MEMORY FLUCTUATION ARTIFACTS

Conférence sur la Réponse aux Incidents & l'Investigation Numérique 2026

31 mars 2026 – Lille

PRESENTATION

CERT/CSIRT Engineer

CTF Player

Interests

Malware & exploit development

AV/EDR Evasion

HUGO PERINAZZO

Introduction

Windows

Linux

macOS

Evasion Reminder – Threats Lives in Memory

Dropping the clear payload on disk is not acceptable

Pattern matching / regex / YARA

OPSEC risk

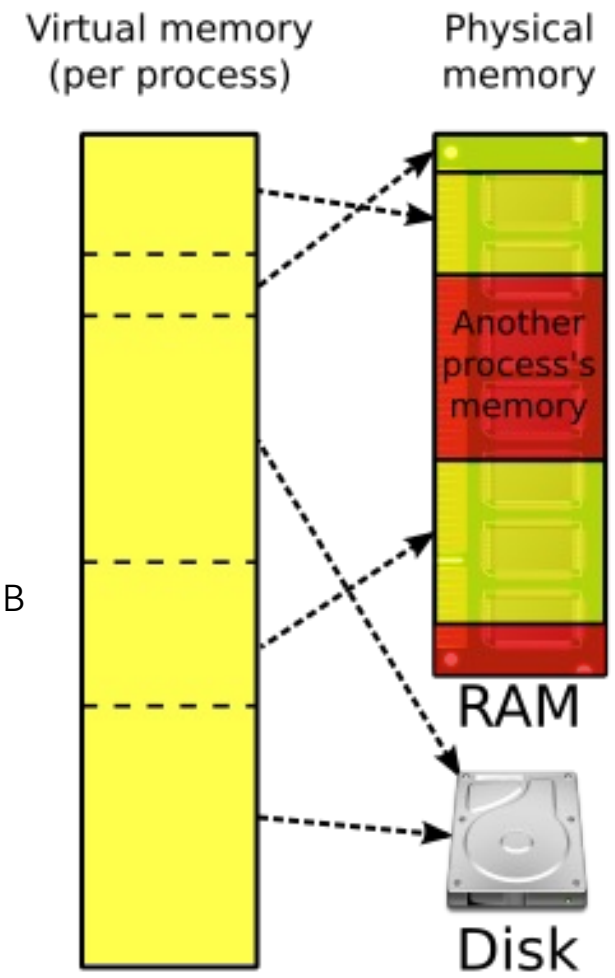
3 NATIVES PAYLOADS :

- Shellcode injection
- DLL via RDI / sRDI
- Custom PE32+

Packers, Crypters, Loaders, etc...

ex : UPX

Memory Scanning



32 bits → 4 GiB

64 bits → 256 TiB

- Memory scanning & pattern matching
- Memory metadata anomalies

forrest-orr / **moneta**

hasherezade / **pe-sieve**

Sleep Obfuscation

- Encrypt yourself in memory before sleeping, decrypt on wake.

