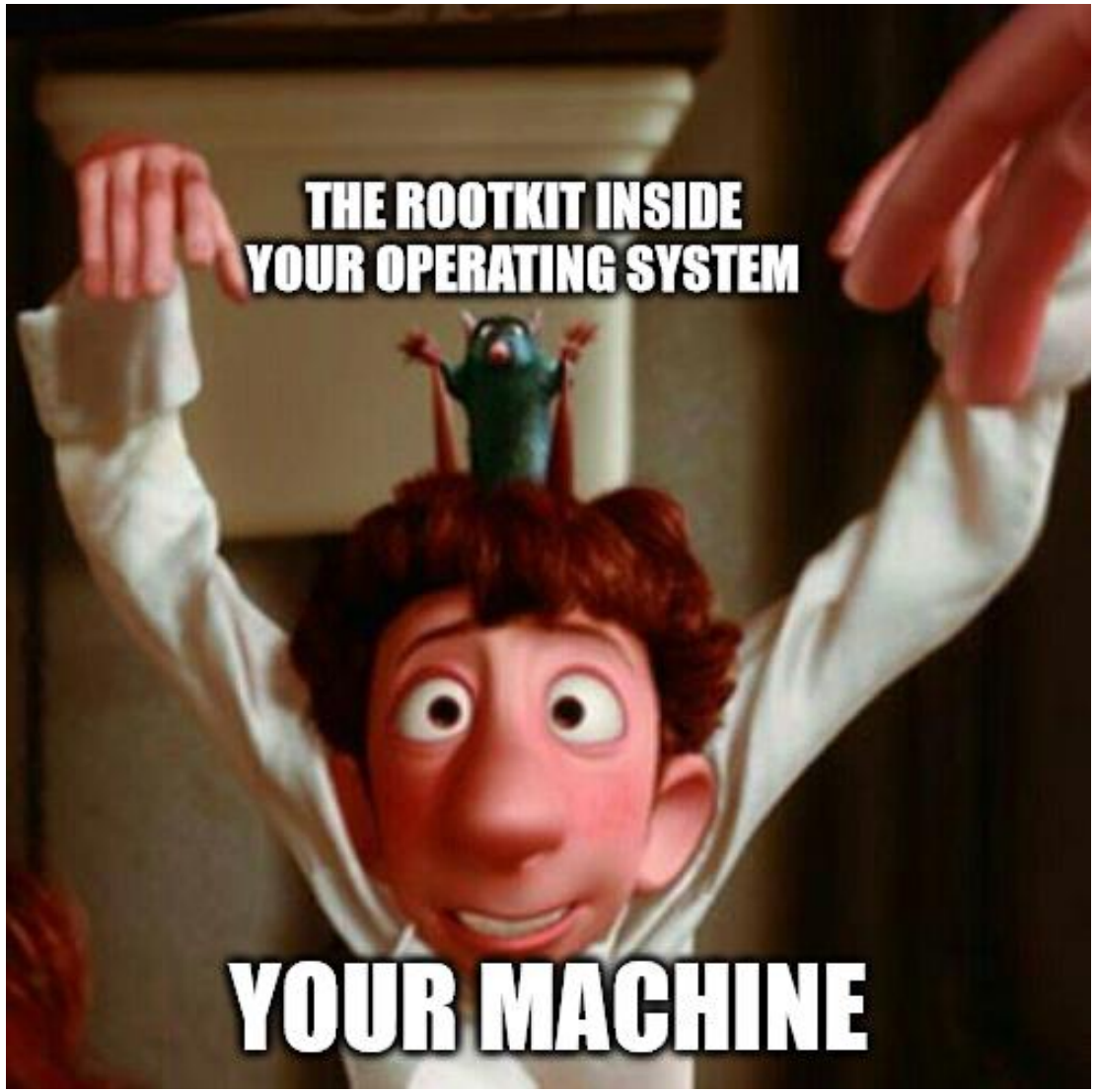
In-Depth Study of Linux Rootkits: Evolution, Detection, and Defense

MALMIUM

FIRST TC - Amsterdam

2025

This talk in a nutshell



About me


Stephan Berger (Nickname Malmö)


Head of investigations at Infoguard AG


Over 13 years into IT security, various certs, tweets and blogs about IR topics. I love Windows and Linux equally ☺


Blog: dfir.ch


Social: @malmoeb (X, Mastodon, etc.)


I will probably not win the contest for the nicest presentation, but well..

A (short) history of Linux rootkits

MALMIUM

Analysis of the KNARK Rootkit
by Toby Miller
Mar 12 2001 6:00PM GMT

Analysis of the KNARK Rootkit

Purpose

The purpose of this paper is to identify signatures related to the KNARK rootkit. This paper does no comparisons between this rootkit and other rootkits. This paper will attempt to educate the reader rootkit.

What is a rootkit?

A rootkit is a collection of files/programs used by attacker(s) to re-enter a network/computer witho various "popular" exploits to assist the attacker in the re-entry of a system. Recently, many of the found in BIND, Linux line printer, and Washington University's FTP program.

In addition to the exploits, many rootkits also come with and install sniffers. This is done because over the network; a sniffer can do this and it's quite hard to detect. A rootkit can also change com them.

Common binaries are binaries that can be used to monitor a systems operation. Some of the comr and /usr/bin/top (this is not a complete list). Now that we have covered rootkit basics, lets look at

IEEE Security and Privacy, volume 4, issue 1, pages 24-32

The t0rn rootkit

Submitted by: Paulo Braga Rodrigues Craveiro
Attended: Internet
Date Submitted: 2002/05/31

UNIX and Linux based Rootkits
Techniques and Countermeasures

Andreas Bunten
DFN-CERT Services GmbH
Heidenkampsweg 41
D-20097 Hamburg
bunten@dfn-cert.de

April 30, 2004

Abstract

A rootkit enables an attacker to stay unnoticed on a compromised system and to use it for his purposes. This paper reviews techniques currently used by attackers on UNIX and Linux systems with a focus on kernel rootkits. Example rootkits are classified according to code injection and how the flow of execution is diverted within kernel. The efficiency of different countermeasures is discussed for these examples.

Detecting and categorizing kernel-level rootkits to aid future detection

J F Levine[1], J B Grizzard[1], H L Owen[1]

Hide authors affiliations

[1] Georgia Institute of Technology, USA

Publication type: Journal Article **Publication date**: 2006-01-01

Chapter PDF Available

Rootkit Detection Mechanism: A Survey

January 2011 · Communications in Computer and Information Science 203:366-374
203:366-374
DOI:10.1007/978-3-642-24037-9_36
In book: Advances in Parallel Distributed Computing (pp.366-374) · Publisher: Springer Berlin Heidelberg · Editors: Nagamalai, Dhinaharan, Renault, Eric, Dhanuskodi, Murugan

Authors:

Jestin Joy
St. George's College Aruvithura

Anita John
Rajagiri School of Engineering and Tech…

James Joy
Tata Elxsi

Thursday, September 20, 2012

MoVP 2.4 Analyzing the Jynx rootkit and LD_PRELOAD

Month of Volatility Plugins

In this post I will analyze the Jynx rootkit using Volatility's new Linux features.

If you would like to follow along or recreate the steps taken, please see the LinuxForensicsWiki for instructions on how to do so.

Obtaining the Samples

In order to have samples to test against, I used the sample provided by SecondLook on their Linux memory images page, and I also loaded the Jynx2 rootkit against a running netcat process in my Debian virtual machine that was running the 2.6.32-5-686 32-bit kernel. I then acquired a memory capture of my VM using LiME.

Userland Rootkits

Symbiote: Instead of being a standalone executable file that is run to infect a machine, it is a shared object (SO) library that is loaded into all running processes using LD_PRELOAD, and parasitically infects the machine. [Intezer Blog, 2022]

Orbit: To install the payload and add it to the shared libraries that are being loaded by the dynamic linker, the dropper calls a function called patch_ld. [Intezer Blog, 2022]

Using an open-source rootkit, bedevil (bdvl) to target VMware vCenter servers. [Unit42 Blog, 2024]

Further searching revealed that the threat actor reuses the publicly available beurk rootkit, but with several custom modifications. [TrendMicro Blog, 2020]

The library used to hide Winnti's system activity is a copy of the open-source userland rootkit Azazel, with minor changes.
[Chronicle Blog, 2019]

HiddenWasp authors have adopted a large amount of code from various publicly available open-source malware, such as Mirai and the Azazel rootkit. [Intezer Blog, 2019]

Azazel rootkit is an open-source rootkit that targets older Linux kernels. Azazel is based on the LD_PRELOAD technique.
[Unit42 Blog, 2023]

Kernelspace Rootkits

In early March 2024, we found a new Diamorphine variant undetected in-the-wild. [Decoded - Avast, 2024]

The Kiss-a-dog campaign uses the Diamorphine and libprocesshide rootkits to hide the process from the user space, where the typical cloud practitioner will look for malicious activities. Both rootkits are known to hide processes from the user. [Crowdstrike, 2022]

To achieve persistent access on the FortiManager device, the threat actor deployed a backdoor with the filename /bin/klogd that Mandiant refers to as REPTILE, a variant of a publicly available Linux kernel module (LKM) rootkit. [Mandiant, 2023]

On supported systems, the backdoor downloads, compiles, and installs two open-source rootkits available on GitHub, Diamorphine and Reptile. [Microsoft, 2023]

Mandiant observed the actor use two publicly available rootkits, REPTILE and MEDUSA, on the guest virtual machines to maintain access and evade detection. [Mandiant, 2024]

Command execution and persistence using a combination of rootkits and utilities, including REPTILE and MEDUSA with SEAELF loader, and BUSYBOX. [Mandiant, 2025]

Mandiant has observed evidence that an activity cluster potentially related to APT41 used BPFDOOR to target South Asian government organizations and a Chinese multinational corporation. [Mandiant, 2022]

Application-Level Rootkits

Application-Level rootkits

- The t0rn rookit swaps several key system binaries, amongst them ps, ls, netstat and top
- A SANS GCIH Practical Assignment paper from 2002 discussed t0rn
- Still a thing today..

