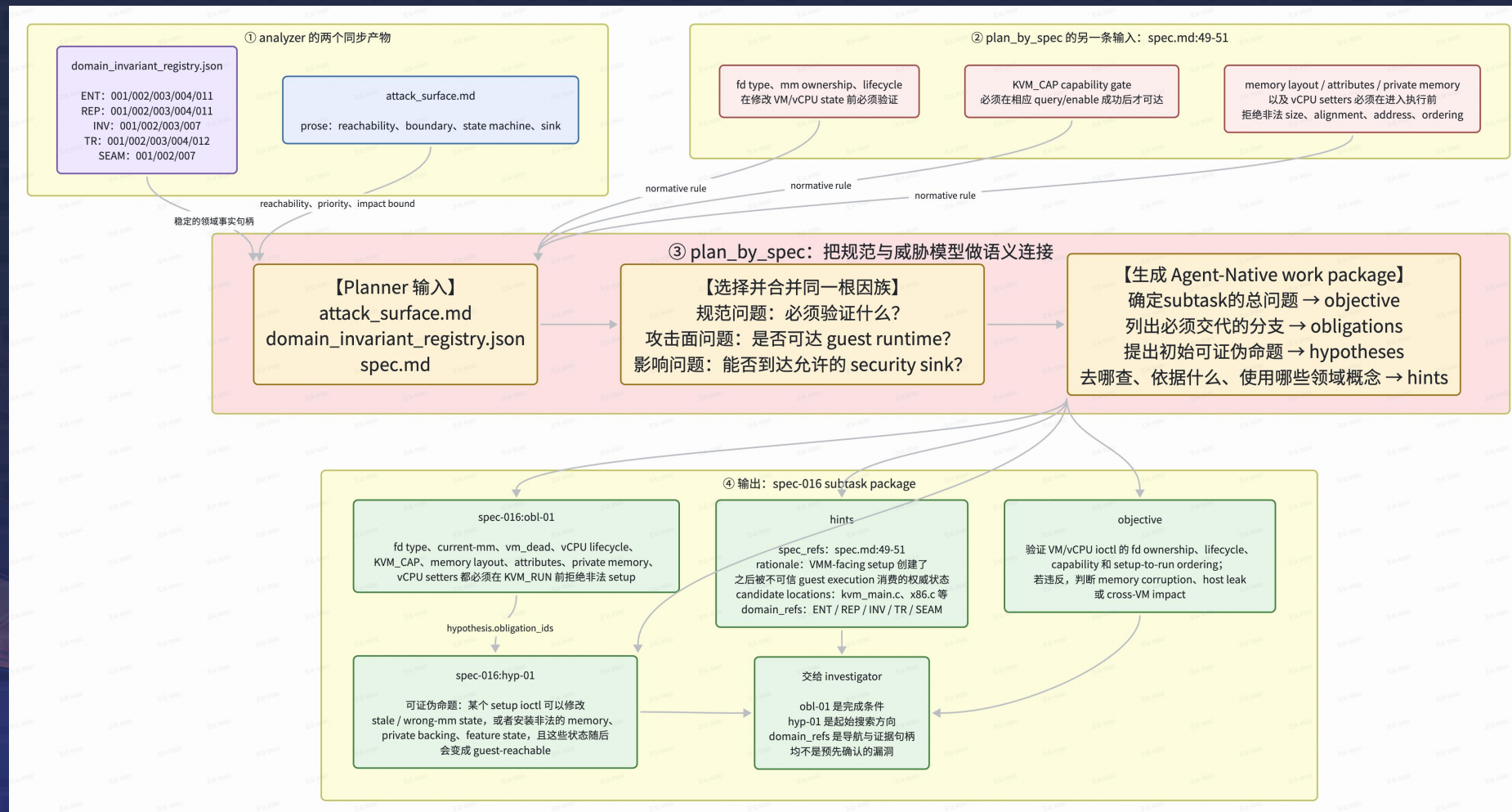
Threat Modeling: 统一描述威胁建模与疑似漏洞点



Threat Modeling: 以MySQL某漏洞为例



Threat Modeling: 将威胁建模转化为漏洞扫描任务包

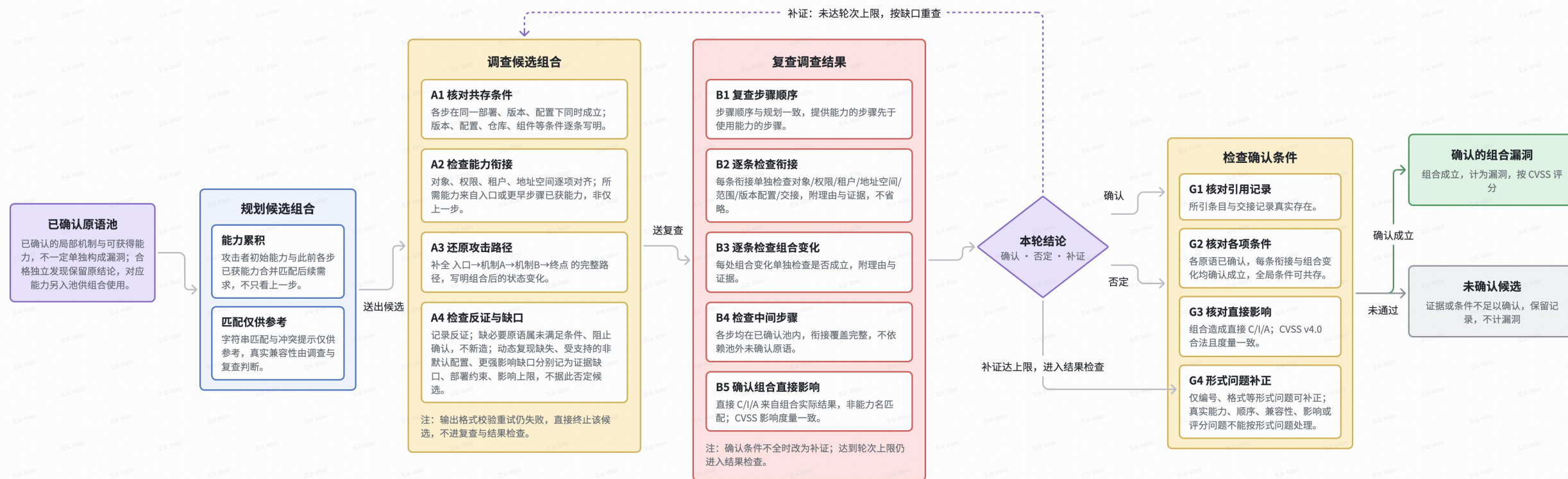




Investigation 漏洞调查 与 攻击链复核

Chain Investigation Subtask: 组合漏洞与漏洞原语

组合漏洞识别流程



0day/1day 案例介绍

案例：KVM CVE-2026-64555, SPSR 污染导致 eret 到 EL2

KVM (arm64) 嵌套虚拟化下, guest vm内能触发的特权级越权导致的虚拟机逃逸



VHE 是 ARM64 的 Virtualization Host Extensions (虚拟化宿主扩展)

