Unmasking The Unseen: A Deep Dive Into Modern Linux Rootkits And Detection Strategies

Friday, September 26, 2025

Presenters: Remco Sprooten & Ruben Groenewoud

elastic | The Search AI Company

Ruben Groenewoud

Senior Security Research Engineer
Threat Research and Detection Engineering

Remco Sprooten

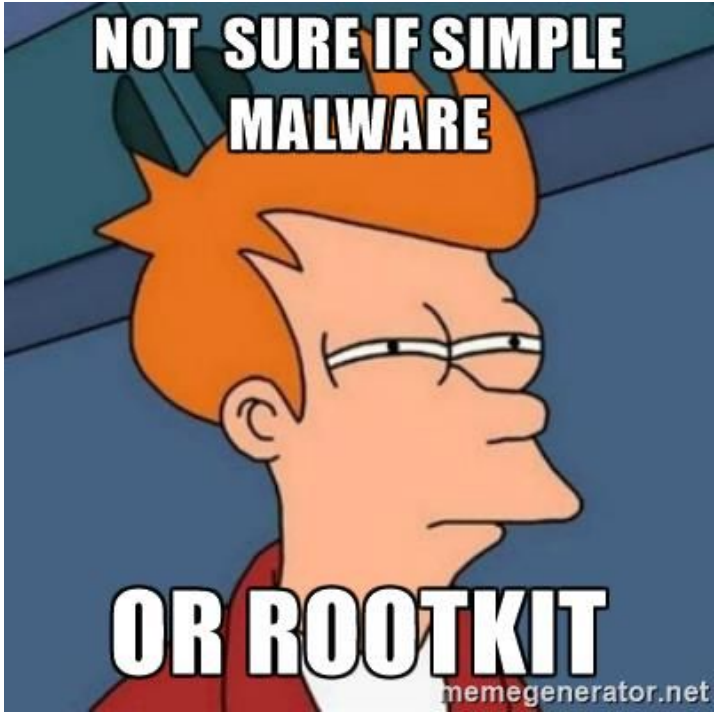
Principal Security Research Engineer
Elastic Security Labs

Agenda

01	**Overview**
02	**Hooking Techniques**
03	**Statistics**
04	**PUMAKIT Case Study**
05	**Detection Engineering**

Rootkits

- Malicious software designed to hide processes, files, or activities from detection.
- On Windows, typically kernel drivers or bootkits; on Linux, userland tricks, LKMs, or “newer” interfaces (eBPF, io_uring).
- Goal: stealth, persistence, and evasion of security controls.



Linux Rootkit Implementations

The “evolution” of Linux Rootkits

Userland

- **Mechanism**: manipulates dynamic linker to hook `libc` functions.
- **Limitation**: confined to userland APIs and easily detected.
- **Examples**: *JynxKit* (2012) & *Azazel* (2014).

Linux Rootkit Implementations

The “evolution” of Linux Rootkits

Userland

- **Mechanism**: manipulates dynamic linker to hook `libc` functions.
- **Limitation**: confined to userland APIs and easily detected.
- **Examples**: *JynxKit* (2012) & *Azazel* (2014).

Kernel-Space

- **Mechanism**: LKMs hook syscalls & kernel structures.
- **Limitation**: blocked by module signing; kernel version dependent & crash-prone.
- **Examples**: *Reptile* (2017) & *Diamorphine* (2013).

Linux Rootkit Implementations

The “evolution” of Linux Rootkits

Userland

- **Mechanism**: manipulates dynamic linker to hook `libc` functions.
- **Limitation**: confined to userland APIs and easily detected.
- **Examples**: *JynxKit* (2012) & *Azazel* (2014).

Kernel-Space

- **Mechanism**: LKMs hook syscalls & kernel structures.
- **Limitation**: blocked by module signing; kernel version dependent & crash-prone.
- **Examples**: *Reptile* (2017) & *Diamorphine* (2013).

eBPF

- **Mechanism**: attach programs, kprobes, tracepoints and LSMs.
- **Limitations:** eBPF support (3.18+, mature by 4.9); visible to BPF tooling; verifier restrictions.
- **Examples**: *TripleCross* (2021) & *Boopkit* (2022).

Linux Rootkit Implementations

The “evolution” of Linux Rootkits

Userland

- **Mechanism**: manipulates dynamic linker to hook `libc` functions.
- **Limitation**: confined to userland APIs and easily detected.
- **Examples**: *JynxKit* (2012) & *Azazel* (2014).

Kernel-Space

- **Mechanism**: LKMs hook syscalls & kernel structures.
- **Limitation**: blocked by module signing; kernel version dependent & crash-prone.
- **Examples**: *Reptile* (2017) & *Diamorphine* (2013).

eBPF

- **Mechanism**: attach programs, kprobes, tracepoints and LSMs.
- **Limitations:** eBPF support (3.18+, mature by 4.9); visible to BPF tooling; verifier restrictions.
- **Examples**: *TripleCross* (2021) & *Boopkit* (2022).

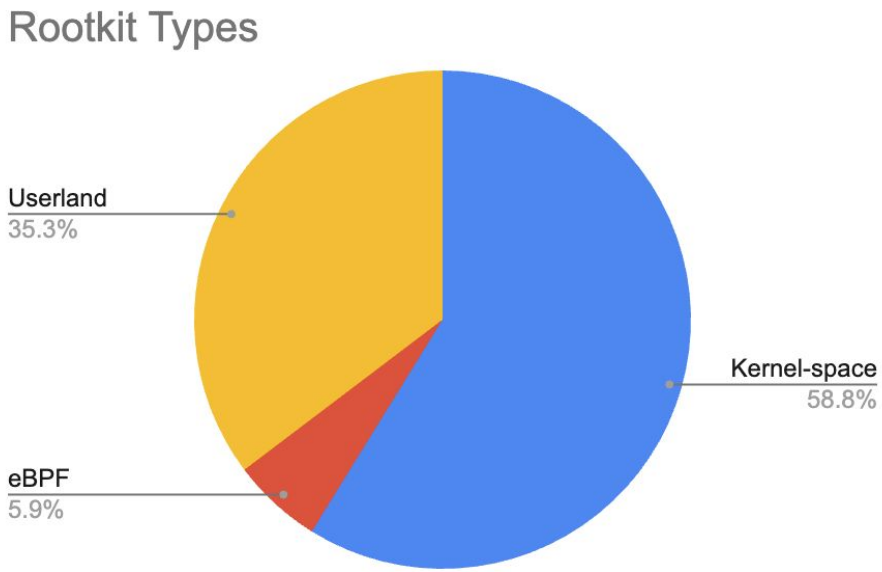
io_uring

- **Mechanism:** async I/O batching via shared rings, reducing syscall visibility.
- **Limitation:** Linux 5.1+; no exec; liburing not default on most systems.
- **Examples:** *Curing* (2024) & *RingReaper* (2025).