"In our attacks the malware dropped crontab, lsof, ldd and top. These tweaked binaries will hide malicious activities, in case someone is using them."

perfctl: A Stealthy Malware Targeting Millions of Linux Servers, Aqua Blog, October 2024

Application-Level rootkits

MALMIUM

I've written a blog post about the bedevil rootkit, which patches the dynamic loader from Linux:

> The command `rpm –V glibc` is used to verify the integrity of the installed glibc package on an RPM-based Linux system (like RHEL, CentOS, or Fedora). The -V (or –verify) option tells RPM to check the package's files against the metadata in the RPM database. if there are no changes, there will be no output.

```
# rpm –V glibc
#
```

The output of the command after the infection:

```
# rpm –V glibc
..5....T.    /lib64/ld-2.17.so
```

Userland Rootkits

LD_PRELOAD

LD_PRELOAD

Amateurs tend to use this technique to create little userland rootkits that

hijack glibc functions.

This is because the preloaded library will take precedence over any of the other shared libraries, so if you name your functions the same as a glibc function such as open() or write() (which are wrappers for syscalls), then your preloaded libraries' version of the functions will execute and not the real open() and write().

This is a cheap and dirty way to hijack glibc functions and
should not be used if an attacker wishes to remain stealthy.

Learning Linux Binary Analysis, Ryan "elfmaster" O'Neill

LD_PRELOAD

MALMIUM

Current landscape

MALMIUM

Name	Year	FH	PH	NH	LPE	Backdoor	Hook	Persistence
Jynx2*	2012	✓	✓	✓	✓	Accept	LD_PRELOAD	`ld.so.preload`
Azazel	2014	✓	✓	✓	✓	Accept, PAM	LD_PRELOAD	`ld.so.preload`
BEURK	2015	✓	✓	✓	✗	Accept	LD_PRELOAD	`ld.so.preload`
Zendar	2015	✓	✗	✗	✓	`/etc/passwd`	LD_PRELOAD	`ld.so.preload`
Umbreon	2016	✓	✓	✓	✓	PAM, ICMP	LD_PRELOAD	`ld.so.preload`
bedevil	2019	✓	✓	✓	✓	Accept, ICMP, PAM	LD_PRELOAD	New preload file
Father	2020	✓	✓	✓	✓	Accept	LD_PRELOAD	`ld.so.preload`
SuckIT*	2001	✓	✓	✓	✓	New user	Interrupt descriptor table	?
Diamorphine	2013	✓	✓	✗	✓	✗	System call table	?
Knark*	2013	✓	✓	✓	✓	UDP-Receive	System call table	?
adore-ng*	2014	✓	✓	✓	✓	✗	VFS handlers	?
Puszek	2016	✓	✓	✓	✗	✗	System call table	?
Reptile	2017	✓	✓	✓	✓	IP-Receive	KHOOK	Udev rule
LilyOfTheValley	2017	✓	✓	✗	✓	✗	Jump hooks in VFS handlers	?
Nuk3Gh0st	2018	✓	✓	✓	✓	User space process	Jump hooks in system calls and VFS handlers	SysVinit, Systemd, Upstart
Honey Pot Bears	2019	✓	✓	✗	✓	New user	System call table	?
Umbra	2021	✓	✗	✗	✓	Netfilter	Ftrace callbacks	?
TripleCross	2021	✓	✗	✓	✓	XDP	eBPF	Cronjob
Reveng_rtkit	2022	(✓)	✓	✗	✓	✗	System call table	?
Boopkit	2022	(✓)	✓	✗	✓	XDP	eBPF	?
Drovorub*	N.A.	✓	✓	✓	✓	WebSockets	N.A.	`/etc/modules`

Source: The Hidden Threat: Analysis of Linux Rootkit Techniques and Limitations of Current Detection Tools, Fraunhofer Institute for Communication, Information Processing and Ergonomics

Hooking the write function

MALMIUM

```c
// Original write function pointer
ssize_t (*original_write)(int fd, const void *buf, size_t
count);

// Replacement keyword and substitute
const char *target = "rocks!";
const char *replacement = "sucks!"

ssize_t write(int fd, const void *buf, size_t count) {
    // Dynamically load the original write() function
    if (!original_write) {
        original_write = dlsym(RTLD_NEXT, "write");
    }

    // Create a buffer to store the modified output
    char modified_buf[count + 1];
    memcpy(modified_buf, buf, count);
    modified_buf[count] = '\0';  // Ensure null-terminated
string

    // Find and replace the target keyword
    char *pos = strstr(modified_buf, target);
    if (pos) {
        // Overwrite the target with the replacement
        size_t offset = pos - modified_buf;
        memcpy(&modified_buf[offset], replacement,
strlen(replacement));
    }

    // Call the original write() with the modified buffer
    return original_write(fd, modified_buf, count);
}
```

LD_PRELOAD in action

MALMIUM

```
root@first:~# cat statement.txt
DFIR rocks!
```

```
root@first:~# LD_PRELOAD=./keyword_swap.so cat statement.txt
DFIR sucks!
```

ldd (List Dynamic Dependencies)


```
root@first:~# export LD_PRELOAD=/root/keyword_swap.so

root@first:~# ldd /usr/bin/cat
linux-vdso.so.1 (0x00007fff5b50f000)
/root/keyword_swap.so (0x0000790e573dc000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x0000790e57000000)
/lib64/ld-linux-x86-64.so.2 (0x0000790e573ee000)
```

Father Rootkit: Function Hooks

MALMIUM

```
63  // functions hooks (interceptions)
64  extern int __lxstat(int version, const char *path, struct stat *buf);
65  extern int __lxstat64(int version, const char *path, struct stat64 *buf);
66  extern int lstat(const char * path, struct stat * buf);
67  extern int fstat(int filedes, struct stat *buf);
68  extern int access(const char * pathname, int mode);
69  extern int open(const char *pathname, int flags, mode_t mode);
70  extern int open64(const char *pathname, int flags, mode_t mode);
71  extern int openat(int dirfd, const char * pathname, int flags);
72  extern struct dirent * readdir(DIR *p);
73  extern int unlink(const char *pathname);
74  extern int unlinkat(int dirfd, const char * pathname, int flags);
75  extern int getsockname(int socket, struct sockaddr * addr, socklen_t * addrlen);
76  extern FILE * fopen(const char * pathname, const char *mode);
77  extern FILE * fopen64(const char * pathname, const char * mode);
78  extern int accept(int sockfd, struct sockaddr * addr, socklen_t * addrlen);
79  extern DIR * opendir(const char *name);
80  extern int execve(const char *path, char *const argv[], char *const envp[]);
81  extern int pam_get_item(const pam_handle_t * pamh, int item_type, const void ** item);
82  extern int pam_authenticate(pam_handle_t * pamh, int flags);
83  //extern long ptrace(enum __ptrace_request request, pid_t pid, void * addr, void * data);
84  //extern long ptrace(enum __ptrace_request request, ...);
85  extern gcry_error_t gcry_pk_verify(gcry_sexp_t sig, gcry_sexp_t data, gcry_sexp_t pkey);
```

Source: https://github.com/mav8557/Father/blob/4eb2712caf612a7dc55fd4f34ff5c72b74c7c332/src/father.h

libprocesshider

MALMIUM

github.com/gianlucaborello/libprocesshider/tree/master

README.md	Update README.md	9 years ago
evil_script.py	changes	10 years ago
processhider.c	Fixed issue with readdir64	5 years ago

README

libprocesshider

Hide a process under Linux using the ld preloader.