EXAMPLE

Cobalt Strike release the Sleep Mask Kit in 2021

STRONG IOC

Unencrypted sleep stub

RWX/RX + MEM_PRIVATE

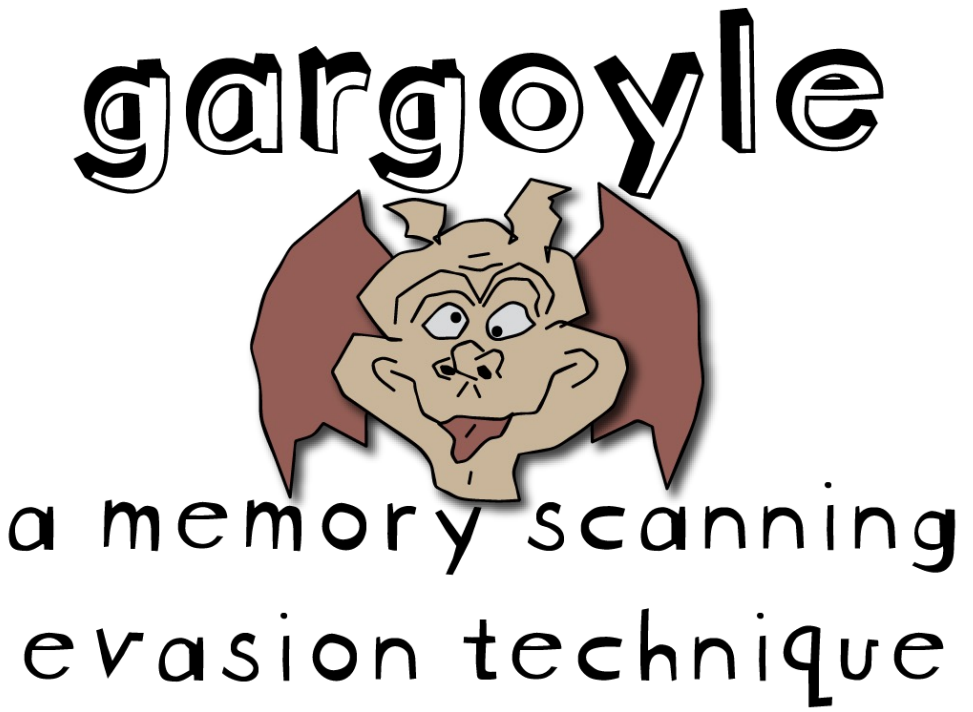
Gargoyle

- PoC demonstrating memory fluctuation
- Release in 2017
- 32 bit only

VARIANTS

waldo-irc / **YouMayPasser**

thefLink / **DeepSleep**



Memory Fluctuation & Sleep Obfuscation

Ensures that for 99% of the time your implant is on the system, it exists only as non-executable, encrypted data.

RW → RX → RW → RX → RW

NOACCESS → RX → NOACCESS

STRONG IOC

VirtualProtect / mprotect / mach_vm_protect

FALSE POSITIVES

- JIT
- NTDLL hooks
- DRM protected / packed applications

START EXECUTION

PROTECTION CHANGE:
- Set RW
- Encrypt (optional)

TRIGGER:
- Sleep hooking
- Timing mechanism
- Synchronization object
- Throwing an exception

UNPROTECT:
- Decrypt
- Set RX

Loop !

Evasion Recap

	Protection	Encryption	Trigger	Bootstrap
Gargoyle	✓		APC Timer	ROP
Obfuscate-and-sleep		✓	Sleep	Shellcode
FOLIAGE	✓	✓	APC Timer	CONTEXT
Shellcode Fluctuation	✓		Exception	Shellcode
DeepSleep	✓		Sleep	ROP
Ekko	✓	✓	Timer queue	CONTEXT

This talk is about sleep obfuscation + fluctuation

- Because it's easier to implement for the three major OSes
- We focus on fluctuation itself, not the trigger