Rootkit Statistics

Dataset

- Open-Source samples
  - Boths research as non-research
  - Investigated based on "Main" branch if available
- Closed sources samples
  - Multiple samples per family
  - Reverse engineered recent samples when available

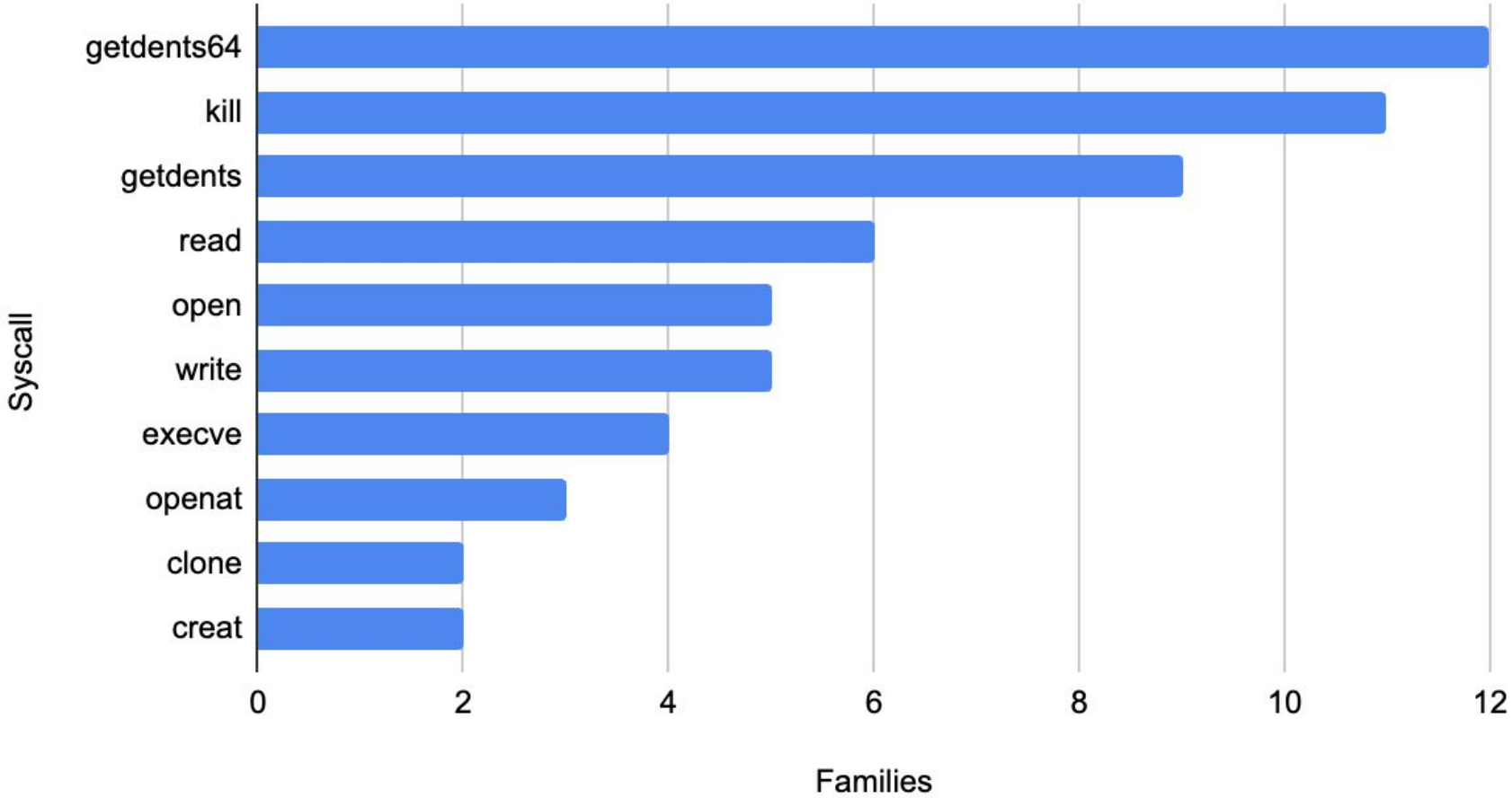


Hooking Techniques

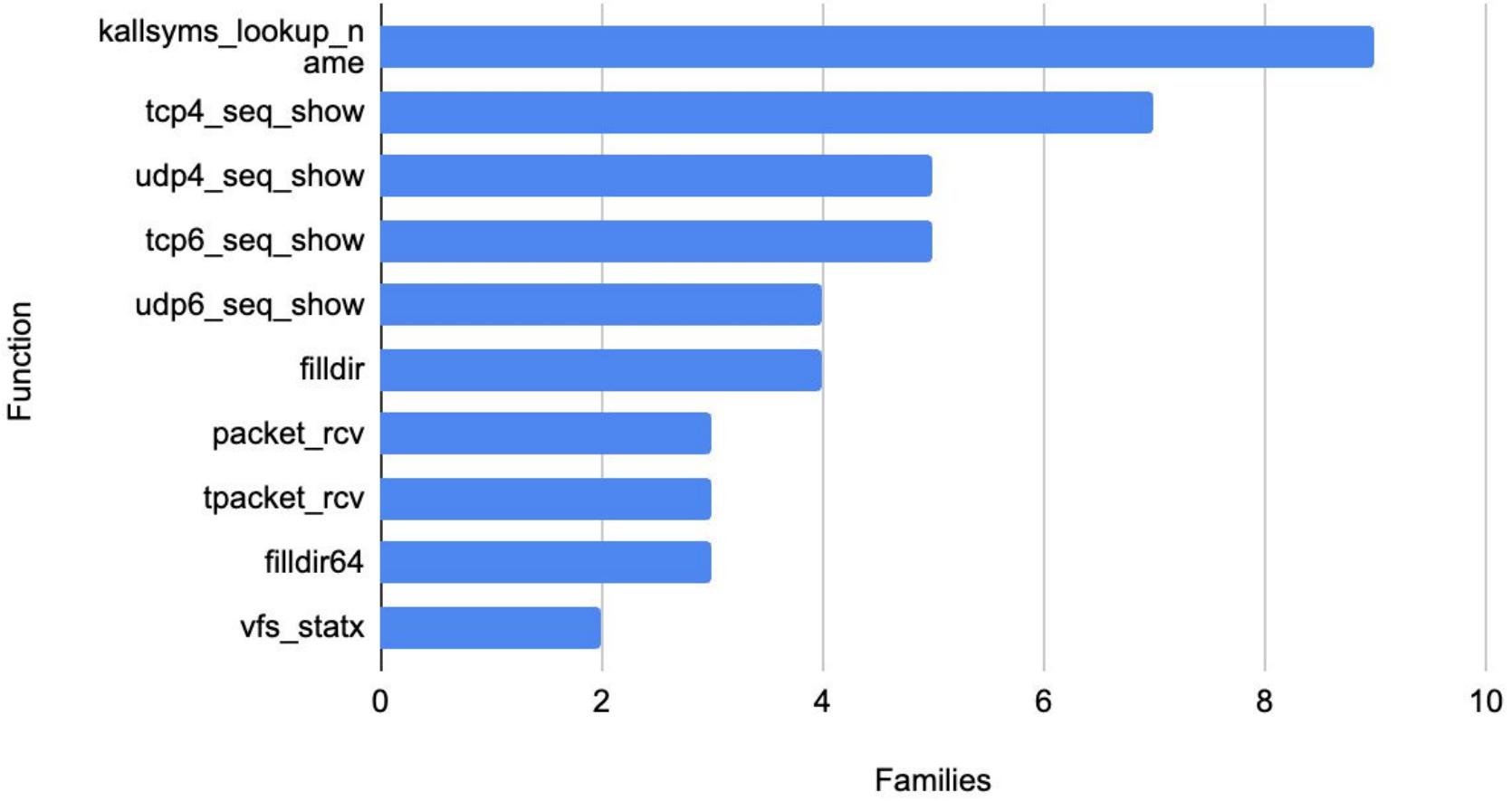
Technique	Mechanism	Modern Status
IDT Hooking	Hijack int 0x80 entry point	❌ Obsolete
Syscall Table Hooking	Overwrite function pointer	❌ Obsolete (Kernel 6.9+)**
Inline Hooking	Overwrite start of function with JMP	🔒 Mitigated (by Lockdown & W^X)
VFS Hooking	Replace function pointer	⚠️ Viable, but limited
Ftrace Hooking	Register a legitimate callback hook	✅ Viable
Kprobes Hooking	Register a dynamic debug probe	✅ Viable
eBPF Hooking	Attach a sandboxed BPF program	✅ Viable