Full tutorial available at https://sysdigcloud.com/hiding-linux-processes-for-fun-and-profit/

In short, compile the library:

```
gianluca@sid:~/libprocesshider$ make
gcc -Wall -fPIC -shared -o libprocesshider.so processhider.c -ldl
gianluca@sid:~/libprocesshider$ sudo mv libprocesshider.so /usr/local/lib/
```

Load it with the global dynamic linker

```
root@sid:~# echo /usr/local/lib/libprocesshider.so >> /etc/ld.so.preload
```

1k stars
26 watching
316 forks
Report repository

Releases

No releases published

Packages

No packages published

Contributors 2

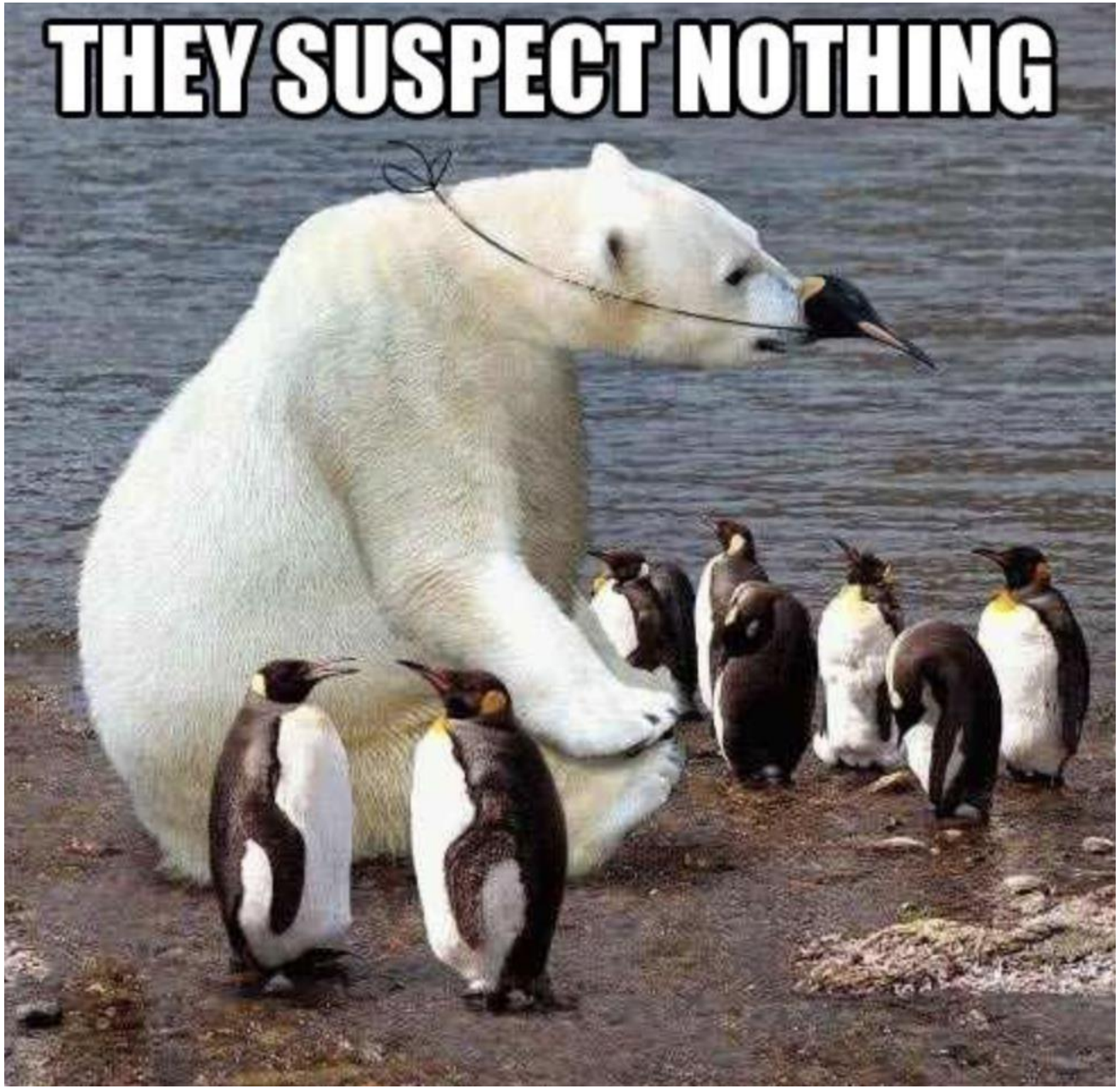
gianlucaborello Gianluca Borello
in7egral Vladimir Putin

Languages

C 85.8% Python 10.1%
Makefile 4.1%

Userland Rootkits

Detection


THEY SUSPECT NOTHING

Audit

MALMIUM

- Audit
  - /etc/ld.so.preload:
  - /etc/ld.so.conf
  - /etc/ld.so.conf.d/*.conf

- Periodically check
  /proc/{pid}/environ

- Has anyone tampered with my loader? (blog post on dfir.ch)

```
# rpm -V glibc
 # same command available for other distros
```

unhide

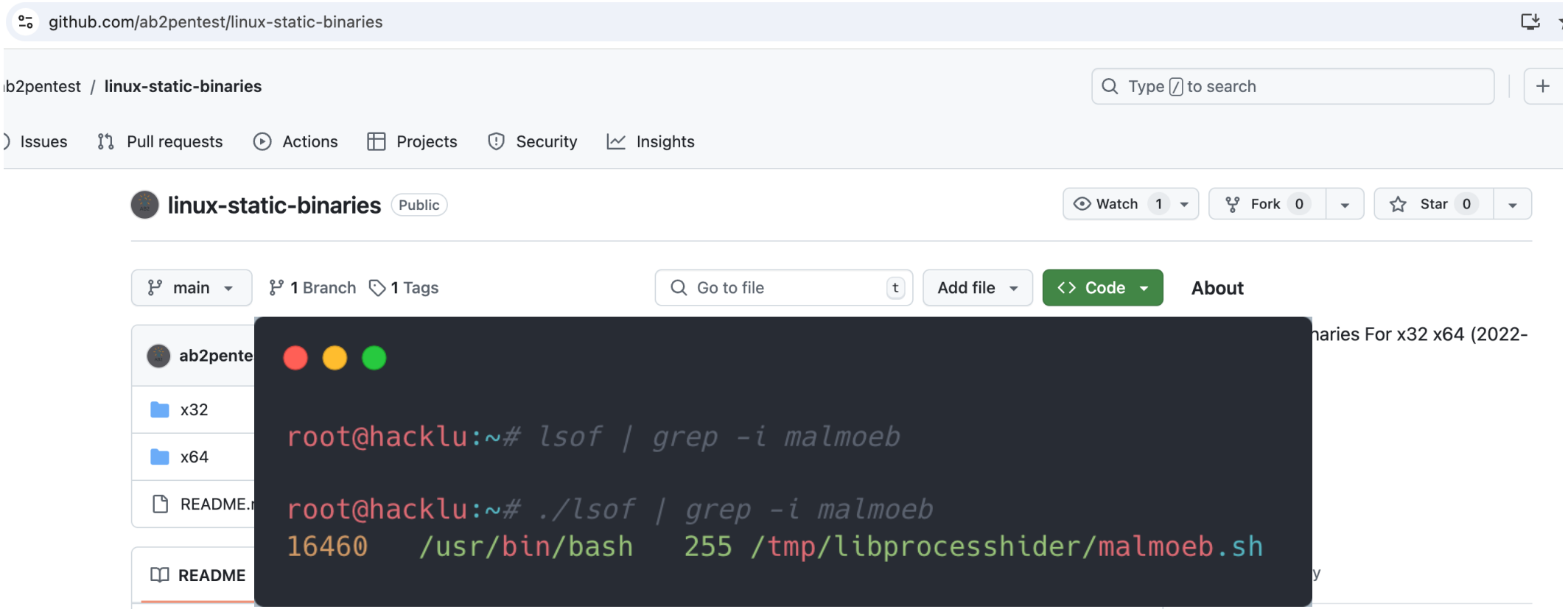
MALMIUM

github.com/YJesus/Unhide

README | GPL-3.0 license | GPL-3.0 license

```
Unhide is a forensic tool to find hidden processes and TCP/UDP ports by rootkits / LKMs
or by another hiding technique.

// Unhide (unhide-linux or unhide-posix)
// -------------------------------------

Detecting hidden processes. Implements six main techniques

1- Compare /proc vs /bin/ps output

2- Compare info gathered from /bin/ps with info gathered by walking thru the procfs. ONLY for unhide-linux version

3- Compare info gathered from /bin/ps with info gathered from syscalls (syscall scanning).

4- Full PIDs space occupation (PIDs bruteforcing). ONLY for unhide-linux version

5- Compare /bin/ps output vs /proc, procfs walking and syscall. ONLY for unhide-linux version
   Reverse search, verify that all thread seen by ps are also seen in the kernel.

6- Quick compare /proc, procfs walking and syscall vs /bin/ps output. ONLY for unhide-linux version
  It's about 20 times faster than tests 1+2+3 but maybe give more false positives.
```

Static binaries



Kernelspace Rootkits

Loadable Kernel Modules

Loadable Kernel Modules (LKM)

Another level of rootkit installations are made through Loadable Kernel Modules (LKM).