Introduction

Windows

Linux

macOS

Windows Maldev

Inspired by  mgeeky / **ShellcodeFluctuation**

Hook

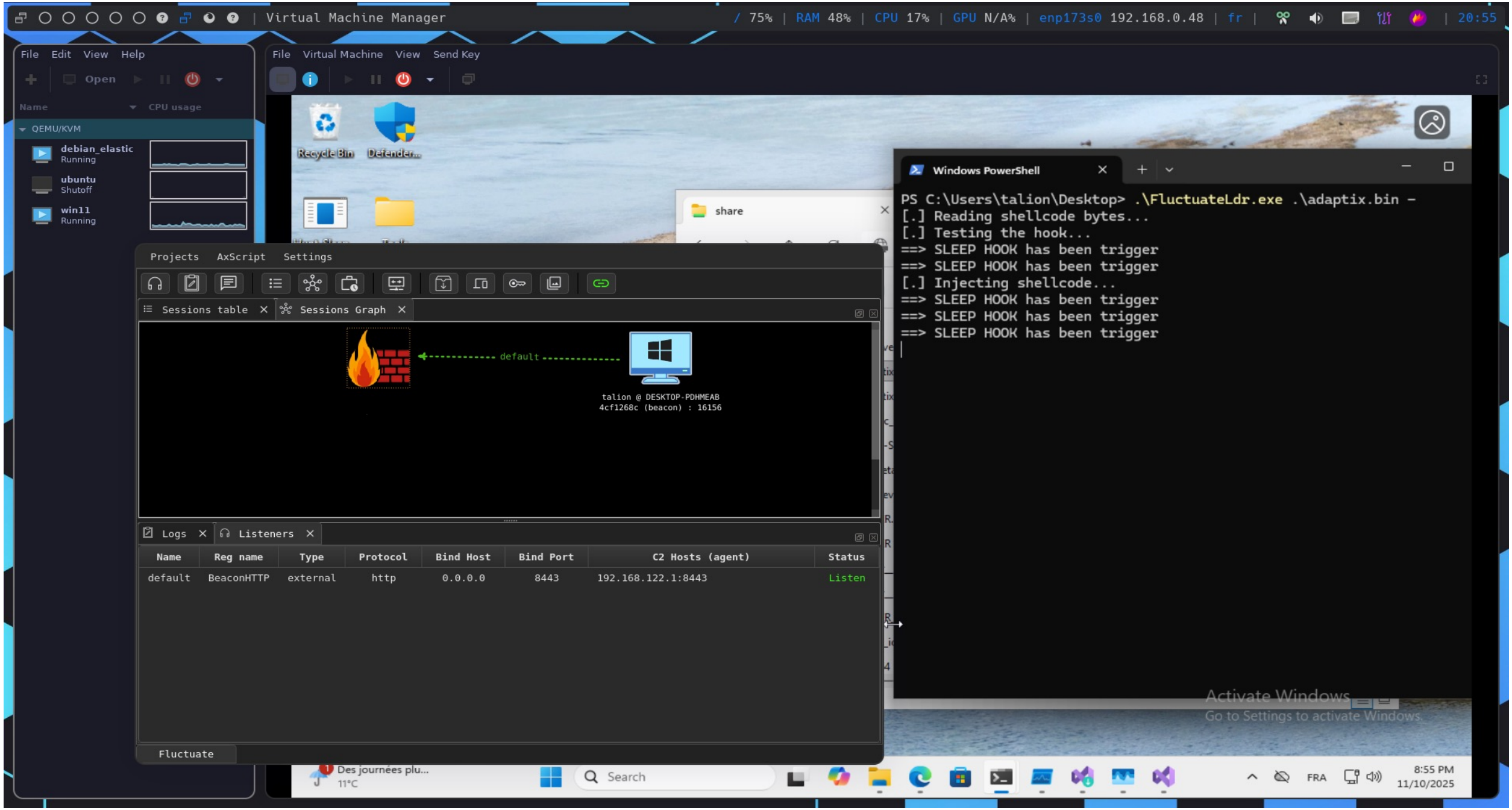
kernel32!Sleep

Lifecycle

Decrypt → Unhook → Sleep → Encrypt → Hook → Return
Decrypt → Unhook → Sleep → Encrypt → Hook → Return

```
uint8_t trampoline[] = {
    0x49, 0xBA, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, // mov r10, addr
    0x41, 0xFF, 0xE2              // jmp r10
};
```

```
void WINAPI MySleep(DWORD dwMilliseconds) {
  // ...

  if (!g_fluctuationData.currentlyEncrypted)
    toggleEncryption();

  fastTrampoline(false, (BYTE *)::Sleep, (void *)&MySleep, &buffers);

  ::Sleep(dwMilliseconds);

  fastTrampoline(true, (BYTE *)::Sleep, (void *)&MySleep);

  // ...
}
```