** We will get back to this

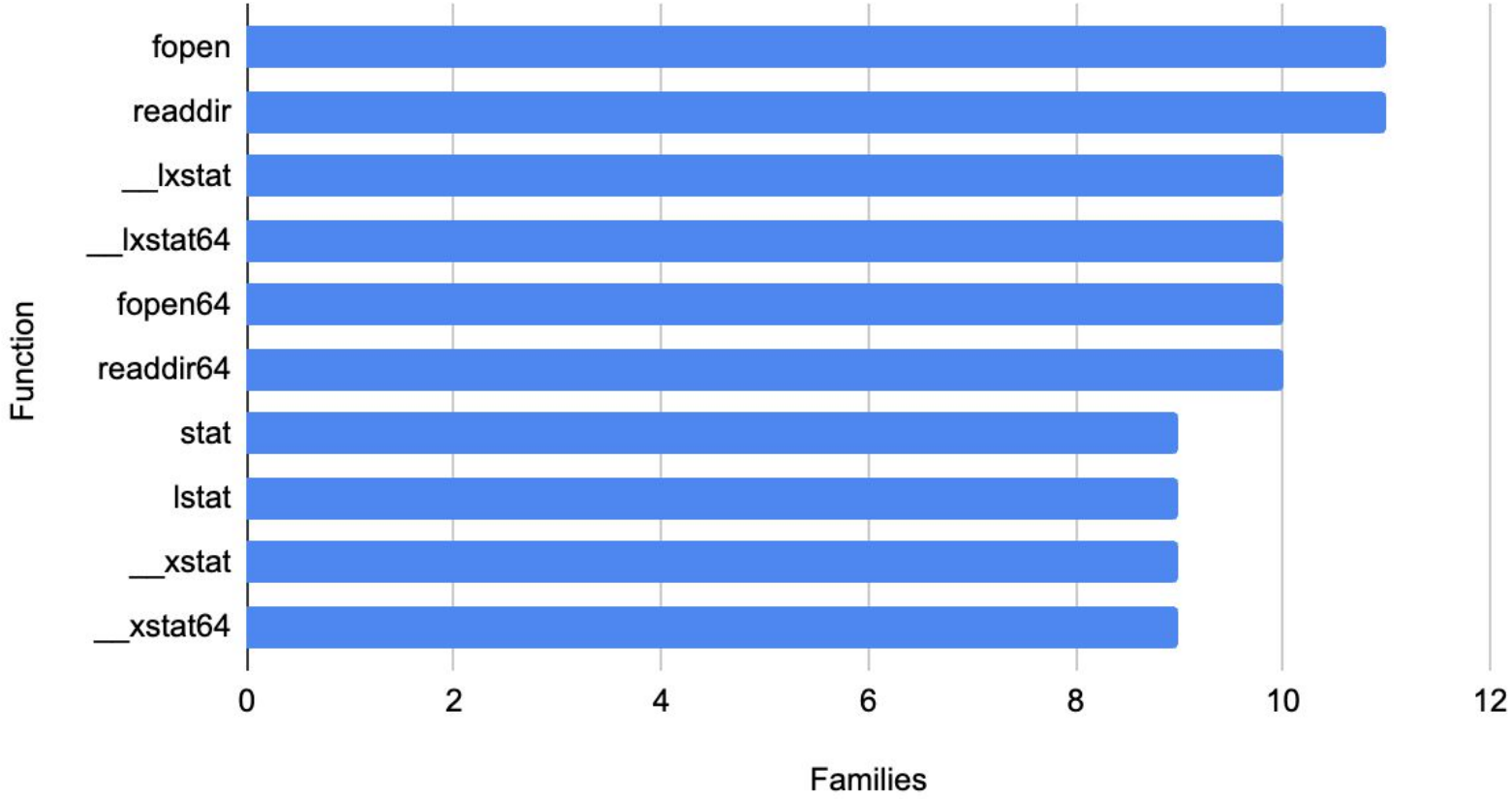
KERNEL ROOTKITS - Top 10 Syscalls



KERNEL ROOTKITS - Top 10 Functions



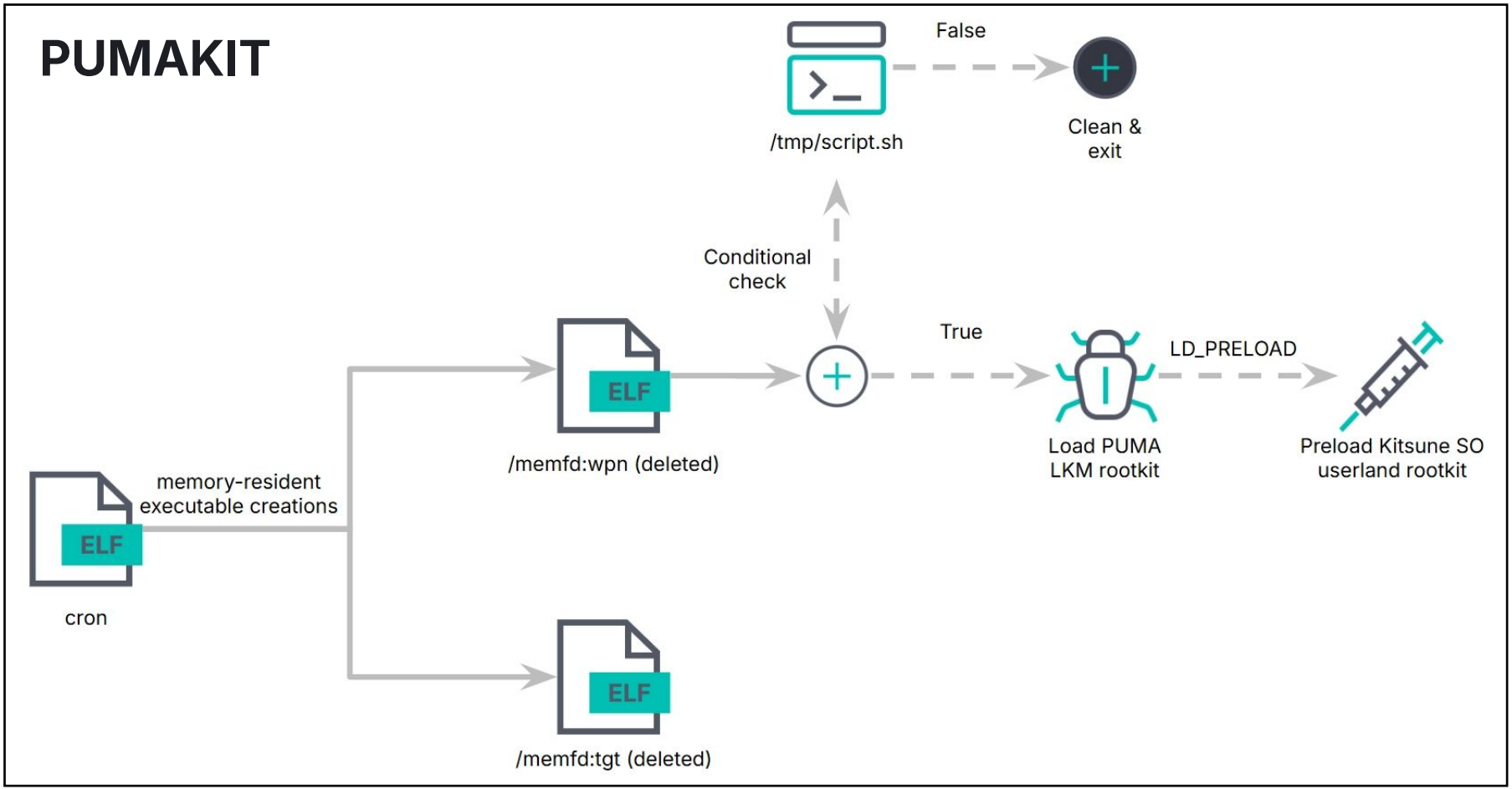
USERLAND ROOTKITS - Top 10 Functions



Rootkit Detection Statistics

Rootkit	Basic detections	Stripped	Null byte added
Azazel	36/66	**19/66**	21/66
Bedevil*	32/66	32/66	21/66
BrokePKG	7/66	**3/66**	3/66
Diamorphine	33/66	**8/64**	22/66
Kovid	27/66	**1/66**	15/66
Modbkit	29/66	**6/66**	17/66
Reptile	32/66	**3/66**	20/66
Snapekit	30/66	**3/66**	19/66
Symbiote	42/66	**8/66**	22/66
TripleCross	31/66	**17/66**	19/66

PUMAKIT



PUMAKIT

Capabilities

- Connection concealment
- Disarm SELinux
- Track programs launched
- Hides files, directories, and processes
- Credential and Key Theft**

```
# rmdir zarya.c.0
```

```
char const data_403c20[0x20] = "-----BEGIN RSA PRIVATE KEY-----", 0
char const data_403c40[0x1f] = "-----BEGIN EC PRIVATE KEY-----", 0

                                                00                .

char const data_403c60[0x24] = "-----BEGIN OPENSSH PRIVATE KEY-----", 0

            00 00 00 00                                   ....

char const data_403c88[0x20] = "-----BEGIN DSA PRIVATE KEY-----", 0
```

```
char const data_40396c[0xc] = "s password:", 0
char const data_403978[0xf] = " password for ", 0
char const data_403987[0xe] = "New password:", 0
char const data_403995[0x12] = "Current password:", 0
char const data_4039a7[0x10] = "Enter password:", 0
char const data_4039b7[0x1d] = "Enter same passphrase again:", 0
char const data_4039d4[0x1a] = "Enter passphrase for key ", 0
char const data_4039ee[0x9] = "-----END", 0
char const data_4039f7[0xf] = "sys_call_table", 0
```

PUMAKIT

```
struct rootkit_hooked_functions* hook = &ftrace_hook_table

do
    if (hook->prologue_match_addr != 0)
    label_4031a5:

        if (hook->resolved_func_addr.d == 0)
            void* prologue_match_addr = hook->prologue_match_addr

            if (prologue_match_addr != 0)
                hook->fops.private = 0x4000014
                hook->fops.next = sub_4006a0

                if (ftrace_set_filter_ip(&hook->fops.next, prologue_match_addr, 0,
                        0) == 0 && register_ftrace_function(&hook->fops.next) == 0)
                    hook->resolved_func_addr.d = 1
    else
        uint64_t p_symbol = kallsyms_lookup_name(hook->hooked_function)
```