它让运行 KVM 的宿主 Linux 内核可以直接运行在物理 EL2, 同时尽量保持原来运行在 EL1 时的编程模型。

Memory copy/set instructions (MOPS)

Armv8.8 的 FEAT_MOPS 为 memcpy/memset 提供了一组指令。一次操作通常由三个阶段组成:

- CPYP / SETP prologue, 初始化
- CPYM / SETM main, 执行主要复制/填充
- CPYE / SETE epilogue, 收尾

主指令或尾声指令可能因各种原因而发生 MOPS 异常, 例如, 当任务迁移到具有不同 MOPS 实现的 CPU 时, 或者当指令的对齐和大小要求未得到满足时。然后, 软件异常处理程序将重置寄存器并从序言指令重新开始执行。

异常处理程序需要:

把源地址、目标地址和剩余长度恢复到 prologue 能接受的格式;

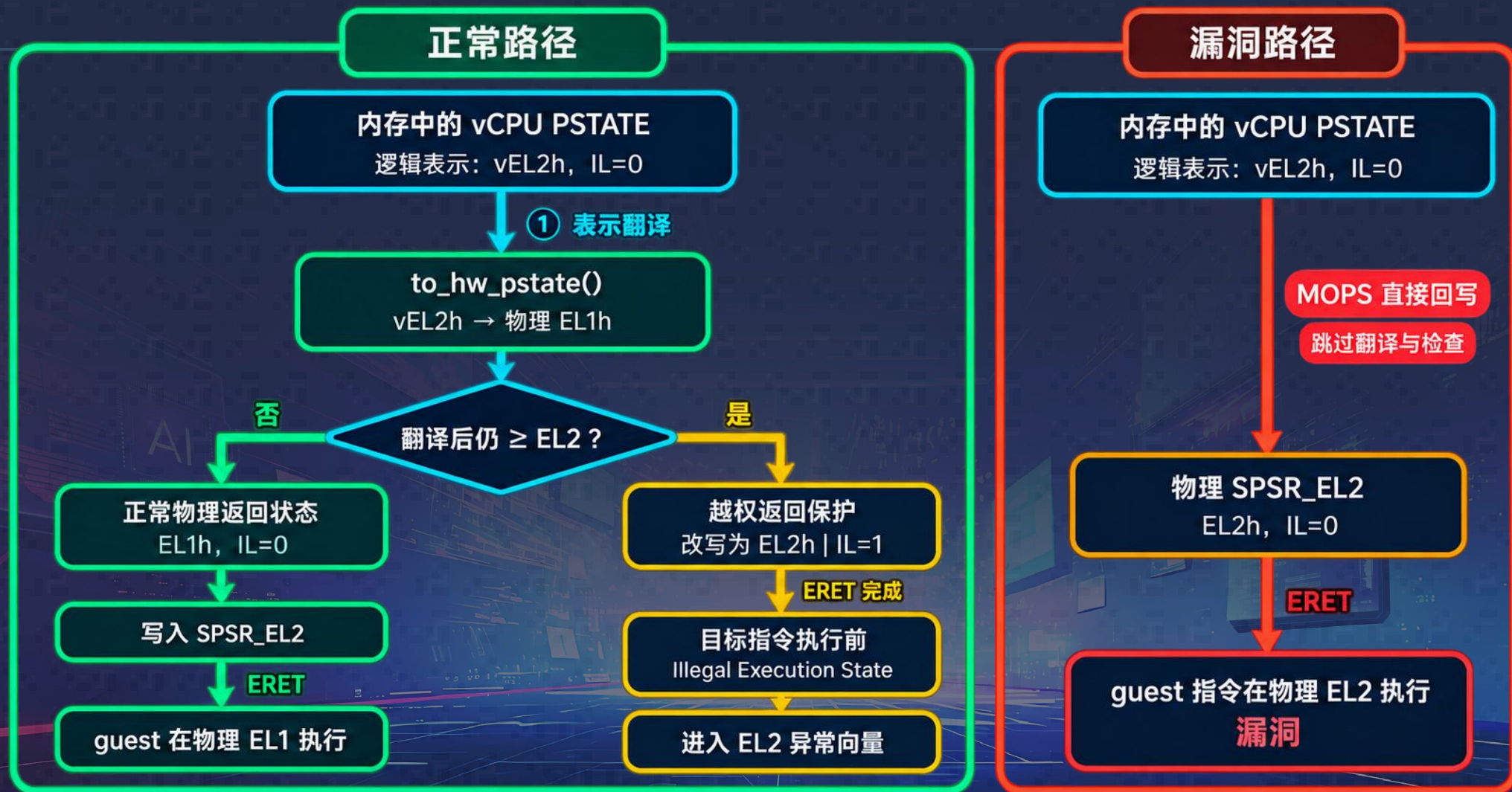
把 PC 回退到 prologue;

从 prologue 重新执行整组 MOPS 指令。

这是 Arm 架构要求的恢复方式。KVM 必须在 hypervisor 内处理此异常

关键机制：一份状态、两种表示

内存里的 pstate 副本 与 硬件寄存器 SPSR_EL2；正常路径必须经过翻译与检查



关键机制：一份状态、两种表示

内存里的 pstate 副本 与 硬件寄存器 SPSR_EL2；正常路径必须经过翻译与检查

```
u64 mode = ctxt->regs.pstate & (PSR_MODE_MASK | PSR_MODE32_BIT);

switch (mode) {
case PSR_MODE_EL2t:
    mode = PSR_MODE_EL1t;
    break;
case PSR_MODE_EL2h:
    mode = PSR_MODE_EL1h;
    break;
}

return (ctxt->regs.pstate & ~(PSR_MODE_MASK | PSR_MODE32_BIT)) | mode;
```

漏洞直接原因：vEL2 未经翻译并被写进 SPSR_EL2

同一趟退出处理里先『改回虚拟表示』，随后 MOPS 处理程序把它原样写回硬件寄存器



code: vhe/switch.c:548-561 -> switch.h:459-460 -> entry.S:88 (ERET)