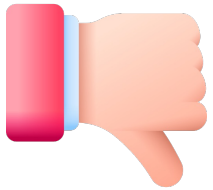

Virtual Machine Manager
/ 75% | RAM 48% | CPU 17% | GPU N/A% | enp173s0 192.168.0.48 | fr | 20:55
File Edit View Help
Open
Name CPU usage
QEMU/KVM
debian_elastic
Running
ubuntu
Shutoff
win11
Running
File Virtual Machine View Send Key
Recycle Bin
Defender...
share
Windows PowerShell
PS C:\Users\talion\Desktop> .\FluctuateLdr.exe .\adaptix.bin -
[.] Reading shellcode bytes...
[.] Testing the hook...
==> SLEEP HOOK has been trigger
==> SLEEP HOOK has been trigger
[.] Injecting shellcode...
==> SLEEP HOOK has been trigger
==> SLEEP HOOK has been trigger
==> SLEEP HOOK has been trigger
Projects AxScript Settings
Sessions table
Sessions Graph
default
talion @ DESKTOP-PDHMEAB
4cf1268c (beacon) : 16156
Logs
Listeners
Name Reg name Type Protocol Bind Host Bind Port C2 Hosts (agent) Status
default BeaconHTTP external http 0.0.0.0 8443 192.168.122.1:8443 Listen
Fluctuate
Activate Windows
Go to Settings to activate Windows.
Des journées plu...
11°C
Search
FRA
8:55 PM
11/10/2025

Detection via Userland Hooks

Place a hook in NTDLL : VirtualProtect

Keep an inventory & count every permission change

Define a threshold



Bypassable / Patchable

Detection via ETW TI

NtProtectVirtualMemory

```
NTSYSAPI
NTSTATUS
NTAPI


NtProtectVirtualMemory(


  IN HANDLE               ProcessHandle,
  IN OUT PVOID            *BaseAddress,
  IN OUT PULONG           NumberOfBytesToProtect,
  IN ULONG                NewAccessProtection,
  OUT PULONG              OldAccessProtection );
```

pathtofile / **Sealighter**

pathtofile / **SealighterTI**

jdu2600 / **EtwTi-FluctuationMonitor**

```
Microsoft Visual Studio Debug Console
[*] Enabling Microsoft-Windows-Threat-Intelligence (KEYWORD_PROTECTVM_LOCAL)
[*] Monitoring for VirtualProtect() for 30 seconds
[.] java.exe 0000014F66300000 RW- => RWX
[.] Ekko.exe 00007FF7962A0000 RW- => RWX
[.] Ekko.exe 00007FF7962A0000 RWX => RW-
[.] Ekko.exe 00007FF7962A0000 RW- => RWX
[!] Ekko.exe 00007FF7962A0000 is fluctuating
[.] Ekko.exe 00007FF7962A0000 RWX => RW-
[*] Done
```

KERNEL CODE EXECUTION

- via PPLDump exploit
- via BYOVD

EPROCESS

- PPL
- EMAL

ETW TI Event

- THREATINT_PROTECTVM_LOCAL

Detection via CFG Bitmap

Control Flow Guard

Exploit mitigation

Compiler flag: `/guard:cf`

Use a bitmap of valid indirect call targets

SetProcessValidCallTargets function (memoryapi.h)

Summarize this article for me

Provides Control Flow Guard (CFG) with a list of valid indirect call targets and specifies whether they should be marked valid or not. The valid call target information is provided as a list of offsets relative to a virtual memory range (start and size of the range). The call targets specified should be 16-byte aligned and in ascending order.

Syntax

C++ Copy

```
BOOL SetProcessValidCallTargets(
  [in]      HANDLE                hProcess,
  [in]      PVOID                 VirtualAddress,
  [in]      SIZE_T                RegionSize,
  [in]      ULONG                 NumberOfOffsets,
  [in, out] PCFG_CALL_TARGET_INFO OffsetInformation
);
```

```
PS C:\Users\talion\Desktop> .\CFG-FindHiddenShellcode.exe
===== Hidden Executable Pages - quickly scanning all process
PS C:\Users\talion\Desktop> .\CFG-FindHiddenShellcode.exe
===== Hidden Executable Pages - quickly scanning all process
ServiceHub.IdentityHost.exe(7268) - 1 hidden allocations
 * 000001F37B01A000 MEM_MAPPED
PS C:\Users\talion\Desktop> .\CFG-FindHiddenShellcode.exe
===== Hidden Executable Pages - quickly scanning all process
ServiceHub.IdentityHost.exe(7268) - 1 hidden allocations
 * 000001F37B01A000 MEM_MAPPED
PS C:\Users\talion\Desktop>
```

```
[>] Flipped to RW.
[<] Decoding...
[<] Flipped back to RX/RWX.

[.] Access Violation occured at 0x24fca0800ba
[+] Shellcode wants to Run. Restoring to RX and Decrypting

[>] Flipped to RW.
[>] Encoding...
```