PUMAKIT

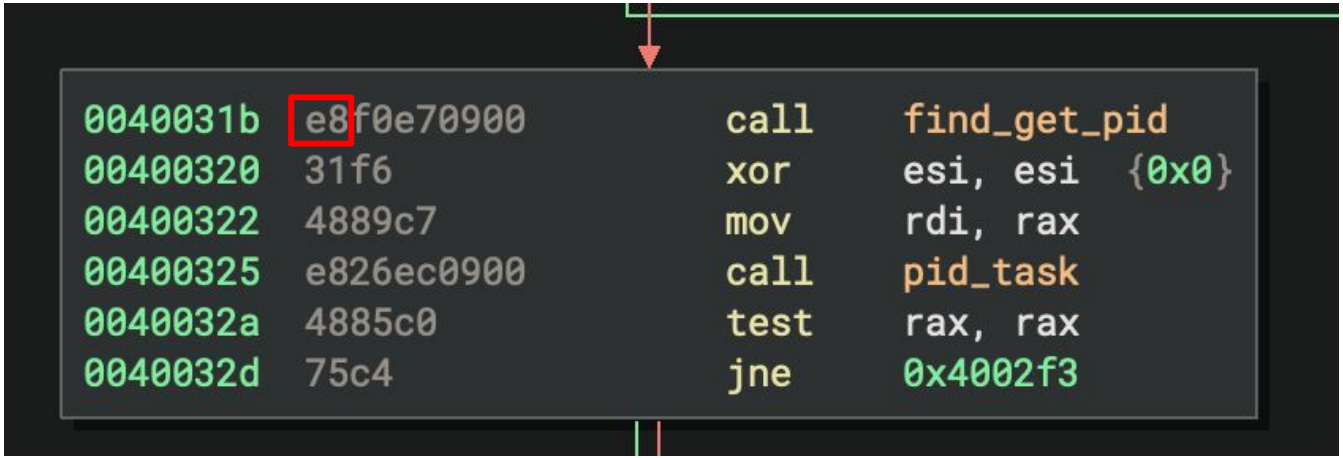
```
004031f5        while (&ideal_nops != hook)
004031f5
004031fb        disable_write_protection()
00403200        uint64_t (** p_sys_call_table_1)(struct pt_regs const* arg1
00403207        int64_t rdx_1 = p_sys_call_table_1[21]
0040320e        p_sys_call_table_1[0x15] = hooked_access
00403219        data_4568f8 = rdx_1
00403220        int64_t rdx_2 = p_sys_call_table_1[3]
00403224        p_sys_call_table_1[3] = hooked_sys_read
0040322c        data_4568b8 = rdx_2
00403233        int64_t rdx_3 = p_sys_call_table_1[0x3b]
0040323a        p_sys_call_table_1[0x3b] = hooked_sys_execve
00403245        data_4568b0 = rdx_3
0040324c        int64_t rdx_4 = p_sys_call_table_1[0x142]
00403253        p_sys_call_table_1[0x142] = hooked_sys_execveat
0040325e        data_4568a8 = rdx_4
00403265        int64_t rdx_5 = p_sys_call_table_1[4]
```

PUMAKIT

```
do
    void* call_inst_addr = *(i + &ftrace_hook_table[0].target_func_addr)

    if (call_inst_addr == 0)
        void* real_func = *(i + &ftrace_hook_table[0].patch_location)

        if (real_func != 0)
            for (int64_t j = 5; j != 0x45; j += 1)
                if (*(real_func + j - 5) == 0xe8
                        && sx.q(*(real_func + j - 4)) + j + real_func
                        == sub_400000)
                    call_inst_addr = real_func + j
                    break
```



PUMAKIT

Hooking Techniques

```
hook->fops.func.d = rax_4

if (p_symbol != _p_symbol)
    int32_t rax_5
    rax_5.b = *p_symbol == 0xe5894855
    hook->fops.func:4.d = rax_5

hook->prologue_match_addr = _p_symbol
```

```
sub_4004c0:
004004c0  e8e3e50900          call    __fentry__
004004c5  55                  push    rbp {__saved_rbp}
004004c6  4889e5              mov     rbp, rsp {__saved_rbp}
004004c9  4883ec38            sub     rsp, 0x38
004004cd  65488b0425280000…   mov     rax, qword [gs:0x28]
004004d6  488945f8            mov     qword [rbp-0x8 {var_10}], rax
004004da  31c0                xor     eax, eax  {0x0}
004004dc  85ff                test    edi, edi
004004de  7858                js      0x400538
```

Syscall table hooking is dead (or is it..?)

Pre Version 6.9

```
#define __SYSCALL(nr, sym) extern long __x64_##sym(const struct pt_regs *);
#include <asm/syscalls_64.h>
#undef __SYSCALL

#define __SYSCALL(nr, sym) __x64_##sym,

asmlinkage const sys_call_ptr_t sys_call_table[] = {
#include <asm/syscalls_64.h>
};
```