案例: kvm CVE-2026-64555, SPSR 污染导致 eret 到 EL2

KVM (arm64) 嵌套虚拟化下, guest vm内能触发的特权级越权导致的虚拟机逃逸

```

v2:
- Reword the changelog (Oliver Upton).
- Modify the hardware SPSR_EL2 directly instead of translating the
  synthetic vcpu_cpsr(), per review (Oliver Upton).

arch/arm64/kvm/hyp/include/hyp/switch.h | 9 ++++++---
1 file changed, 6 insertions(+), 3 deletions(-)

diff --git a/arch/arm64/kvm/hyp/include/hyp/switch.h b/arch/arm64/kvm/hyp/include/hyp/switch.h
index e9b36a3b27bbc..0995e34aa3c54 100644
--- a/arch/arm64/kvm/hyp/include/hyp/switch.h
+++ b/arch/arm64/kvm/hyp/include/hyp/switch.h
@@ -448,16 +448,19 @@ static inline bool __populate_fault_info(struct kvm_vcpu *vcpu)

static inline bool kvm_hyp_handle_mops(struct kvm_vcpu *vcpu, u64 *exit_code)
{
+   u64 spsr;
+
   *vcpu_pc(vcpu) = read_sysreg_el2(SYS_ELR);
   arm64_mops_reset_regs(vcpu_gp_regs(vcpu), vcpu->arch.fault.esr_el2);
   write_sysreg_el2(*vcpu_pc(vcpu), SYS_ELR);

   /*
    * Finish potential single step before executing the prologue
    * instruction.
    * instruction. Modify the hardware SPSR_EL2 directly, as vcpu_cpsr()
    * may hold a synthetic (vEL2) value for a guest hypervisor.
    */
-   *vcpu_cpsr(vcpu) &= ~DBG_SPSR_SS;
-   write_sysreg_el2(*vcpu_cpsr(vcpu), SYS_SPSR);
+   spsr = read_sysreg_el2(SYS_SPSR);
+   write_sysreg_el2(spsr & ~DBG_SPSR_SS, SYS_SPSR);

   return true;
}
__ https://lore.kernel.org/all/20260617040820.2194831-2-bestswngs@gmail.com/
2.43.0

```

硬件: flags | SS | EL1h

vcpu_cpsr:
flags | EL2h ← *vcpu_cpsr(vcpu) &= ~DBG_SPSR_SS

硬件 SPSR_EL2:
SYS_SPSR) ← write_sysreg_el2(*vcpu_cpsr(vcpu), SYS_SPSR)

flags | EL2h ← 错误, 本应保持 EL1h

读取硬件值:
spsr = flags | SS | EL1h

只清除 SS:
spsr = flags | EL1h

写回:
SPSR_EL2 = flags | EL1h

Take away: 新功能引入时未复用mitigation

SPSR_EL2 写入点 (共 11 处)

路径1 = 进入 guest 时恢复返回状态

路径2 = 模拟 guest 的 ERET

路径3 = A510 errata 的单步同步

路径4 = 跳过指令时重写状态

路径5 = 注入 EL1 同步异常

路径6 = SError 中止窗口恢复原值

路径7 = 注入宿主异常

路径8 = PSCI 宿主入口常量

路径9 = panic 返回宿主常量

路径10 = stub HVC 软重启常量

路径11 = MOPS 指令处理 (本漏洞)

其余 10 个 writer: 满足返回状态约束

`to_hw_pstate()`

EL2h → EL1h; 非法模式置 PSR_IL_BIT

`__guest_enter(): ERET`

读取 SPSR_EL2

`kvm_hyp_handle_mops()`

未调用 `to_hw_pstate()`, 直接写入 SPSR_EL2

PoC: Return to ArbitraryAddress

```
# ./poc_mops_c
poc_mops_clean poc_mops_control
# ./poc_mops_clean
[*] kvm_hyp_handle_mops SPSR privilege escalation PoC
[*] Target: Linux kernel with CONFIG_KVM_ARM_NV + FEAT_MOPS

[+] vCPU created with nested virt (NV)
[+] GICv3 initialized
[+] vCPU: PC=0x40000000 PSTATE=EL2h SP=0x43fff000

[*] Running guest. If kernel panics -> vulnerability confirmed.

[+] MMIO #1: CurrentEL = 0x8 (EL2)
[ 3823.914213] Kernel panic - not syncing: HYP panic:
[ 3823.914213] PS:00000009 PC:0000000040000028 ESR:86000005
[ 3823.914213] FAR:0000000040000028 HPFAR:0000000000402000 PAR:1de7ec7edbadc0de
[ 3823.914213] VCPU:0000000043acaad8
[ 3823.918737] CPU: 1 UID: 0 PID: 323 Comm: poc_mops_clean Not tainted 7.0.12 #1 PREEMPT
[ 3823.919391] Hardware name: linux,dummy-virt (DT)
[ 3823.919802] Call trace:
[ 3823.920152] show_stack+0x18/0x24 (C)
[ 3823.921157] dump_stack_lvl+0x38/0x90
[ 3823.921387] dump_stack+0x18/0x24
[ 3823.921566] vpanic+0x464/0x4c4
[ 3823.921745] do_panic_on_target_cpu+0x0/0x1c
[ 3823.922080] kvm_unexpected_el2_exception+0x0/0x3c0
[ 3823.922320] hyp_panic+0x0/0x7c
[ 3823.922498] kvm_arm_vcpu_enter_exit+0x6c/0x9c
[ 3823.923070] SMP: stopping secondary CPUs
[ 3823.923946] Kernel Offset: 0x1d65f8a00000 from 0xffff800080000000
[ 3823.924409] PHYS_OFFSET: 0x40000000
[ 3823.924722] CPU features: 0x00000000,001a7005,fbe367e1,957e7f3f
[ 3823.925550] Memory Limit: none
[ 3823.926710] ---[ end Kernel panic - not syncing: HYP panic:
[ 3823.926710] PS:00000009 PC:0000000040000028 ESR:86000005
[ 3823.926710] FAR:0000000040000028 HPFAR:0000000000402000 PAR:1de7ec7edbadc0de
[ 3823.926710] VCPU:0000000043acaad8 ]---
```