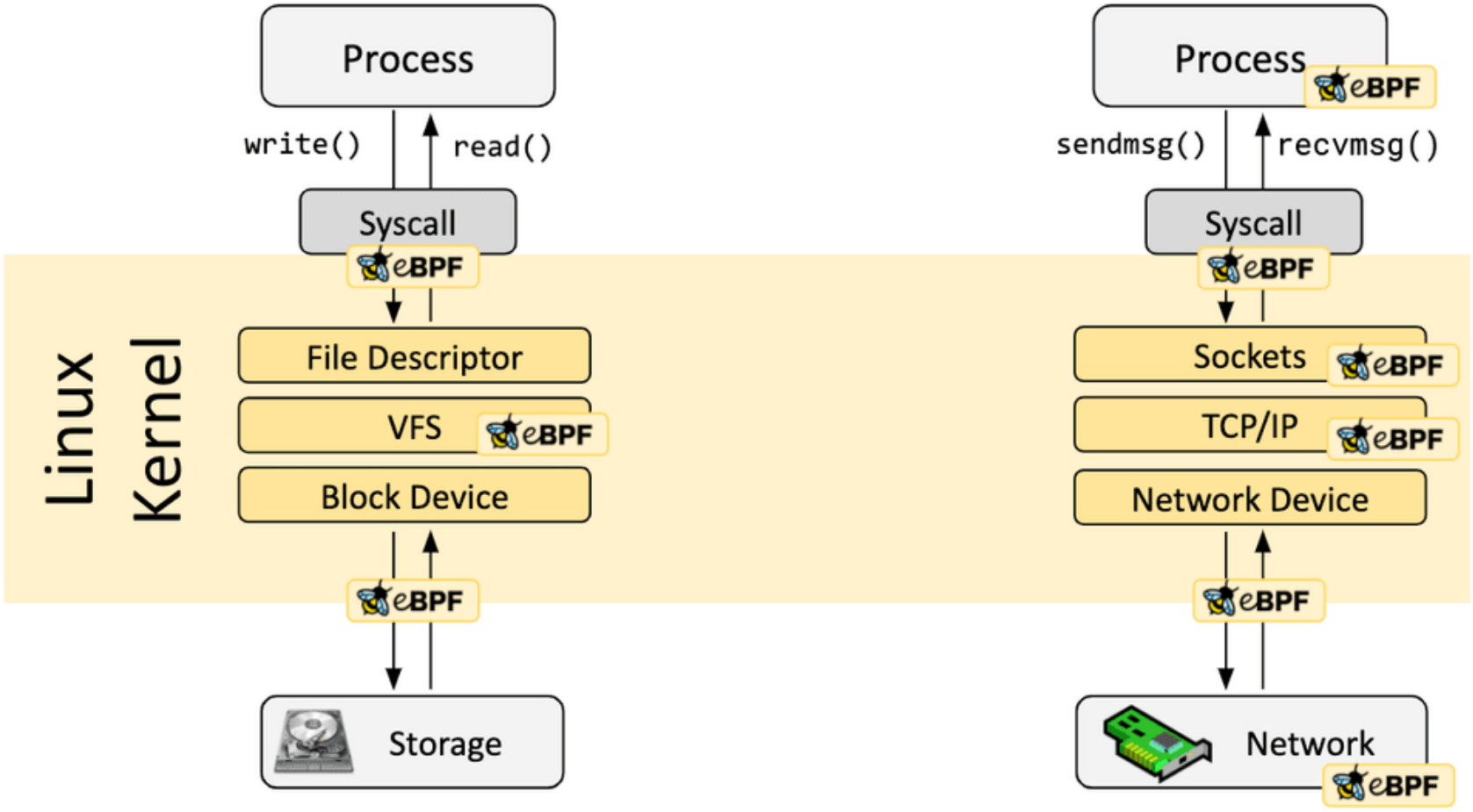
Basically, almost every modern Unix flavour (Linux, Solaris and FreeBSD) allows the system administrators to load device drivers on the fly into the kernel, avoiding the necessity of kernel recompilation and reboot of the systems.

The t0rn rootkit, SANS Institute, 2002

Kernelspace Rootkits

eBPF

extended Berkley Packet Filter (eBPF)



Source: https://ebpf.io/what-is-ebpf/

Kernelspace Rootkits

Hooking Techniques (Example)

Ftrace – Umbra

```
// Syscalls to be hooked using frace

struct ftrace_hook hooks[] = {
    //HOOK("sys_mkdir", hook_mkdir, &orig_mkdir),
    HOOK("sys_kill", hook_kill, &orig_kill),
    HOOK("sys_getdents64", hook_getdents64, &orig_getdents64),
    HOOK("sys_getdents", hook_getdents, &orig_getdents)
};
```

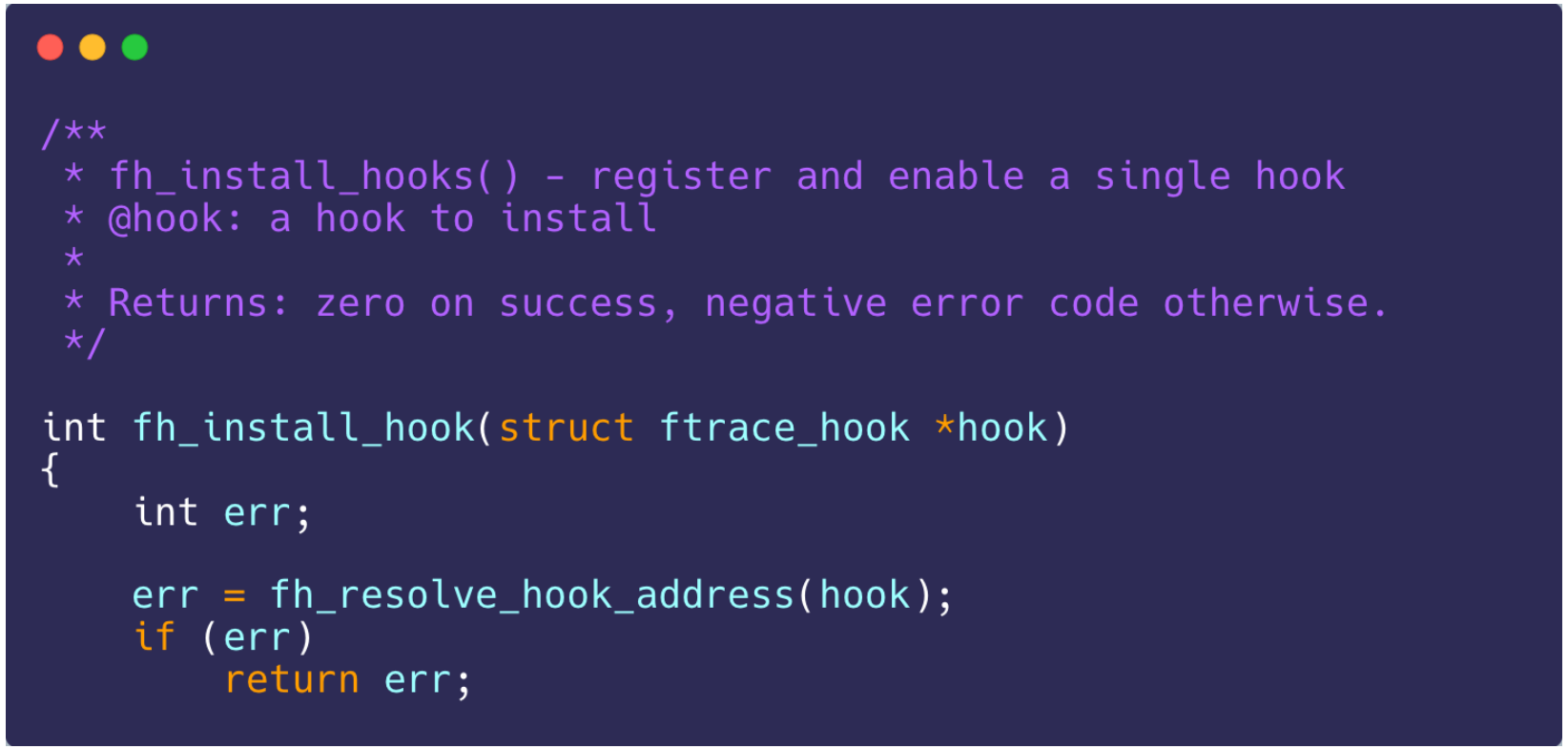
Source: https://github.com/h3xduck/Umbra/blob/96443367bb35650d8b468a9e14557f01eeb26653/kernel/src/hookers.c

Ftrace – Umbra

```
// Check if directory/file contains the umbra prefix

    if(memcmp(UMBRA_DIRECTORY_PREFIX, current_dir->d_name,
    strlen(UMBRA_DIRECTORY_PREFIX))==0){
            //Special case directory/file is the first
            if(current_dir == dirent_ker){
                ret -= current_dir->d_reclen;
                memmove(current_dir, (void *)current_dir +
current_dir->d_reclen, ret);
                continue;
            }

// We hide this entry by incrementing the rec_len field, now
pointing to the next entry

    prev_dir->d_reclen += current_dir->d_reclen;

}else{

    prev_dir = current_dir;
```

Source: https://github.com/h3xduck/Umbra/blob/96443367bb35650d8b468a9e14557f01eeb26653/kernel/src/hookers.c

Ftrace – Umbra



```
/**
 * fh_install_hooks() - register and enable a single hook
 * @hook: a hook to install
 *
 * Returns: zero on success, negative error code otherwise.
 */

int fh_install_hook(struct ftrace_hook *hook)
{
    int err;

    err = fh_resolve_hook_address(hook);
    if (err)
        return err;
```