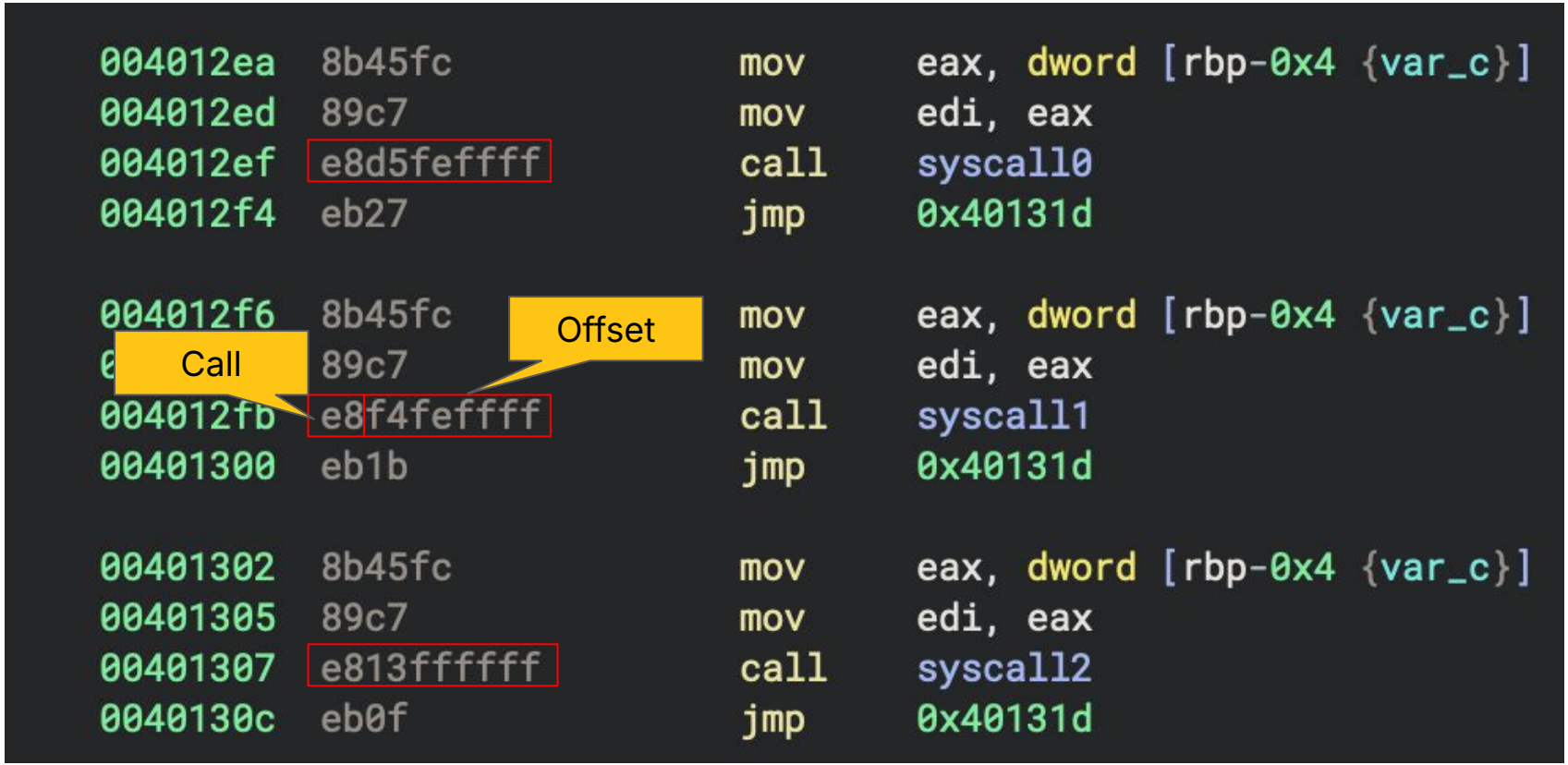
Syscall table hooking is dead (or is it..?)

Version 6.9+

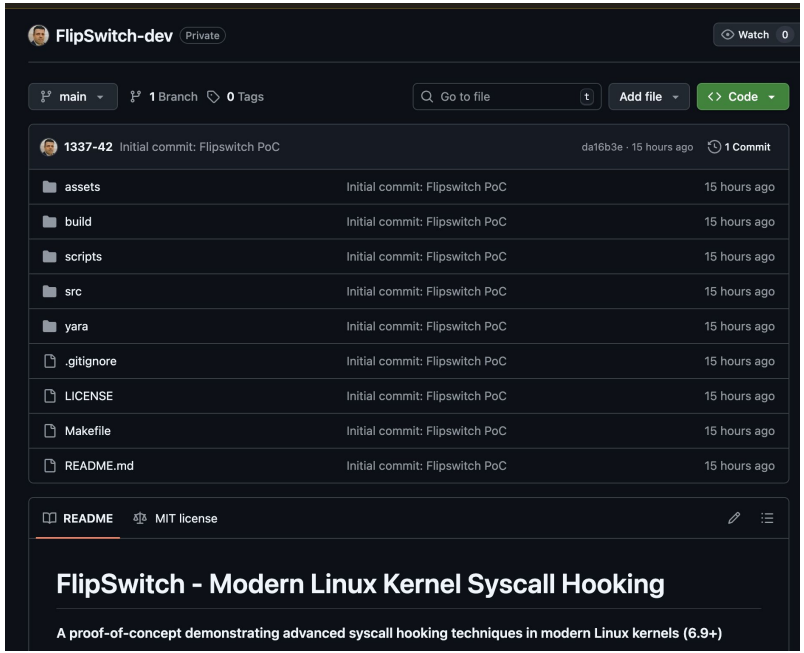
```
 * The sys_call_table[] is no longer used for system calls, but
 * kernel/trace/trace_syscalls.c still wants to know the system
 * call address.
 */
#define __SYSCALL(nr, sym) __x64_##sym,
const sys_call_ptr_t sys_call_table[] = {
#include <asm/syscalls_64.h>
};
#undef  __SYSCALL

#define __SYSCALL(nr, sym) case nr: return __x64_##sym(regs);
long x64_sys_call(const struct pt_regs *regs, unsigned int nr)
{
        switch (nr) {
        #include <asm/syscalls_64.h>
        default: return __x64_sys_ni_syscall(regs);
        }
```

Syscall table hooking is dead (or is it..?)