图中Kernel Panic日志显示

PS:00000009

即PSTATE=PSR_MODE_EL2h, 说明已经eret to EL2

PC:0000000040000028

即eret后的PC为0x400000028, 而该PC来自Guest且由Guest控制, 因此可以实现ret2addr

详细Writeup和PoC可以参考下文



发现过程：启发自 kvm CVE-2018-18021

来源 writeup

main/writeup/_index.md#CVE-2018-18021—writeup context 的 summary table 里, CVE-2018-18021 被归类为 "CWE-20 improper input validation of guest-set PSTATE.M / core reg file"

```
.zeromdash > workspace_for_kvm-arm64__20260702T045342 > main > patch > CVE-2018-18021.patch
---
22 diff --git a/arch/arm64/kvm/guest.c b/arch/arm64/kvm/guest.c
23 index 3088463bafc1fa..a6c9fbaeefcdd 100644
24 --- a/arch/arm64/kvm/guest.c
25 +++ b/arch/arm64/kvm/guest.c
26 @@ -152,17 +152,25 @@ static int set_core_reg(struct kvm_vcpu *vcpu, const struct kvm_one_reg *reg)
27 }
28
29 if (off == KVM_REG_ARM_CORE_REG(regs.pstate)) {
30     u32 mode = (*(u32 *)va1p) & PSR_AA32_MODE_MASK;
31     u64 mode = (*(u64 *)va1p) & PSR_AA32_MODE_MASK;
32     switch (mode) {
33     case PSR_AA32_MODE_USR:
34         if (!system_supports_32bit_el0())
35             return -EINVAL;
36         break;
37     case PSR_AA32_MODE_FIQ:
38     case PSR_AA32_MODE_IRQ:
39     case PSR_AA32_MODE_SVC:
40     case PSR_AA32_MODE_ABT:
41     case PSR_AA32_MODE_UND:
42         if (!vcpu_el1_is_32bit(vcpu))
43             return -EINVAL;
44         break;
45     case PSR_MODE_EL0t:
46     case PSR_MODE_EL1t:
47     case PSR_MODE_EL1h:
48         if (vcpu_el1_is_32bit(vcpu))
49             return -EINVAL;
50         break;
51     default:
52         err = -EINVAL;
```

.zeromdash > workspace_for_kvm-arm64__20260702T045342 > main > writeup > _index.md > abc # kvm-arm64 — target vulnerability history > abc ## CVE-2018-18021 — AArch64 KVM SPSR/PSTATE.M validation bypass by: 预览 Markdow

kvm-arm64 — target vulnerability history

38 ## CVE-2018-18021 — AArch64 KVM SPSR/PSTATE.M validation bypass via KVM_SET_ONE_REG

- 39
- 40 - **CVE id / title:** CVE-2018-18021 — KVM arm64 `guest.c` mishandles `KVM\_SET\_ONE\_REG`; insufficient restrictions on userspace access to the core register file and PSTATE.M validation does not prevent unintended execution modes.
 - 41 - **Root-cause class:** CWE-20 (Improper Input Validation) of guest/userspace-controlled exception-return state. A guest that can drive `KVM\_SET\_ONE\_REG` (or a nested ERET) can set an illegal `PSTATE.M`, redirecting hypervisor flow of control with full register control, or panic the hypervisor via an illegal exception return.
 - 42 - **Affected component:** `arch/arm64/kvm/guest.c:76` (`core\_reg\_size\_from\_offset` — whitelist of core-register offsets, `default: -EINVAL` at :103, SVE/VREG guard at :115, size-match at :129) and `arch/arm64/kvm/emulate-nested.c:2733` (`kvm\_check\_illegal\_exception\_return` — rejects EL3 / illegal M / 32-bit EL / EL1+TGE, forces `PSR\_IL\_BIT`).
 - 43 - **Writeup URL:** <https://nvd.nist.gov/vuln/detail/CVE-2018-18021>
 - 44 - **Patch commit SHA:** `2a3f93459d689d990b3ecfbe782fec89b97d3279` and `d26c25a9d19b5976b319af528886f89cf455692d` (mainline, torvalds/linux). Verified present in snapshot at `guest.c:76` and `emulate-nested.c:2733`.
 - 45 - **Lesson:** re-verify the `KVM\_SET\_ONE\_REG` core-reg whitelist (`guest.c:76`) and the PSTATE.M illegal-return sanitizer (`emulate-nested.c:2733`) cover every ERET path, including nested virtual ERET (`kvm\_emulate\_nested\_eret`) and SRS/ERETax.

发现过程：根据 CVE-2018-18021 相关的 Context 提出假设

```
.zeromdash > workspace_for_kvm-arm64__20260702T045342 > log > review_log > plan_by_vuln.json > {} 33 > abc objective
645   },
646   {
647     "id": "vuln-034",
648     "bucket": "vuln",
649     "objective": "Investigate whether the PSTATE.M/core-reg validation class (CVE-2018-18021) recurs on ANY ERET path beyond the patched kvm_check_illegal_exception_return: the KVM_SET_ONE_REG core-reg whitelist must be complete and the PSTATE.M sanitizer must cover nested virtual ERET (kvm_emulate_nested_eret), SRS/ERETax, and saved-SPSR slots.",
650     "hints": {
651       "cve": "CVE-2018-18021",
652       "bug_class": "CWE-20 improper input validation of guest-set PSTATE.M / core reg file",
653       "writeup": "main/writeup/_index.md#CVE-2018-18021",
654       "patch": "main/patch/CVE-2018-18021.patch",
655       "candidate_code_locations": [
656         "arch/arm64/kvm/guest.c:76",
657         "arch/arm64/kvm/guest.c:81",
658         "arch/arm64/kvm/guest.c:85",
659         "arch/arm64/kvm/emulate-nested.c:2733",
660         "arch/arm64/kvm/hyp/include/hyp/sysreg-sr.h"
661       ]
662     }
663   },
```

发现过程：审计与复核，iteration 1

Investigation · iter 1 · no-finding

验证三条参考路径：set_core_reg PSTATE.M 白名单、
__sysreg_restore_el2_return_state IL clamp、嵌套 ERET 消毒
结论：CVE-2018-18021 未复发

confidence 0.86 · severity low

被推翻

Critic · iter 1 · revise

独立重新搜索物理 SPSR_EL2 writer，发现 ≥6 个未提及：
kvm_hyp_handle_mops、synchronize_vcpu_pstate 等
核心问题：unclamped writer 依赖的隐含假设被断言但未证明