Windows Takeaways

DFIR

Moneta & Pe-sieve

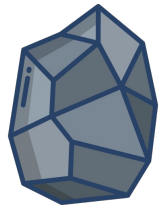
jdu2600/EtwTi-FluctuationMonitor

jdu2600/CFG-FindHiddenShellcode

EDR

Monitor VirtualProtect !

TI Event : THREATINT_PROTECTVM_LOCAL



Hardening

ACG

HSP

Introduction

Windows

Linux

macOS

Process Scanning

ptrace(PTRACE_PEEKDATA, ...)

process_vm_readv()

via procfs :

- /proc/<pid>/mem
- /proc/<pid>/maps

```
sh-5.3$ ./memscan 3610223
Private and Executable Memory Regions for PID 3610223:

Address Range          Perms  Offset   Dev   Inode       Pathname
-------------------------------------------------------------------------------
0x555555575000-0x55555561b000 r-xp 00135168 103:0 0000000002 14054150
0x7ffff7c28000-0x7ffff7da9000 r-xp 00163840 103:0 0000000002 13799953
0x7ffff7fb6000-0x7ffff7fb7000 r-xp 00004096 103:0 0000000002 13799958
0x7ffff7fc2000-0x7ffff7fc4000 r-xp 00000000 00:00 0000000000 [vdso]
0x7ffff7fc5000-0x7ffff7ff0000 r-xp 00004096 103:0 0000000002 13799947
0xffffffffff600000-0xffffffffff601000 --xp 00000000 00:00 0000000000 [vsyscall]
-------------------------------------------------------------------------------
Total: 6 private executable regions
```

```
std::string maps_path = "/proc/" + std::to_string(pid) + "/maps";
```

Linux Maldev

ELF to shellcode

jonatanSh / **shelf**

fork() + PTRACE_TRACEME

The child execute the shellcode

Intercept syscalls

nanosleep

clock_nanosleep

kyleavery / **pendulum**

```
$ ./ELFluctuateLdr ./shellcode --fluctuate
Shellcode loaded at: 0x770e5fc8b000
[TRACER] Got shellcode from child: 0x770e5fc8b000, size 761856
[TRACER] Inject address (vdso): 770e6009e000
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
```

Fork allow you to trace the child and intercept syscalls

Linux Maldev

Unprotected

```
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
```

```
Every 0.1s: ./memscan 94696 94695                                   ubuntu: Tue Mar 10 14:46:50 2026

Private and Executable Memory Regions for PID 94696:

Address Range           Perms  Offset   Dev   Inode       Pathname
-------------------------------------------------------------------------------
0x5ac25de1a000-0x5ac25de1c000 r-xp 00004096 fd:02 0001584393 /home/talion/Desktop/shelf/v3_sleepobf/ELF
luctuateLdr
0x79c0fa68b000-0x79c0fa745000 r-xp 00000000 00:00 0000000000
0x79c0fa828000-0x79c0fa9b0000 r-xp 00163840 fd:02 0001849807 /usr/lib/x86_64-linux-gnu/libc.so.6
0x79c0faa9d000-0x79c0faa9f000 r-xp 00000000 00:00 0000000000 [vdso]
0x79c0faaa0000-0x79c0faacb000 r-xp 00004096 fd:02 0001849804 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.
so.2
0xffffffffff600000-0xffffffffff601000 --xp 00000000 00:00 0000000000 [vsyscall]
-------------------------------------------------------------------------------
Total: 6 private executable regions
```

```
Every 0.1s: sudo yara ./rule.yar 94695                              ubuntu: Tue Mar 10 14:46:49 2026

detect_payload_execute 94695
```