Kernelspace Rootkits

Detection

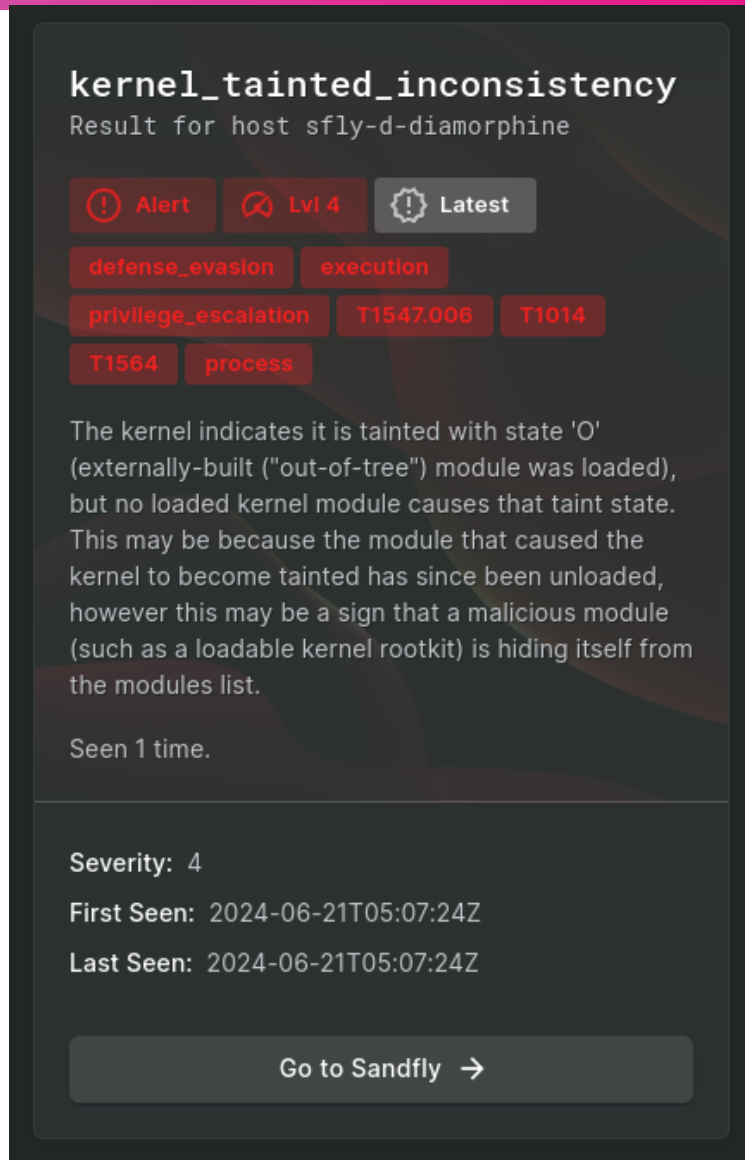
Kernel Tainting

MALMIUM

root@first:~# dmesg
[334.056326] dfir: loading out-of-tree module taints kernel.
[334.056343] dfir: module verification failed: signature and/or required key missing - tainting kernel

root@first:~# cat /proc/modules | grep OE
dfir 12288 0 - Live 0xffffffffc0c52000 (OE)

OE = Out-of-tree & unsigned (blog post on dfir.ch)



Diamorphine Rootkit (FTRACE)

root@first:~# cat /sys/kernel/tracing/available_filter_functions
| grep diamorphine

is_invisible.part.1 [diamorphine]

hacked_getdents [diamorphine]

hacked_getdents64 [diamorphine]

get_syscall_table_bf [diamorphine]

find_task [diamorphine]

is_invisible [diamorphine]

give_root [diamorphine]

[…]

ebpfkit-monitor

ebpfkit-monitor

License GPL v2 License Apache 2.0

`ebpfkit-monitor` is an utility that you can use to statically analyse eBPF bytecode or monitor suspicious eBPF activity at runtime. It was specifically designed to detect `ebpfkit`.

ebpfkit

License GPL v2 License Apache 2.0

`ebpfkit` is a rootkit that leverages multiple eBPF features to implement offensive security techniques. We implemented most of the features you would expect from a rootkit: obfuscation techniques, container breakouts, persistent access, command and control, pivoting, network scanning, Runtime Application Self-Protection (RASP) bypass, etc.

Source: https://github.com/Gui774ume/ebpfkit-monitor

Linux Kernel Runtime Guard (LKRG)

MALMIUM

Linux Kernel Runtime Guard (LKRG) - Linux Kernel Runtime Integrity Checking and Exploit

Development

- Topics
- More
- Categories
  - News
  - Support
  - Qubes-Whonix
  - KVM
  - VirtualBox
  - All categories
- Tags
  - status_closed_issue_impl…

Date: Sun, 14 Jun 2020 17:37:59 +0200
From: Solar Designer <solar@…nwall.com>
To: lkrg-users@…ts.openwall.com
Subject: rootkit detection

Hi,

Adam found this interesting Master's Thesis of Juho Junnila, entitled "Effectiveness of Linux Rootkit Detection Tools":

http://jultika.oulu.fi/files/nbnfioulu-202004201485.pdf 1

It shows LKRG as the most effective kernel rootkit detector (of those tested), as long as LKRG is loaded before the rootkit. It also shows LKRG sometimes detect rootkits even when LKRG is loaded after the rootkit (we never intended that to work, but it sometimes does anyway, which is OK). Finally, it shows that LKRG never detects pure userspace rootkits (as expected; it would probably be a bug if it did).

Source: https://www.openwall.com/lists/lkrg-users/2020/06/14/5

Linux Kernel Runtime Guard (LKRG)

In particular, when LKRG is loaded before the rootkit, it detected 8 out of 9 kernel rootkits tested:

- Diamorphine ✅
- Honey Pot Bears
- LilyOfTheValley
- Nuk3 Gh0st
- Puszek
- Reptile ✅
- Rootfoo Linux Rootkit,
- Sutekh.

There were no false positives.

Source: https://oulurepo.oulu.fi/bitstream/handle/10024/15182/nbnfioulu-202004201485.pdf

Linux Kernel Runtime Guard (LKRG)

MALMIUM

wget https://lkrg.org/download/lkrg-0.9.5.tar.gz

insmod ./lkrg.ko kint_enforce=0

```
[  190.959830] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  190.959888] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  203.322121] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  203.322171] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  218.608012] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  218.608071] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  233.969168] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  233.969220] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  249.334367] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  249.334425] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  264.692390] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  264.692443] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  280.049979] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  280.050042] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  295.407691] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  295.407749] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  310.764484] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  310.764535] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  326.123190] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  326.123255] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  341.484515] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  341.485576] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
[  356.847823] LKRG: ALERT: DETECT: Kernel: Found 1 fewer modules in module list (58) than in KOBJ (59), maybe hidden module name reptile_module
[  356.848875] LKRG: ALERT: DETECT: Kernel: 1 checksums changed unexpectedly
```

bpf_probe_write_user


Using bpf_probe_write_user anything that gets sent or received as a userspace buffer can be altered.

grep bpf /var/log/messages

[..]

Oct 7 13:05:23 dfir kernel: $programm[6262] is installing a program with bpf_probe_write_user helper that may corrupt user memory!

dmesg

[440.029740] $programm[6262] is installing a program with bpf_probe_write_user helper that may corrupt user memory!

bpf_probe_write_user

MALMIUM

repo:elastic/protections-artifacts bpf_probe_write_user

Filter by

- Code 0
- Issues 0
- Pull requests 0
- Discussions 0
- Commits 0
- Packages 0
- Wikis 0

Advanced search

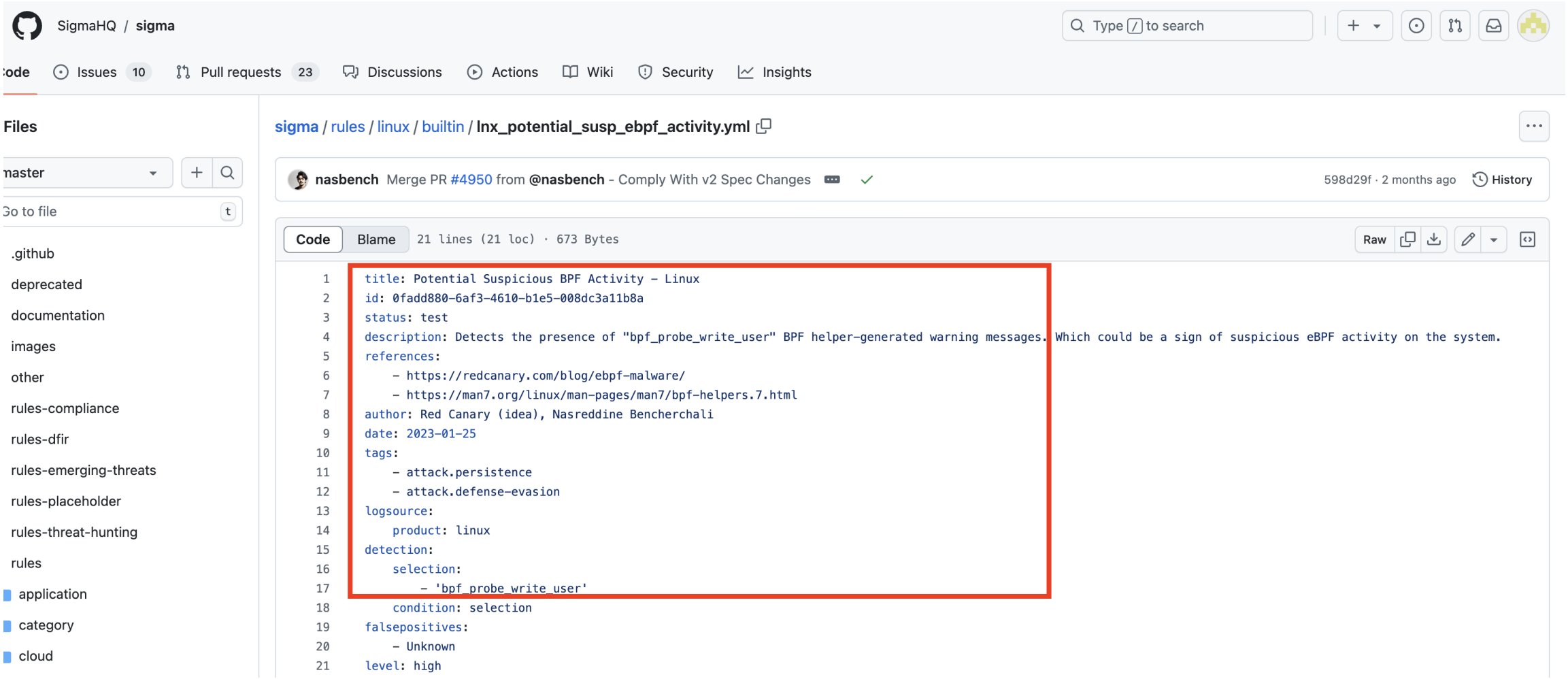
0 files (40 ms) in elastic/protections-artifacts