要求：sink 锚定 + 逐 writer 完备枚举

.zeromdash > workspace_for_kvm-arm64__20260702T045342 > log > review_log > vuln-034.investigation.log

```

1
2  --- iteration 1 ---
3  {
4    "title": "no-finding",
5    "detail": "I did not find a recurrence of CVE-2018-18021 in the reviewed ERET paths. KVM_SET_ONE_REG reaches `kvm_arm_set_reg()` only after the initialized vCPU ioctl gate (`main/repo/arch/arm64/kvm/arm.c:1843`, `main/repo/arch/arm64/kvm/arm.c:1848`, `main/repo/arch/arm64/kvm/arm.c:1863`), and core registers go through `set_core_reg()` (`main/repo/arch/arm64/kvm/guest.c:719`, `main/repo/arch/arm64/kvm/guest.c:726`). The core-reg whitelist covers PSTATE and all five saved-SPSR slots, rejects unknown offsets/alignment/SVE aliases/size mismatches, and PSTATE itself is checked for 32-bit support, EL1 width, NV-only vEL2 modes, and invalid M values before copy-in (`main/repo/arch/arm64/kvm/guest.c:80`, `main/repo/arch/arm64/kvm/guest.c:103`, `main/repo/arch/arm64/kvm/guest.c:107`, `main/repo/arch/arm64/kvm/guest.c:115`, `main/repo/arch/arm64/kvm/guest.c:129`, `main/repo/arch/arm64/kvm/guest.c:242`, `main/repo/arch/arm64/kvm/guest.c:245`, `main/repo/arch/arm64/kvm/guest.c:255`, `main/repo/arch/arm64/kvm/guest.c:258`, `main/repo/arch/arm64/kvm/guest.c:269`). At the host ERET sink, `__sysreg_restore_el2_return_state()` translates virtual EL2 PSTATE to physical EL1 PSTATE and still forces AArch64 EL2-or-higher returns to `EL2h|IL` before writing SPSR_EL2 (`main/repo/arch/arm64/kvm/hyp/include/hyp/sysreg-sr.h:291`, `main/repo/arch/arm64/kvm/hyp/include/hyp/sysreg-sr.h:324`, `main/repo/arch/arm64/kvm/hyp/include/hyp/sysreg-sr.h:328`). Nested virtual ERET is also covered at point of use: the trapped ERET handler routes vEL2 ERET to `kvm_emulate_nested_eret()`, which reads live SPSR_EL2, rejects EL3, illegal M, AArch32, and EL1+TGE returns via `kvm_check_illegal_exception_return()`, handles ERETAX after that check, and only then writes PC/PSTATE (`main/repo/arch/arm64/kvm/handle_exit.c:271`, `main/repo/arch/arm64/kvm/handle_exit.c:288`, `main/repo/arch/arm64/kvm/emulate-nested.c:2733`, `main/repo/arch/arm64/kvm/emulate-nested.c:2744`, `main/repo/arch/arm64/kvm/emulate-nested.c:2770`, `main/repo/arch/arm64/kvm/emulate-nested.c:2775`, `main/repo/arch/arm64/kvm/emulate-nested.c:2803`). The VHE same-regime ERET fast path is a sibling sink, but it falls back to the slow sanitized path when forwarding is needed, admits only EL0t under E2H+TGE or local vEL2 returns translated to EL1, and falls back if ERETAX authentication fails before writing SPSR_EL2/ELR_EL2 (`main/repo/arch/arm64/kvm/hyp/vhe/switch.c:337`, `main/repo/arch/arm64/kvm/hyp/vhe/switch.c:352`, `main/repo/arch/arm64/kvm/hyp/vhe/switch.c:359`, `main/repo/arch/arm64/kvm/hyp/vhe/switch.c:374`, `main/repo/arch/arm64/kvm/hyp/vhe/switch.c:384`). Saved-SPSR slots are not an unsanitized host-ERET source: core `spsr[]` maps to guest SPSR_EL1/AArch32 saved program-status registers, and restore writes them to guest EL1/AArch32 SPSR state, not directly to host SPSR_EL2 (`main/repo/arch/arm64/kvm/guest.c:154`, `main/repo/arch/arm64/kvm/guest.c:157`, `main/repo/arch/arm64/kvm/guest.c:166`, `main/repo/arch/arm64/kvm/hyp/include/hyp/sysreg-sr.h:284`, `main/repo/arch/arm64/kvm/hyp/include/hyp/sysreg-sr.h:365`).",
6    "evidence": {
7      "code_locations": [ ...
17    ],
18    "call_path": [ ...
23    ],
24    "writeup_refs": [ ...
32    ],
33    "patch_refs": [ ...
40    ]
41  },
42  "severity": "low",
43  "confidence": 0.86,
44  "status": "no-finding"
45 }

```

发现过程：审计与复核， iteration 2

Investigation · iter 2 · confirm

以 sink (ERET 消费的物理 SPSR_EL2) 为锚，完备枚举全部 11 个写入点，逐条分类核对不变量；唯一违反者：kvm_hyp_handle_mops
结论：虚拟 EL2h 未经 to_hw_pstate/IL clamp 直写物理 SPSR_EL2

confidence 0.87 · severity critical → high

被确认

Critic · iter 2 · confirm

独立重推导全部 5 个承重步骤（读完整源码非引用片段）：
fixup 改写内存 pstate → MOPS 无守卫分发 → 直写 → ERET 重入
三方视角 (Felix Wilhelm / Ian Beer / Tavis Ormandy) 全部收敛