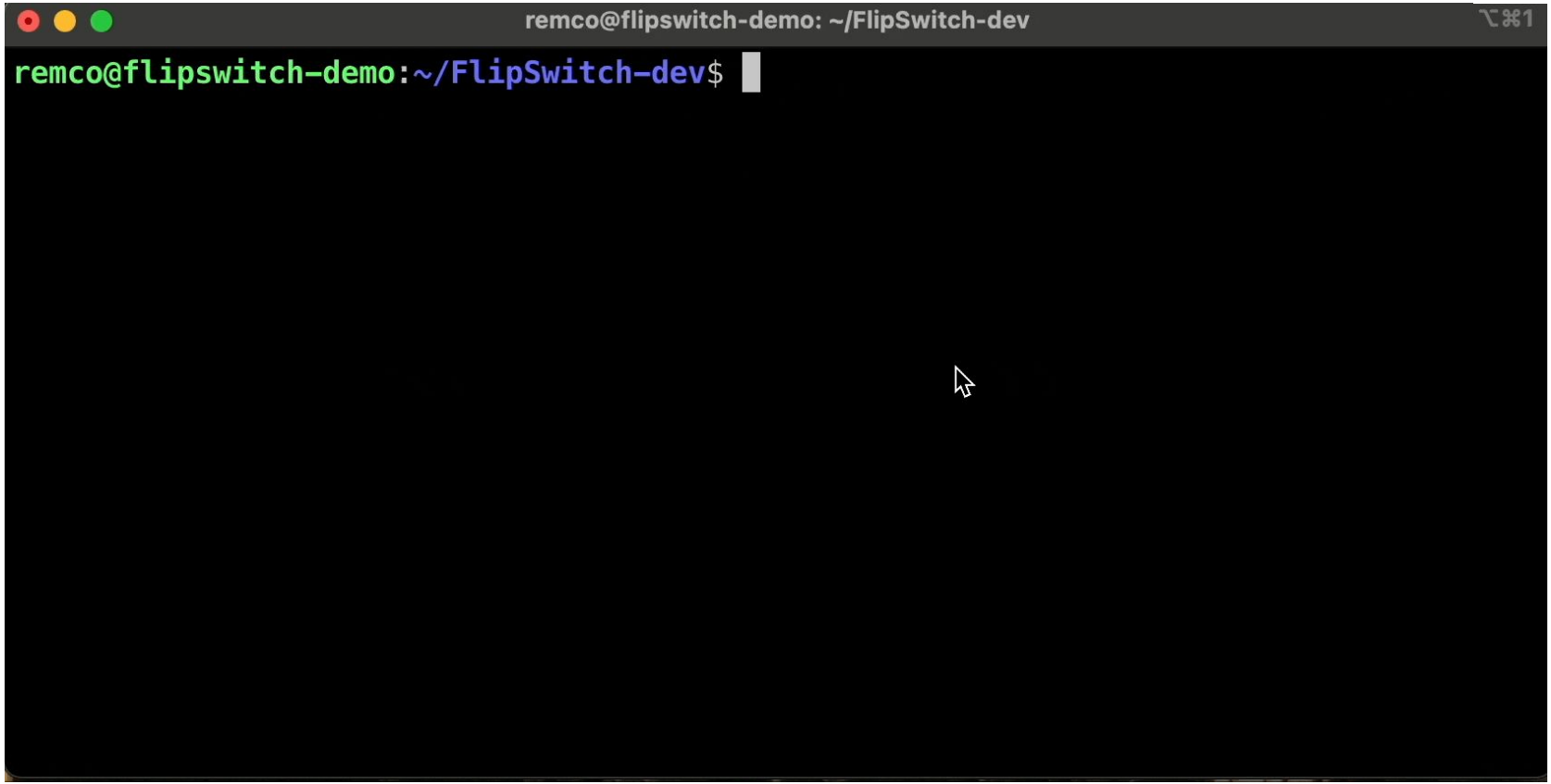


Introducing FlipSwitch

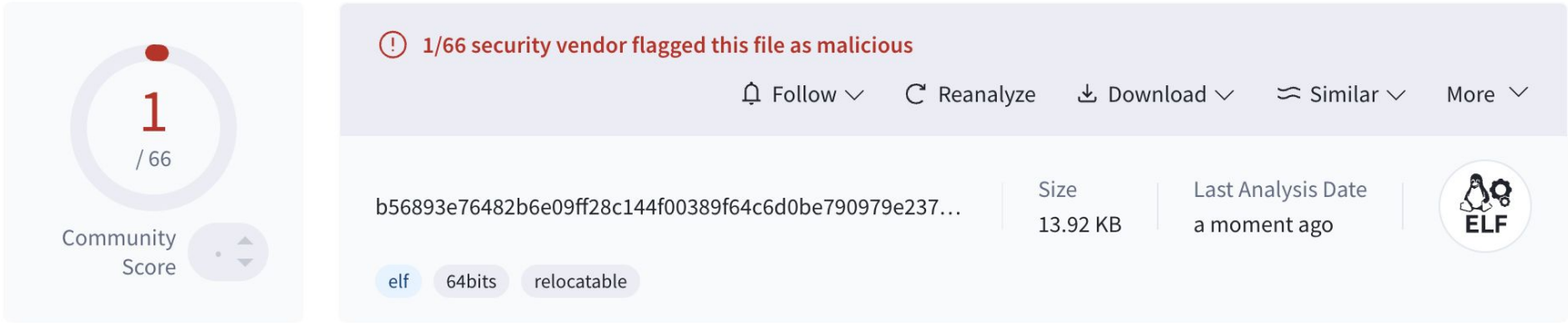


https://github.com/1337-42/FlipSwitch-dev

Introducing FlipSwitch



Syscall table hooking is dead (or is it..?)



elastic

Rootkit Detection Engineering

Behavioral Detection

- When static detection fails (and it certainly will)
- Detect the technique, rather than the threat
- Written in Elastic EQL & KQL, but easily converted
- These & more detection queries are available in the research paper

Rootkit Detection Engineering

Shared Object (Userland) Rootkits

- Creation of a shared object file on-disk
- Creation of, or addition to, an existing dynamic linker configuration file

```
file where event.action in ("creation", "rename") and
file.path like ("/etc/ld.so.preload", "/etc/ld.so.conf", "/etc/ld.so.conf.d/*")
```

- Modification of the `LD_PRELOAD` environment variable

```
process where event.type == "start" and event.action == "exec" and process.env_vars != null
```

Rootkit Detection Engineering

Loadable Kernel Module (kernel-space) Rootkits

- LKM loading
  - Utility (insmod/modprobe)
  - Loader

```
-a always,exit -F arch=b64 -S finit_module -S init_module
-a always,exit -F arch=b32 -S finit_module -S init_module
```

```
driver where event.action == "loaded-kernel-module" and auditd.data.syscall in ("init_module", "finit_module")
```

Rootkit Detection Engineering

Loadable Kernel Module (kernel-space) Rootkits

- Tainted kernel
  - Out-of-tree module → built outside kernel source tree
  - Unsigned module → lacks a valid cryptographic signature

```
[ 2853.023215] reveng_rtkit: loading out-of-tree module taints kernel.
[ 2853.023219] reveng_rtkit: module license 'unspecified' taints kernel.
[ 2853.023220] Disabling lock debugging due to kernel taint
[ 2853.023297] reveng_rtkit: module verification failed: signature and/or required key missing - tainting
kernel
```

Rootkit Detection Engineering

Loadable Kernel Module (kernel-space) Rootkits

- Unusual kill signals (32-64)
  - `kill(pid_t pid, int sig)`

```
process where event.action == "killed-pid" and auditd.data.syscall == "kill" and auditd.data.a1 in (
  "21", "22", "23", "24", "25", "26", "27", "28", "29", "2a", "2b", "2c", "2d", "2e", "2f", "30",
  "31", "32", "33", "34", "35", "36", "37", "38", "39", "3a", "3b", "3c", "3d", "3e", "3f", "40",
  "41", "42", "43", "44", "45", "46", "47"
)
```