Your search did not match any code

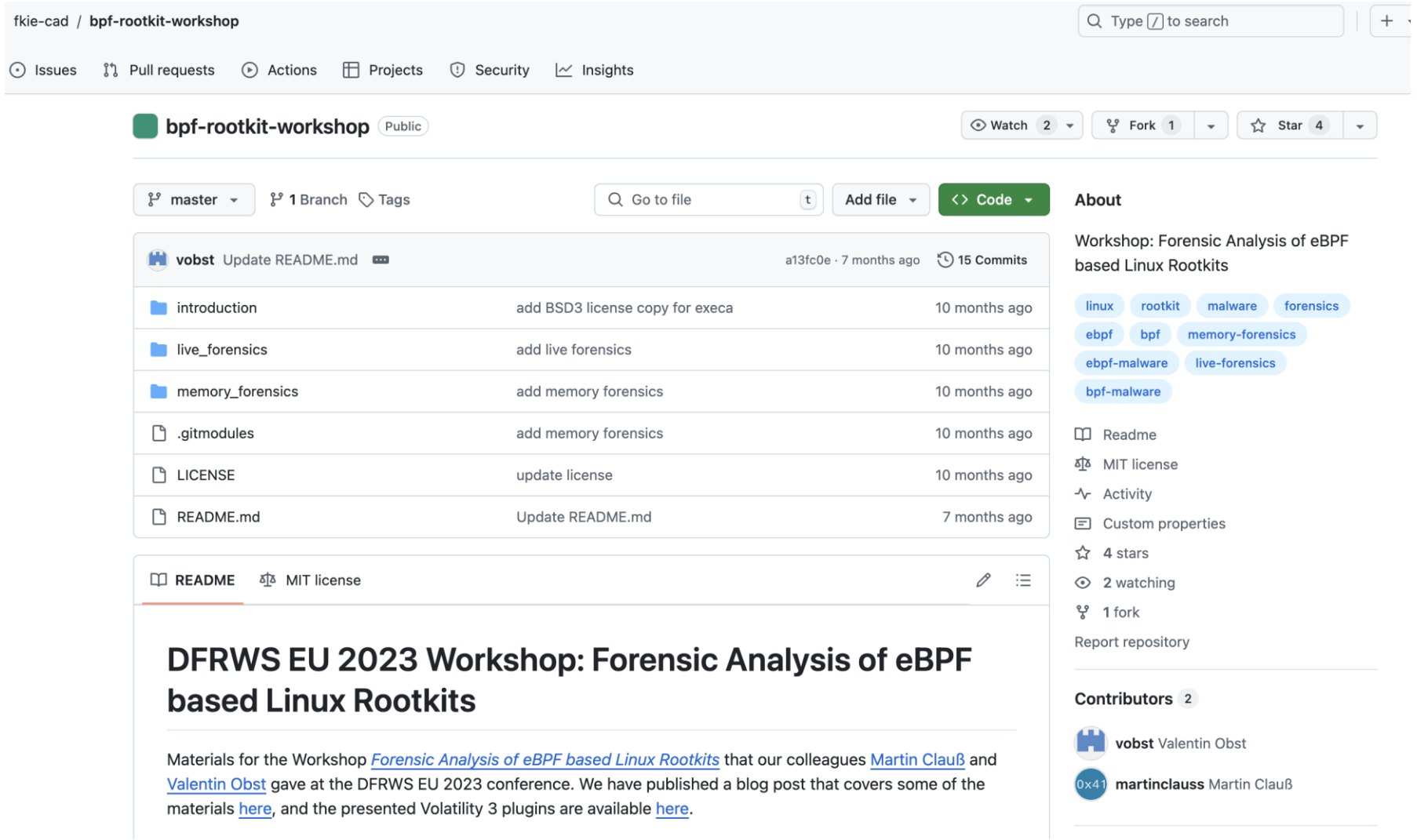
You could try one of the tips below.

- Search across repositories
- Search across an organization
- Find a particular file extension
- Why wasn't my code found?
- Regular expressions
- Saved searches