confidence 0.82 · severity high

.zeromdash > workspace_for_kvm-arm64__20260702T045342 > log > review_log > vuln-034.investigation.log

```

48 --- iteration 2 ---
49 {
50   "title": "VHE MOPS re-entry can write virtual EL2 PSTATE.M into physical SPSR_EL2",
51   "detail": "Entrypoint/trust boundary/sink: the task is about userspace/guest-controlled exception-return state entering through KVM_SET_ONE_REG core registers and nested-virtualization sysreg/ERET state, crossing into the EL2 guest-entry path whose privileged sink is physical SPSR_EL2 consumed by __guest_enter's ERET; the attack-surface report file was empty, so this framing comes from the subtask and the CVE writeup refs. CVE-2018-18021 was a PSTATE.M validation failure: the local writeup says guest/userspace-controlled KVM_SET_ONE_REG or nested ERET state can redirect hypervisor control flow or panic EL2 if illegal PSTATE.M reaches exception return state, and the historical patch added mode validation for regs.pstate. The reference entry path is mostly safe: set_core_reg validates regs.pstate and permits PSR_MODE_EL2h/EL2t only for NV vCPUs; __sysreg_restore_el2_return_state then calls to_hw_pstate(), translating virtual EL2h/EL2t to physical EL1h/EL1t before write_sysreg_el2(pstate, SYS_SPSR), and clamps remaining AArch64 EL2-or-higher modes to EL2h/IL. The bypass is a sibling re-entry path after guest exit on VHE: __compute_hcr records VCPU_IN_HYP_CONTEXT for virtual EL2 guests and states PSTATE.M will be fixed up on exit; fixup_guest_exit first synchronizes physical SPSR_EL2 into vcpu->arch.ctx.regs.pstate, then when VCPU_IN_HYP_CONTEXT is set rewrites the in-memory mode from EL1t/EL1h back to virtual EL2t/EL2h. The same fixup then dispatches ESR_ELx_EC_MOPS to kvm_hyp_handle_mops, which clears only DBG_SPSR_SS and writes *vcpu_cpsr (vcpu) directly with write_sysreg_el2(*vcpu_cpsr(vcpu), SYS_SPSR); __fixup_guest_exit returns true and the run loop immediately calls __guest_enter again, whose ERET consumes the now-raw physical SPSR_EL2. Under VHE, write_sysreg_el2(v, SYS_SPSR) macro-expands to write_sysreg_s(v, SYS_SPSR_EL1), the physical EL2 return SPSR alias, so the virtual EL2 mode produced by the VHE fixup can reach the host ERET sink without to_hw_pstate() or the IL clamp. This is not the guest SPSR_EL2 slot path: __sysreg_restore_el2_state writes the guest SPSR_EL2 slot with write_sysreg_el1(..., SYS_SPSR), which under VHE targets SYS_SPSR_EL12, and it occurs in __vcpu_load_switch_sysregs before the run loop later executes sysreg_restore_guest_state_vhe and __sysreg_restore_el2_return_state. Nested virtual ERET/ERETx itself is covered because kvm_emulate_nested_eret checks SPSR_EL2 with kvm_check_illegal_exception_return before assigning *vcpu_cpsr, and the VHE fast ERET path either rejects to the slow path or maps EL2t/EL2h to EL1t/EL1h before writing physical SPSR_EL2. The saved SPSR core-reg slots are raw-copyable, but their consumers do not target physical SPSR_EL2: SPSR_EL1 restores through write_sysreg_el1(..., SYS_SPSR) (SPSR_EL12 under VHE), and SPSR_ABT/UND/IQ/FIQ restore to AArch32 banked SPSRs. The explicit SRS audit found no SRS instruction, ESR_ELx_EC_SRS, or symbol hit under arch/arm64/kvm; the only case-insensitive substring hit is the word ISRs in vgic/vgic.c:56, so I treated the subtask's SRS item as save/restore-state and audited sysreg-sr.h save/restore sinks.",
52   "evidence": {
53     "code_locations": [ ...
105     "call_path": [ ...
108     "writeup_refs": [ ...
122     "patch_refs": [ ...
128     "spsr_el2_writers": [
129       { ...
135       { ...
141       {
142         "location": "arch/arm64/kvm/hyp/include/hyp/switch.h:460",
143         "function": "kvm_hyp_handle_mops",
144         "macro_target": "VHE physical SPSR_EL2 alias SYS_SPSR_EL1; non-VHE SYS_SPSR_EL2/hVHE SYS_SPSR_EL1",
145         "source": "*vcpu_cpsr(vcpu) after VHE fixup_guest_exit may have been converted from physical EL1 to virtual EL2 at arch/arm64/kvm/hyp/vhe/switch.c:548-561",
146         "classification": "vulnerable: assumed-already-sanitized but virtual EL2 survives to physical SPSR_EL2 without to_hw_pstate or IL clamp"
147       },

```

案例2: Milvus #53759, sparse search 堆溢出

Search / HybridSearch 的查询行在 C++ 解码时触发越界写; 漏洞发生在检索之前

Search / HybridSearch 接收稀疏查询行, 每项为 8 B, 行长度由输入控制。

C++ 按整项分配、按原始长度复制, 未在复制前拒绝非法尾部。

长度为 $8k + r$ ($k \geq 1, 1 \leq r \leq 7$) 时, 越界写 r B; 尚未证明 RCE。

Sparse Search: 仅保存和匹配非零维度

倒排索引按查询的非零维度找到候选文档, 累加共同维度的贡献并排序。

输入为稀疏向量 q 与 topK , 输出为 Top-K 文档 ID 与得分。

稀疏查询 q

$\{(2, 0.5), (9, 1.2)\}$

倒排索引 + 评分

按维度取候选, 累加得分

Top-K ID + score

排序并返回结果

IP 示例: $\text{score}(q, d) = \sum q[i] \times d[i]$; BM25 的评分由 metric 决定。

行编码: $(\text{uint32 index}, \text{float32 value}) \times \text{NNZ}$, 每项 8 B; 通过 PlaceholderGroup 传入。

漏洞原因：按整项分配，却按原始长度复制

size / 8 向下取整；validate=true 的长度检查发生在 FastMemcpy 之后

漏洞代码

```
n = size / 8;
SparseRow row(n); // n * 8 B
FastMemcpy(row.data(), data, size);
if (validate)
    AssertInfo(size % 8 == 0, ...);
```

修复 #53759

```
if (size % 8 != 0) {
    ThrowInfo(ErrorCode::DataFormatBroken,
        "Invalid size for sparse row data");
}
n = size / 8;
SparseRow row(n);
FastMemcpy(row.data(), data, size);
if (validate) { /* value checks */ }
```

9 B 输入：分配 8 B，复制 9 B

SparseRow(1): 8 B 堆对象



对象内 8 B

越界 1 B

$8k + r \text{ B} \rightarrow \text{越界写 } r \text{ B}$

漏洞：分配 → 复制 → 校验

修复：校验 → 分配 → 复制

Exploit: Heap overflow before assert

```
00%
Last login: Thu Oct 22 00:45:46 on ttys001
You have new mail.
[swing at Air in ~ 26-10-22 - 1:32:20]
o ssh swing@192.168.50.3
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Wed Oct 21 10:32:17 2026 from 192.168.50.6
swing@Terra:~$ podman logs -f --since 0s milvus-standalone 2>&1 \
| grep --line-buffered -E 'uid=[0-9]+\(\root\)|Invalid size for sparse'
[]
```

```
Last login: Thu Oct 22 01:32:19 on ttys003
You have new mail.
[swing at Air in ~ 26-10-22 - 1:32:21]
o ssh swing@192.168.50.3
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Wed Oct 21 10:32:22 2026 from 192.168.50.6
swing@Terra:~$ cd /home/swing/Desktop/milvus/spec037-poc
swing@Terra:~/Desktop/milvus/spec037-poc$ .venv/bin/python prepare_collection.py --target 127.0.0.1:19530 --token
root:Milvus
usage: prepare_collection.py [-h] [--target TARGET] [--token TOKEN]
prepare_collection.py: error: argument --token: expected one argument
bash: root:Milvus: command not found...
swing@Terra:~/Desktop/milvus/spec037-poc$
```