Protected

```
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
```

```
Every 0.1s: ./memscan 94696 94695                                   ubuntu: Tue Mar 10 14:47:12 2026

Private and Executable Memory Regions for PID 94696:

Address Range           Perms  Offset   Dev   Inode       Pathname
-------------------------------------------------------------------------------
0x5ac25de1a000-0x5ac25de1c000 r-xp 00004096 fd:02 0001584393 /home/talion/Desktop/shelf/v3_sleepobf/ELF
luctuateLdr
0x79c0fa828000-0x79c0fa9b0000 r-xp 00163840 fd:02 0001849807 /usr/lib/x86_64-linux-gnu/libc.so.6
0x79c0faa9d000-0x79c0faa9f000 r-xp 00000000 00:00 0000000000 [vdso]
0x79c0faaa0000-0x79c0faacb000 r-xp 00004096 fd:02 0001849804 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.
so.2
0xffffffffff600000-0xffffffffff601000 --xp 00000000 00:00 0000000000 [vsyscall]
-------------------------------------------------------------------------------
Total: 5 private executable regions
```

```
Every 0.1s: sudo yara ./rule.yar 94695                              ubuntu: Tue Mar 10 14:47:12 2026
```

Sleep based fluctuation is working !

Linux Maldev

Unprotected

```
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
```

```
Every 0.1s: ./memscan 94696 94695                                   ubuntu: Tue Mar 10 14:46:50 2026

Private and Executable Memory Regions for PID 94696:

Address Range           Perms  Offset   Dev   Inode       Pathname
------------------------------------------------------------------------------
0x5ac25de1a000-0x5ac25de1c000 r-xp 00004096 fd:02 0001584393 /home/talion/Desktop/shelf/v3_sleepobf/ELF
luctuateLdr
0x79c0fa68b000-0x79c0fa745000 r-xp 00000000 00:00 0000000000
0x79c0fa828000-0x79c0fa9b0000 r-xp 00163840 fd:02 0001849807 /usr/lib/x86_64-linux-gnu/libc.so.6
0x79c0faa9d000-0x79c0faa9f000 r-xp 00000000 00:00 0000000000 [vdso]
0x79c0faaa0000-0x79c0faacb000 r-xp 00004096 fd:02 0001849804 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.
so.2
0xffffffffff600000-0xffffffffff601000 --xp 00000000 00:00 0000000000 [vsyscall]
------------------------------------------------------------------------------
Total: 6 private executable regions
```

```
Every 0.1s: sudo yara ./rule.yar 94695                              ubuntu: Tue Mar 10 14:46:49 2026

detect_payload_execute 94695
```

Linux Maldev

protected

```
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
[FLUCT] Memory decrypted (RX), resuming
payload execute
[FLUCT] Memory encrypted (RW), sleeping 2s...
```

```
Every 0.1s: ./memscan 94696 94695                       ubuntu: Tue Mar 10 14:47:12 2026

Private and Executable Memory Regions for PID 94696:                                    ?

Address Range           Perms  Offset   Dev   Inode       Pathname
-------------------------------------------------------------------------------
0x5ac25de1a000-0x5ac25de1c000 r-xp 00004096 fd:02 0001584393 /home/talion/Desktop/shelf/v3_sleepobf/ELF
luctuateLdr
0x79c0fa828000-0x79c0fa9b0000 r-xp 00163840 fd:02 0001849807 /usr/lib/x86_64-linux-gnu/libc.so.6
0x79c0faa9d000-0x79c0faa9f000 r-xp 00000000 00:00 0000000000 [vdso]
0x79c0faaa0000-0x79c0faacb000 r-xp 00004096 fd:02 0001849804 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.
so.2
0xffffffffff600000-0xffffffffff601000 --xp 00000000 00:00 0000000000 [vsyscall]
-------------------------------------------------------------------------------
Total: 5 private executable regions
```

```
Every 0.1s: sudo yara ./rule.yar 94695                  ubuntu: Tue Mar 10 14:47:12 2026
                                        ?
```

Sleep based fluctuation is working !