bpf_probe_write_user

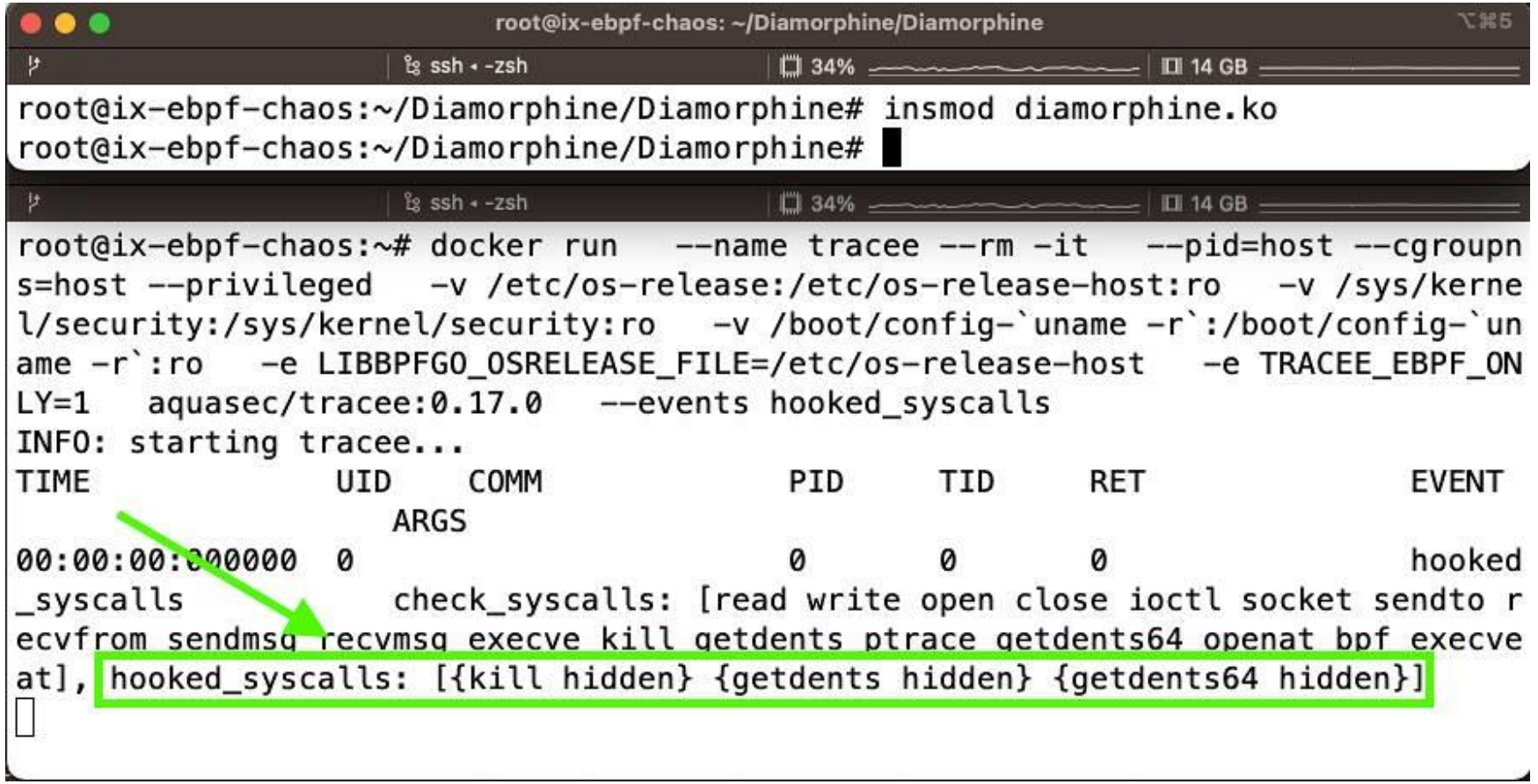


Memory Forensics



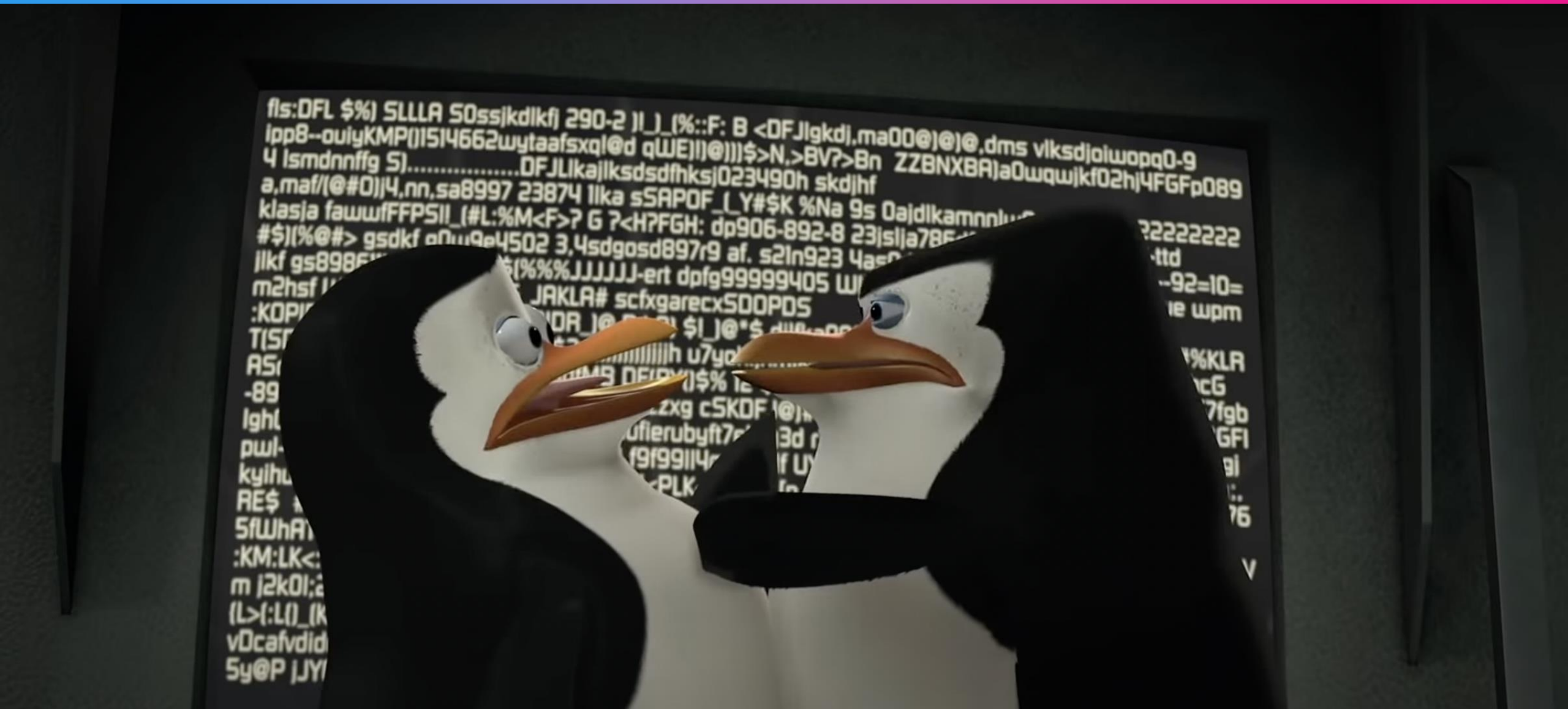
Overview of Tools

- rkhunter
- chkrootkit
- Wazuh
- Tracee



Questions?

MALMIUM



Userland Rootkits

Patched Loader

ITW: bedevil

MALMIUM

bdvl / inc / util / install / ldpatch / **ldpatch.h**

jrowa2 brief ldpatch update...

Code | Blame | 11 lines (8 loc) · 488 Bytes | Code 55% faster with GitHub Copilot | Raw

```
1   #define MAXLDS 10  // the maximum number of ld.so paths that can be found/stored. more than enough?
2
3   static char *const ldhomes[7] = {"/lib", "/lib32", "/lib64", "/libx32",
4                                    "/lib/x86_64-linux-gnu", "/lib/i386-linux-gnu", "/lib/arm-linux-gnueabihf"};
5
6   char **ldfind(int *allf);
7   #include "find.c"
8
9   int _ldpatch(const char *path, const char *oldpreload, const char *newpreload);
10  int ldpatch(const char *oldpreload, const char *newpreload);
11  #include "patch.c"
```

Patched loader


```
root@first:~# ls -l /lib64/ld-linux-x86-64.so.2
[..] /lib64/ld-linux-x86-64.so.2 -> ../lib/x86_64-linux-gnu/ld-linux-x86-64.so.2

root@first:~# perl -pi -e 's/ld.so.preload/ld.so.malmoeb/g'
/root/ld-linux-x86-64.so.2

root@first:~# ls -l /lib64/ld-linux-x86-64.so.2
[..] /lib64/ld-linux-x86-64.so.2 ->
/root/ld-linux-x86-64.so.2

root@first:~# cat /etc/ld.so.malmoeb
/root/dfir.so

root@first:~# cat statement.txt
DFIR sucks!
```

Userland Rootkits

More detection strategies

/proc


```
root@first_tc:~# ps aux | grep vi
root      45560  0.3  1.3  25528 13312 pts/0    Sl+  09:03   0:00 vi statement.txt

root@first_tc:~# cat /proc/45560/maps | grep swap
72cb0b9fe000-72cb0b9ff000 r--p 00000000 fd:01 1623 /root/keyword_swap.so
72cb0b9ff000-72cb0ba00000 r-xp 00001000 fd:01 1623 /root/keyword_swap.so
72cb0ba00000-72cb0ba01000 r--p 00002000 fd:01 1623 /root/keyword_swap.so
72cb0ba01000-72cb0ba02000 r--p 00002000 fd:01 1623 /root/keyword_swap.so
72cb0ba02000-72cb0ba03000 rw-p 00003000 fd:01 1623 /root/keyword_swap.so

root@first_tc:~# cat /proc/45560/environ
SHELL=/bin/bashPWD=/rootLOGNAME=rootXDG_SESSION_TYPE=ttyLD_PRELOAD=/root/keyword_swap.so
```

Father Rootkit – Now you see me


```
root@first_tc:~/Father# cat src/config.h
[..]
/* name to hide for files */
#define PRELOAD "ld.so.preload" // used for hiding

root@first_tc:~# cat /etc/ld.so.preload
cat: /etc/ld.so.preload: No such file or directory

root@first_tc:~# xxd /etc/ld.so.preload
00000000: 2f6c 6962 3634 2f73 656c 696e 7578 2e73  /lib64/selinux.s
00000010: 6f2e 330a                                o.3
```

debugfs


```
root@first_tc:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        24G  2.3G   21G  10% /
[..]

root@first_tc:~# debugfs /dev/vda1
debugfs 1.47.0 (5-Feb-2023)

debugfs:  cat /etc/ld.so.preload
/lib64/selinux.so.3
debugfs:
```