发现过程：启发自 Qdrant CVE-2023-38975 + unfuzzed

多个来源

1. 相似软件历史漏洞表明：攻击者控制的 `vector size/count` 或能触发 `allocation/arithmetic DoS`
2. Milvus Query Parser 没有 `fuzz coverage`

```
.zeromdash > workspace_for_milvus > similar > qdrant > writeup > CVE-2023-38975.md > # CVE-2023-38975 — DoS via
1 # CVE-2023-38975 — DoS via unbounded vector dimension (chunked_vectors.rs)
31 ## Writeup URLs (canonical)
32 - https://nvd.nist.gov/vuln/detail/CVE-2023-38975
33 - Reporter issue: https://github.com/qdrant/qdrant/issues/2268 (360AIVul, opened 2023-07-17)
34
35 ## Patch
36 - **not found** GitHub issue #2268 is closed but links no PR; the GHSA has no
37 package/CPE entry, Dependabot is disabled for it, and no patched version is
38 recorded. 'writeup: partial' — root cause and impact are confirmed from the
39 reporter issue; the patch reference is unavailable, so it is flagged rather
40 than invented.
41
42 ## Why it matters for the target (milvus)
43 Numeric range validation on collection/vector parameters. For milvus, every API
44 that accepts a 'dim' / 'size' / 'count' / 'length' that drives allocation or
45 arithmetic must bound it to a sane maximum **before** use. In C++ code paths an
46 unbounded dimension is not just a DoS but a potential memory-corruption
47 precursor (oversized allocation / arithmetic overflow feeding subsequent
48 indexing) — pair this with the OOB-read entry (PR #9268) when probing
49 on-disk vector storage loaders.
50
51 ## Sources
52 - https://github.com/advisories/GHSA-j4cw-vvxw-4jcv
53 - https://nvd.nist.gov/vuln/detail/CVE-2023-38975
54 - https://github.com/qdrant/qdrant/issues/2268
```

```
.zeromdash > workspace_for_milvus > main > attack_surface > fuzz_drivers.md > # Fuzz Drivers — milvus > 预览 Markdown
1 # Fuzz Drivers — milvus
2
3 Target: **milvus** ('github.com/milvus-io/milvus', Go 1.26.5 + C++ 'internal/core' segcore/
knowhere engine).
4 Repo root: '/Users/bytedance/Projects/Security/ZeroMDASH/.zeromdash/workspace_for_milvus/main/repo'
5 Analysis focus (applied): *security issues causing remote code execution or unauthorized access* —
no driver exercises this focus because none exist.
6
7 ## No existing fuzz drivers
8
9 No fuzz harnesses were found in the repository, in OSS-Fuzz, or referenced by the
10 project's CI. The conclusion below is a **verified negative**; every search path and
11 grep pattern tried is listed so the critic can reproduce the absence.
```

发现过程：随后生成了个看似离结果很远的攻击面和初始假设

analyze_attack_surface节点早期猜想的攻击面

公共 Search/HybridSearch 请求携带序列化的 PlaceholderGroup: SearchRequest

- PlaceholderGroup protobuf
- repeated Placeholder
- repeated bytes values

而Agent观察到,

- values_size() 来自客户端提供的 repeated field, 且 num_of_queries_ = ph.values_size();
- values(0).size() 来自客户端提供的 bytes, 且 line_size = ph.values(0).size();
- 以上两者都参与 C++ allocation 需要确认 proxy 是否在进入 C++ 前约束了 $nq \times dim \times value_size$.

```
.zeromdash > workspace_for_milvus > log > review_log > plan_by_spec.json > {} 37 > {} hints
690 {
691   "id": "spec-037",
692   "bucket": "spec",
693   "objective": "Verify PlaceholderGroup vector value sizes, nq, topk, and dim products are bounded before C++ reserve/insert allocations; if not, judge allocation DoS or memory-safety impact.",
694   "hints": {
695     "spec_clause": "gRPC message length prefix; Protocol Buffers repeated bytes field-size bounds",
696     "candidate_code_locations": [
697       "internal/core/src/query/Plan.cpp:169",
698       "internal/core/src/query/Plan.cpp:176",
699       "internal/core/src/query/Plan.cpp:190",
700       "internal/core/src/query/Plan.cpp:200",
701       "internal/distributed/proxy/service.go:316",
702       "pkg/util/paramtable/grpc_param.go:40",
703       "pkg/util/paramtable/grpc_param.go:205"
704     ],
705     "rationale": "Even with a total gRPC size cap, attacker-controlled repeated vector values can drive large allocations or arithmetic overflow if product bounds are missing.",
706     "priority_reason": "Medium-low priority because it is mostly DoS, but public Search reaches C++ allocation at attack_surface.md:350-357."
707   }
708 }
```

```
.zeromdash > workspace_for_milvus > main > attack_surface > attack_surface.md > abc # Milvus - Attack
1 # Milvus - Attack-Surface Report (data-flow + invariant audit)
198 ## 5. Hypothesis list
305 ### Memory safety (C++ segcore, unfuzzed)
350 **H-NQ - `internal/core/src/query/Plan.cpp:190,200`**
351 · memory safety (allocation DoS) · opensearch CVE-2023-31419 ·
352 `target.reserve(line_size * element.num_of_queries_)` where both factors are
353 client-controlled from the `PlaceholderGroup` proto (`values(0).size()` up to
354 2GB, `values_size()` up to  $2^{32}$ ). No `MaxNq`/`MaxPlaceholderBytes` enforced
355 before `ParsePlaceholderGroup` (`Plan.cpp:169-170`). Check: confirm proxy
356 (`task_search.go`) bounds `nq` * `topk` * `dim`; send a search with one 4GB
357 placeholder.
```

发现过程：3轮审计与复核后扫到了 SparseFloat 的 Search

Investigation · iter 1 · no-finding

对象：传入 C++ ParsePlaceholderGroup() 的序列化 PlaceholderGroup bytes, values[] 保存查询行

H-NQ：查询行长度 / nq 是否驱动不受限的分配？当轮只分析 dense / embedding-list 的 reserve() 与 insert()。

调用：segcore.NewSearchRequest() → C.ParsePlaceholderGroup() → C++ ParsePlaceholderGroup()。

当轮引用的检查函数与异常处理

searchTask.PreExecute() → checkNq() / getNq() / getNqFromSubSearch() → validateNQLimit()：默认 nq ≤ 16384。

gRPC server → grpc.MaxRecvMsgSize()：默认单条消息 ≤ 256 MiB；C 包装层 catch(std::exception&) → CStatus。

当轮结论：no-finding；来源枚举尚不完整

以这些约束否定 H-NQ，但将单一 C 调用点当作路径已覆盖，未枚举 Go 层谁生成 / 改写 PlaceholderGroup bytes。

尚未分析 ParsePlaceholderGroup() 中 SparseFloatVector 的独立分配与复制路径。

confidence 0.88 · severity low

补全证据

Investigation · iter 2 · no-finding

补枚举 6 类生成 / 改写 PlaceholderGroup bytes 的路径

① 透传：GetPlaceholderGroup() → internalSubReq.PlaceholderGroup。

② SearchByPK：handleIfSearchByPK() → FieldDataToPlaceholderGroupBytesWithCount() → request.SearchInput.PlaceholderGroup。