Rootkit Detection Engineering

Loadable Kernel Module (kernel-space) Rootkits

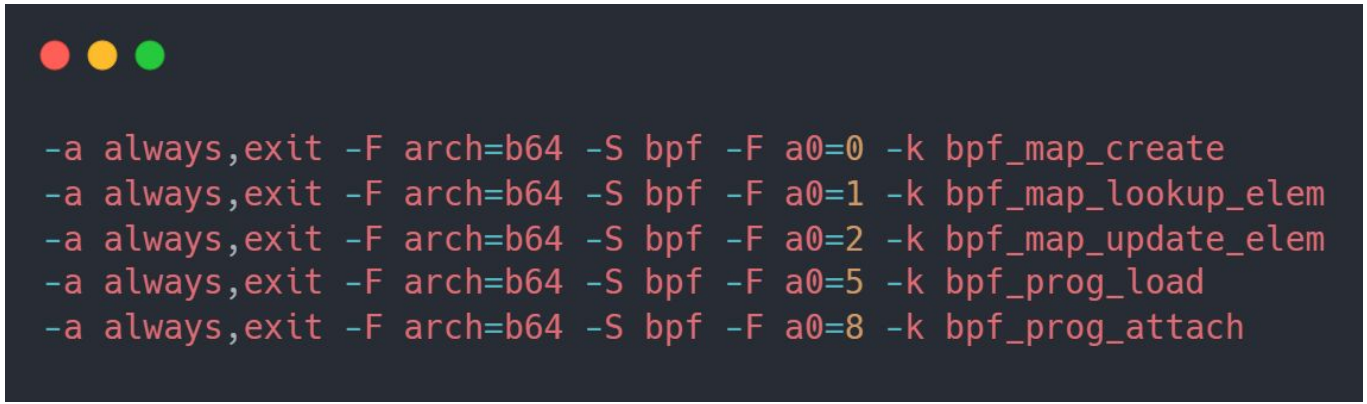
- Segfaults

```
event.dataset:"system.syslog" and process.name:"kernel" and message:"segfault"
```

Rootkit Detection Engineering

eBPF Rootkits

- `bpf(BPF_MAP_CREATE / LOOKUP_ELEM / UPDATE_ELEM, ...)`
  - Used to create, read and update maps (in-kernel key/value stores)
- `bpf(BPF_PROG_LOAD / ATTACH, ...)`
  - Loads eBPF byte code into kernel & attaches it to hooks



Rootkit Detection Engineering

eBPF Rootkits

- `bpf_probe_write_user` helper
  - Allows an eBPF program running in kernel context to write directly into user-space memory.
- Community is debating whether to restrict or remove it

```
event.dataset:"system.syslog" and process.name:"kernel" and message:"bpf_probe_write_user"
```

Detection Engineering

IO_URING rootkits

- `io_uring_setup()` → initialize submission/completion rings
- `io_uring_enter()` → execute queued I/O requests
- `io_uring_register()` → pin/register files or memory buffers

```
-a always,exit -F arch=b64 -S io_uring_setup -S io_uring_enter -S io_uring_register -k io_uring
```

Rootkit Detection Engineering

LKM Persistence Techniques

- Configuration files (`modules`, `modprobe.d/` and `modules-load.d/`)

```
file where event.action in ("rename", "creation") and file.path like (
  "/etc/modules",
  "/etc/modprobe.d/*",
  "/run/modprobe.d/*",
  "/usr/local/lib/modprobe.d/*",
  "/usr/lib/modprobe.d/*",
  "/lib/modprobe.d/*",
  "/etc/modules-load.d/*",
  "/run/modules-load.d/*",
  "/usr/local/lib/modules-load.d/*",
  "/usr/lib/modules-load.d/*"
)
```

Rootkit Detection Engineering

General Persistence Techniques

- Udev rules

```
ACTION=="add", ENV{MAJOR}=="1", ENV{MINOR}=="8", RUN+="/lib/udev/reptile"
```

```
file where event.action in ("rename", "creation") and file.extension == "rules" and file.path like (
  "/lib/udev/*",
  "/etc/udev/rules.d/*",
  "/usr/lib/udev/rules.d/*",
  "/run/udev/rules.d/*",
  "/usr/local/lib/udev/rules.d/*"
)
```

Rootkit Detection Engineering

General Persistence Techniques

- Systemd units/timers
- Cron jobs
- Initialization scripts (`init`, `init.d` and `rc.local`)
- Sudoers configuration file

Rootkit Detection Engineering

Defense Evasion Techniques

- Process masquerading
  - `kworker`, `migration`, `rcu_sched`
  - `sshd`, `systemd`, `dbus-daemon`, `bash`

```
process where event.type == "start" and event.action == "exec" and
process.parent.name like~ ("kworker*", "kthreadd") and
process.name in ("bash", "dash", "sh", "tcsh", "csh", "zsh", "ksh", "fish") and
process.args == "-c"
```

Rootkit Detection Engineering

Defense Evasion Techniques

- Log/audit file cleansing
- Clearing shell history

```
file where event.type == "deletion" and file.path in (
  "/var/log/syslog", "/var/log/messages", "/var/log/secure", "/var/log/auth.log",
  "/var/log/boot.log", "/var/log/kern.log", "/var/log/dmesg"
)
```

Rootkit Detection Engineering

Defense Evasion Techniques

- Kernel message buffer clearing
  - Dmesg
  - Journalctl

```
process where event.type == "start" and event.action == "exec" and process.name == "dmesg" and
process.args in ("-c", "--clear")
```

Rootkit Detection Engineering

Defense Evasion Techniques

- Timestomping
  - Alter timestamps to hide malicious files/activities

```
process where event.type == "start" and event.action == "exec" and process.name == "touch" and
process.args like ("-t*", "-d*", "-a*", "-m*", "-r*", "--date=*", "--reference=*", "--time=*")
```

Minimizing the Attack Surface

- Keep your kernel up-to-date
- Kernel hardening (e.g. secure boot and signature enforcement)
- Kernel protection tools such as Linux Kernel Runtime Guard (LKRG)
- Behavioral monitoring (Auditd, file integrity monitoring, EDRs)
- Access control frameworks (SELinux/AppArmor)
- Periodical active scanning (OSQuery, Lynis, rkhunter, chkrootkit)
- Minimize initial access vectors

Thank you!

Remco Sprooten

https://www.linkedin.com/in/remco-sprooten/

https://github.com/1337-42/FlipSwitch-dev

Ruben Groenewoud

https://x.com/RFGroenewoud

https://www.linkedin.com/in/ruben-groenewoud/

elastic | The Search AI Company