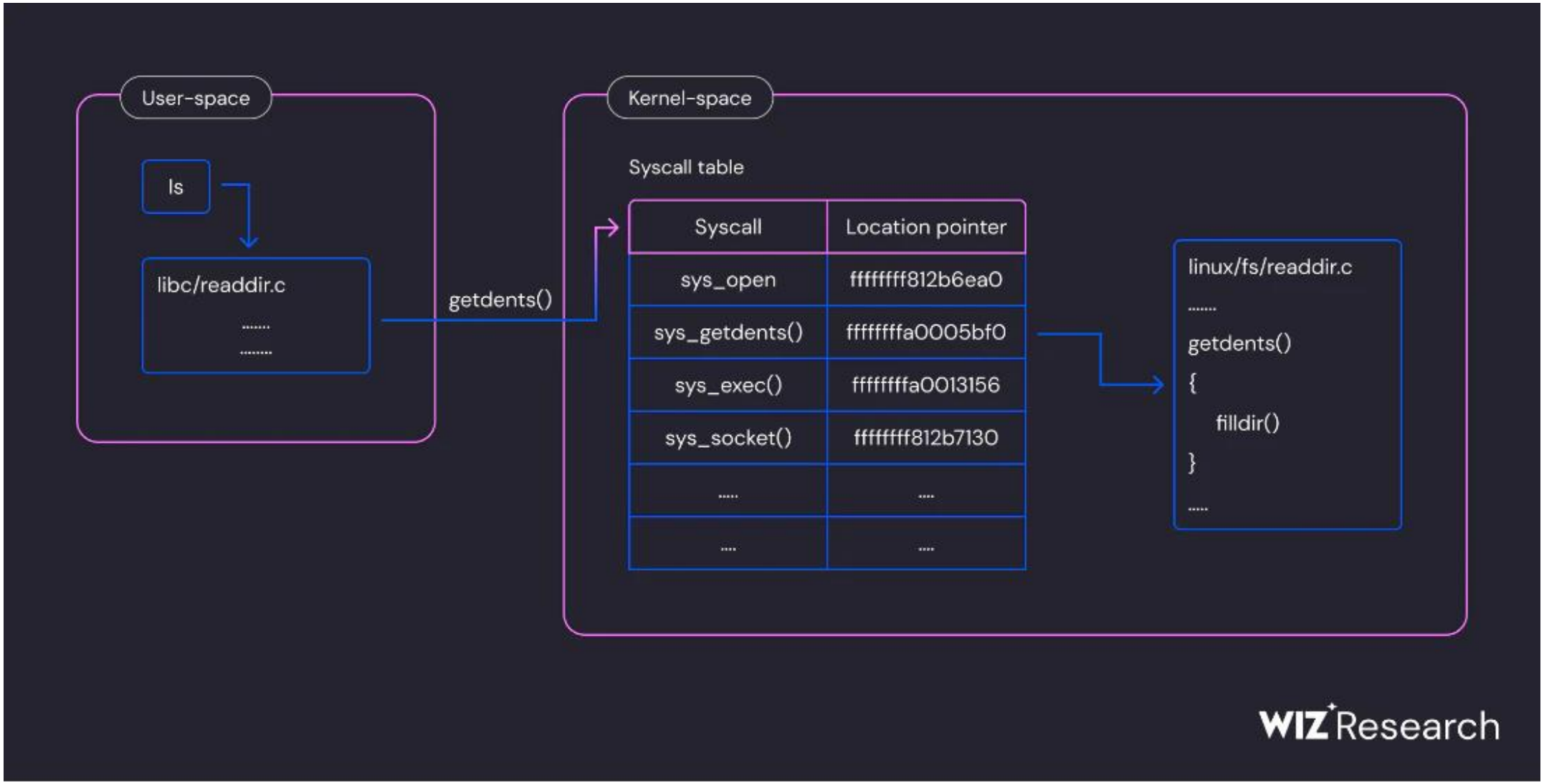
Output Encoding Scheme


```
root@first:~# LD_PRELOAD=./keyword_swap.so cat statement.txt
DFIR sucks!

root@first:~# LD_PRELOAD=./keyword_swap.so xxd statement.txt
00000000: 4446 4952 2072 6f63 6b73 210a          DFIR rocks!.

root@first:~# strace -s 1000 xxd statement.txt
write(1, "00000000: \33[1;32m44\33[0m\33[1;32m46\33[0m \33[1;32m49\33[0m\33[1;32m52\33[0m
\33[1;32m20\33[0m\33[1;32m72\33[0m \33[1;32m6f\33[0m\33[1;32m63\33[0m \33[1;32m6b\33[0m\33[1;32m73\33[0m
\33[1;32m21\33[0m\33[1;33m0a\33[0m       \33[1;31m \33[0m\33[1;31m \33[0m\33[1;31m \33[0m\33[1;31m \33[0m
\33[1;32mD\33[0m\33[1;32mF\33[0m\33[1;32mI\33[0m\33[1;32mR\33[0m\33[1;32m
\33[0m\33[1;32mr\33[0m\33[1;32mo\33[0m\33[1;32mc\33[0m\33[1;32mk\33[0m\33[1;32ms\33[0m\33[1;32m!\33[0m\33[1;33m.\33[0
m\n", 37200000000: 4446 4952 2072 6f63 6b73 210a          DFIR rocks!.
```

Kernelspace Rootkits

Hooking Techniques (Example)

Syscall Table Modification



Source: https://www.wiz.io/blog/linux-rootkits-explained-part-2-loadable-kernel-modules

Ftrace

MALMIUM

- ftrace = function tracer: trace and record the execution of functions within the Linux kernel

- ftrace hooks into the kernel at various trace points

- You can enable and disable these trace points dynamically, without the need to reboot or recompile the kernel

```
#define HOOK(_name, _function, _original)                    \
        {                                                    \
            .name = (_name),                                 \
            .function = (_function),                         \
            .original = (_original),                         \
        }

static struct ftrace_hook hooked_functions[] = {
        HOOK("sys_clone",   fh_sys_clone,   &real_sys_clone),
        HOOK("sys_execve",  fh_sys_execve,  &real_sys_execve),
};
```

Source: https://www.apriorit.com/dev-blog/546-hooking-linux-functions-2

Ftrace – Umbra

MALMIUM

```
193     // Syscalls to be hooked using frace
194     struct ftrace_hook hooks[] = {
195         //HOOK("sys_mkdir", hook_mkdir, &orig_mkdir),
196         HOOK("sys_kill", hook_kill, &orig_kill),
197         HOOK("sys_getdents64", hook_getdents64, &orig_getdents64),
198         HOOK("sys_getdents", hook_getdents, &orig_getdents)
199     };
```

```
163     //iterate through all directories
164     while (offset < ret){
165         current_dir = (void *)dirent_ker + offset;
166 
167         //Check if directory/file contains the umbra prefix
168         if(memcmp(UMBRA_DIRECTORY_PREFIX, current_dir->d_name, strlen(UMBRA_DIRECTORY_PREFIX))==0){
169             //Special case directory/file is the first
170             if(current_dir == dirent_ker){
171                 ret -= current_dir->d_reclen;
172                 memmove(current_dir, (void *)current_dir + current_dir->d_reclen, ret);
173                 continue;
174             }
175             //We hide this entry by incrementing the rec_len field, now pointing to the next entry
176             prev_dir->d_reclen += current_dir->d_reclen;
177             printk(KERN_INFO "UMBRA:: Skipped over secret entry.\n");
178         }else{
179             prev_dir = current_dir;
180         }
181         //Next entry
182         offset += current_dir->d_reclen;
```

Source: https://github.com/h3xduck/Umbra/blob/96443367bb35650d8b468a9e14557f01eeb26653/kernel/src/hookers.c

KHOOK - Linux Kernel hooking engine

MALMIUM

```
KHOOK_EXT(long, __x64_sys_kill, const struct pt_regs *);
static long khook___x64_sys_kill(const struct pt_regs *regs) {
   if (regs->si == 0) {
                        if (is_proc_invisible(regs->di)) {
                                return -ESRCH;
                        }
                }

                return KHOOK_ORIGIN(__x64_sys_kill, regs);
}
```

return –ESRCH => No process or process group can be found corresponding to that specified by pid.

Source: https://github.com/f0rb1dd3n/Reptile/blob/1e17bc82ea8e4f9b4eaf15619ed6bcd283ad0e17/kernel/main.c#L87

eBPF

Source: https://www.youtube.com/watch?v=g6SKWT7sROQ

eBPF

MALMIUM

```
// trace the entry to the openat syscall
SEC("tp/syscalls/sys_enter_openat")
int handle_read_enter(struct trace_event_raw_sys_enter *ctx)
{
        const int test_len = 5;
        const char test[test_len] = "test";
        char comm[test_len];

   // Use helper to get executable name:
   bpf_get_current_comm(&comm, sizeof(comm));

   // check to see if name matches
   if (int i = 0; i < test_len; i++) {
      if (test[i] != comm[i]) {
         return 0;
      }
   }
   // executable name is 'test', log event,
   // perform additional checks, to do anything else

   return 0;
```

Source: https://blog.tofile.dev/2021/08/01/bad-bpf.html

Hardening

modules_disabled

MALMIUM

```
# echo 1 > /proc/sys/kernel/modules_disabled

# insmod dfir.ko
insmod: ERROR: could not insert module dfir.ko:
Operation not permitted
```

- All subsequent attempts to load or unload kernel modules will fail.
- Switching the flag back is not possible too, only after a reboot of the machine.
- The unloading of already loaded modules is also not possible

kernel_lockdown

- The Kernel Lockdown feature is designed to prevent direct and indirect access to a running kernel image, attempting to protect against unauthorized kernel modification.

- When lockdown is in effect, several features are disabled or restricted.

- This includes:
  - /dev/mem
  - /dev/kmem
  - /dev/kcore
  - /dev/ioports
  - BPF
  - kprobes

Unprivileged eBP

- Nowadays, to install an eBPF program, you typically need root — or at least CAP_SYS_ADMIN and/or CAP_BPF (this was not always the case).

- Unprivileged eBPF was introduced around kernel 4.4.

- Be sure to check this config option by running:

```
# sysctl kernel.unprivileged_bpf_disabled
```

Unprivileged eBP

MALMIUM

- **`BPF_SYSCALL=n`** : Disables the BPF system call. Probably breaks most systemd-based systems.
- **`DEBUG_INFO_BTF=n`** : Disables generation of BTF debug information, i.e., CORE no longer works on this system. Forces attackers to compile on/for the system they want to compromise.
- **`BPF_LSM=n`** : BPF programs cannot be attached to LSM hooks.
- **`LOCK_DOWN_KERNEL_FORCE_INTEGRITY=y`** : Prohibits the use of **`bpf_probe_write_user`**.
- **`NET_CLS_BPF=n`** and **`NET_ACT_BPF=n`** : BPF programs cannot be used in TC classifier actions. Stops some data exfiltration techniques.
- **`FUNCTION_ERROR_INJECTION=n`** : Disables the function error injection framework, i.e., BPF programs can no longer use **`bpf_override_return`**.
- **`NETFILTER_XT_MATCH_BPF=n`** : Disables option to use BPF programs in nftables rules . Could be used to implement malicious firewall rules.
- **`BPF_EVENTS=n`** : Removes the option to attach BPF programs to kprobes, uprobes, and tracepoints.

Source: https://lolcads.github.io/posts/2023/12/bpf_memory_forensics_with_volatility3/