③ Hybrid：ConvertPlaceholderGroup() → convertPlaceholder() → proto.Marshal() → internalSubReq.PlaceholderGroup。

④ 文本 Embedding：FunctionExecutor.ProcessSearch() → TextEmbeddingFunction.ProcessSearch()

→ Float32VectorsToPlaceholderGroup() → proto.Marshal() → req.PlaceholderGroup / req.SubReqs[].PlaceholderGroup。

⑤ HTTP REST：generatePlaceholderGroup() / vectors2PlaceholderGroupBytes() → SearchRequest.SearchInput.PlaceholderGroup。

⑥ QueryNode BM25-IDF：BuildIDF() → SparseVectorDataToPlaceholderGroupBytes() → req.PlaceholderGroup。

核对检查与计数关系：checkNq() → getNq() / getNqFromSubSearch() → validateNQLimit()

req.Nq 非零时可与 values_size() 不一致；文本行数由 MaxBatch() 检查；核对各节点 grpc.MaxRecvMsgSize()。

当轮仍为 no-finding；继续以消息总量与 catch → CStatus 论证；但未分析 SparseFloatVector 分支

confidence 0.88 · severity low

发现遗漏

Investigation · iter 3 · confirm

补齐构造器：knowhere v3.0.6 sparse_utils.h，SparseRow(n) 恰好分配 $n \times \text{element_size}()$ B

new uint8_t[n * element_size()]; element_size() = sizeof(uint32_t) + sizeof(float) = 8 B，无额外容量。

Search / HybridSearch 到危险写入的调用链

ConvertPlaceholderGroup() 在 SparseFloat 类型匹配时返回原 bytes；internalSubReq.PlaceholderGroup = subReq.GetPlaceholderGroup()。

segcore.NewSearchRequest() → C.ParsePlaceholderGroup() → C++ ParsePlaceholderGroup() 的 SparseFloatVector 分支

→ SparseBytesToRows(..., true) → CopyAndWrapSparseRow()。

写入早于长度校验：size = 8k + r → n = size / 8 = k → SparseRow(k) 分配 8k B

FastMemcpy(..., size) 写 8k + r B → 越界 r B → AssertInfo(size % 8 == 0, ...) 抛错 → C 包装层 catch → CStatus。

9 B 输入 → 分配 8 B → 复制 9 B → 校验前越界写 1 B；异常捕获发生在写入之后。

当轮确认：k ≥ 1、1 ≤ r ≤ 7 的堆溢出；Search 未前置调用 ValidateSparseFloatRows()

越界长度与尾部内容由输入控制；稳定利用与 RCE 尚未证明。

confidence 0.82 · severity high

被确认

Critic · iter 1 · revise

独立发现 3 类生成 / 写入 PlaceholderGroup bytes 的路径，要求分别补证

① 透传：subReq.GetPlaceholderGroup() → initAdvancedSearchRequest() 写 internalSubReq.PlaceholderGroup。

② BM25 / SearchByPK 重建：handleIfSearchByPK() → FieldDataToPlaceholderGroupBytesWithCount(fieldData)

→ 写 request.SearchInput.PlaceholderGroup，并将 request.Nq 重设为返回的行数。

③ Hybrid 转换：initAdvancedSearchRequest() → ConvertPlaceholderGroup() → convertPlaceholder() → proto.Marshal()。

fp32 查询行转换为 fp16 / bf16，再写 internalSubReq.PlaceholderGroup。

补证要求：重编码后的行数是否仍是检查过的行数？

核对重建函数返回 len(Values)，转换函数按输入行数生成 convertedValues；handleIfSearchByPK() 应先于 PreExecute()。

核对内部监听器 grpc.MaxRecvMsgSize()；补 ParsePlaceholderGroup() 中 offsets.reserve(num_of_queries_ + 1)。

裁决：revise，原 no-finding 暂保留；本轮尚未确认 SparseFloat 堆溢出

confidence 0.82 · severity low

Critic · iter 2 · revise

漏查的分叉点：C++ ParsePlaceholderGroup()，Plan.cpp:171–174

ph.type() == PlaceholderType::SparseFloatVector → SparseBytesToRows(ph.values(), /*validate=*/true)

→ CopyAndWrapSparseRow(row.data(), row.size(), validate)；else 分支才走 dense / embedding-list 的 reserve() / insert()。

CopyAndWrapSparseRow() 的另一套机制：分配整项，复制原始长度，最后校验

n = size / element_size() // 向下取整；SparseRow row(n) // 分配大小尚未核实。

FastMemcpy(row.data(), data, size) → if (validate) AssertInfo(size % element_size() == 0, ...)。

长度检查发生在复制之后；grpc.MaxRecvMsgSize() 只限制消息总字节数，不检查单行的 8 B 对齐。

裁决：revise，先取得构造器证据并核查 Search 前置校验

knowhere 子模块缺失：获取 SparseRow(n) 构造器、element_size()，判断缓冲区是否恰好 $n \times \text{element_size}()$ B。

核查 Search → ConvertPlaceholderGroup() 是否调用 ValidateSparseFloatRows()；证明非法长度是否可到达 FastMemcpy()。

此时是具体的潜在越界机制，分配大小尚未核实，不能直接 confirm。

confidence 0.80 · severity medium

Critic · iter 3 · confirm

独立复核分配大小、复制时序与可达调用链

重新获取 knowhere v3.0.6，核实 SparseRow(n) 分配恰好 8n B，FastMemcpy() 写完整 size 后才做 AssertInfo() 长度检查。

重查 ParsePlaceholderGroup() → SparseBytesToRows() → CopyAndWrapSparseRow()，确认不属于 dense / emb-list 的 reserve()。

独立搜索 ValidateSparseFloatRows() 所有调用方：len(row) % 8 != 0 会拒绝非法行

调用位于 insert / upsert / import，Search 未调用；ConvertPlaceholderGroup() 类型匹配时直接返回原 bytes。

task_search.go:631 写 raw blob；ParsePlaceholderGroup() 内 check_data_type() 只核对类型枚举，不检查单行字节长度。

此前的检查为何不能防止本次越界写？

grpc.MaxRecvMsgSize() 限制消息总量；validateNQLimit() 检查 nq；二者均不约束单行 size % 8。

catch(std::exception&) 只捕获后续抛错，不能撤销已发生的 FastMemcpy() 写入。

裁决：confirm；Ian Beer / Tavis Ormandy 两种视角均确认校验前 1–7 B 堆溢出

dense / emb-list 保持 no-finding；RCE 仍需运行时与堆布局分析。

confidence 0.82 · severity high



repo

<https://github.com/JANUS-SEC/ZeroMDASH>

author

<https://github.com/Cossack9989>

<https://github.com/winmin>