Telemetry Investigation

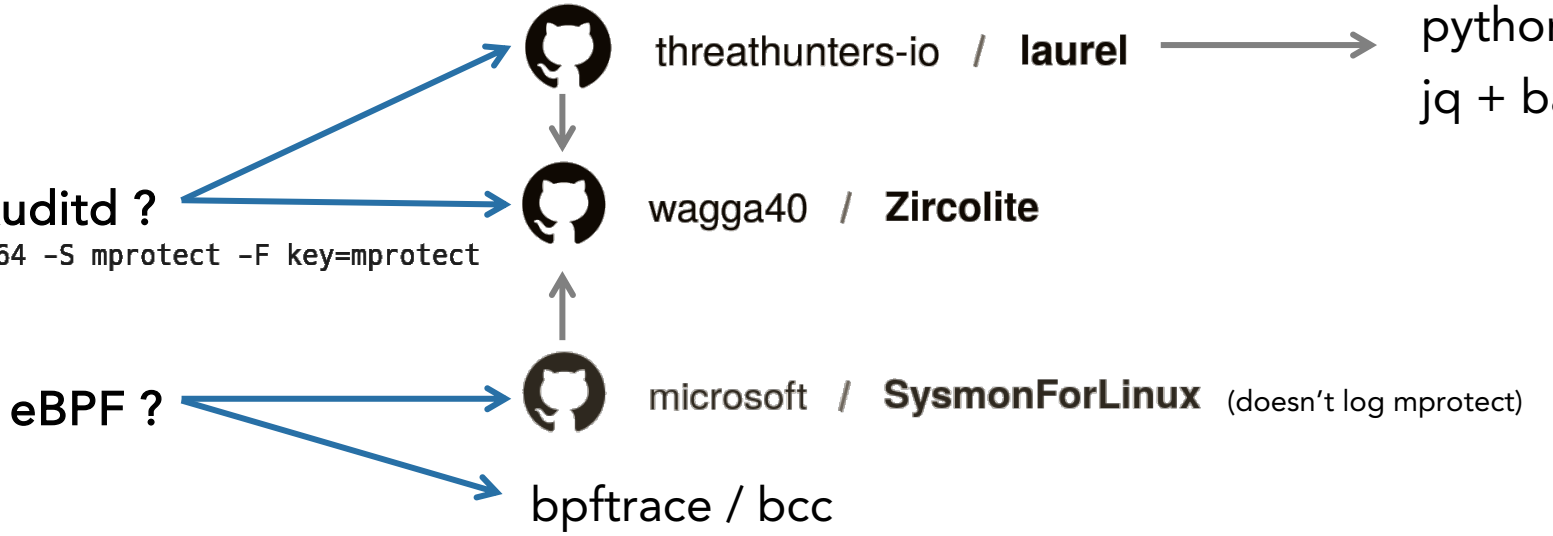
Userland

HOW ?
- LD_PRELOAD
- /etc/ld.so.preload

USAGE
- Reverse
- Live Analysis ?

USELESS WHEN
- Static binary
- Direct syscall
- Can be patch

Kerneland




```
bpftrace -e 'tracepoint:syscalls:sys_enter_mprotect { printf("%s %d\n", comm, pid); }'
```

Telemetry investigation

auditd (mprotect) + laurel + python script

```
talion@ubuntu:~/Desktop/auditd_mprotect_fluct$ sudo ./venv/bin/python ./auditd_mprotect_fluct.py
FLUCTUATION DETECTED !! => PID: 172976 (ELFluctuateLdr) - /tmp/ELFluctuateLdr/ELFluctuateLdr
FLUCTUATION DETECTED !! => PID: 175155 (nvim) - /usr/bin/nvim
```

Zircolite

auditd.log

Severity	Rule	Events	ATT&CK
HIGH	High Frequency mprotect Calls - Memory Fluctuation Indicator	39	T1055
MEDIUM	mprotect Memory Fluctuation Pattern - RW to RX Transition	377	T1055.001
MEDIUM	mprotect with Executable Memory Permissions	103	T1055.001
MEDIUM	mprotect on Anonymous Memory Region	15	T1055

```
zircolite.py --events ./logs/auditd.log --ruleset custom_mprotect_rules.json --auditd --config Zircolite/config/config.yaml
```

Live Analysis

In-kernel virtual machine that allows safe, sandboxed programs to run in the Linux kernel.

3 common tracing mechanisms :

- **kprobes/kretprobes :** Dynamically hook any kernel function.
- **tracepoints :** Predefined hook points in the kernel
- **uprobes** : Similar to kprobes but hook user-space functions

> Note that more tracing mechanisms exist

eBPF

Fully supported since Linux kernel 5.x.

```
talion@ubuntu:~/Desktop/ebpf_mon$ sudo bpftrace ./mprotect_fluct.bt
Attaching 4 probes...
Monitoring mprotect fluctuations (RWX/RX <-> RW transitions)...
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 1903 (gnome-shell)
FLUCTUATION DETECTED !! => PID: 175930 (ELFluctuateLdr)
```

Linux Takeaways

Fluctuation is possible on Linux.

DFIR

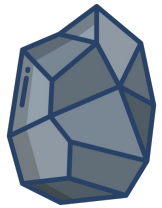
Use bpftrace

Use Zircolite

EDR

Memory scanning work fine

Monitor mprotect with eBPF



Hardening

Log kernel telemetry (auditd / sysmonforlinux)

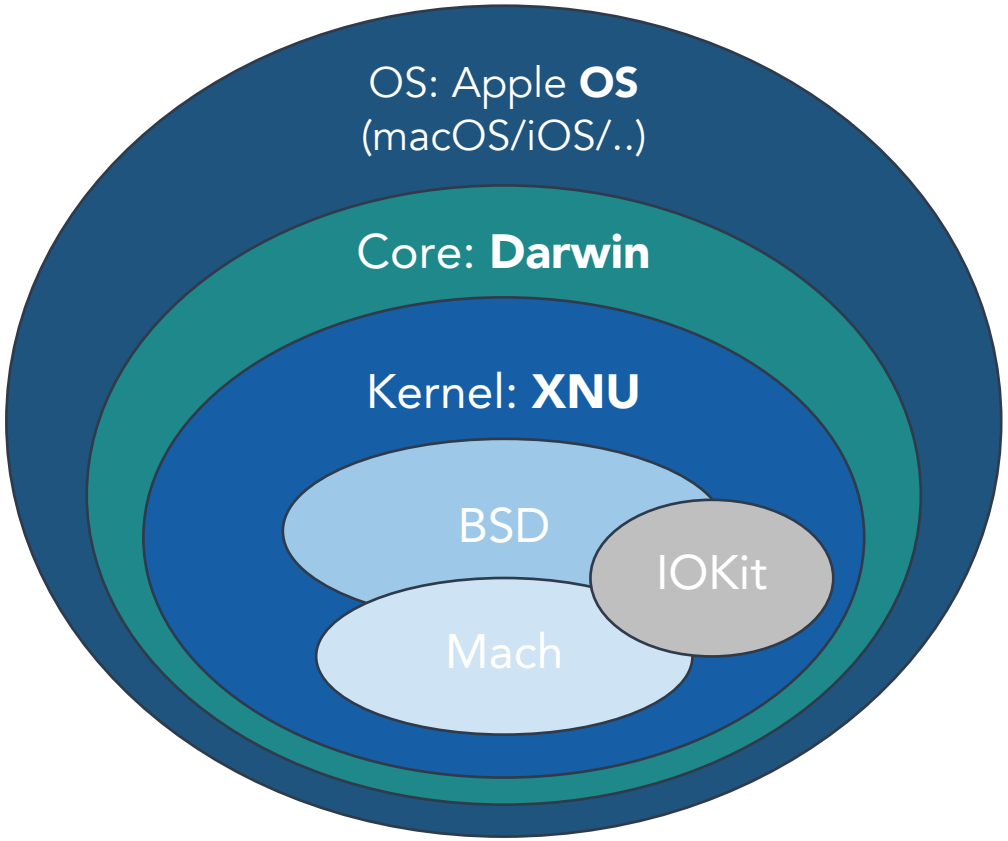
MAC / Application Whitelisting (fapolicyd)

Yama LSM : double-edged

Seccomp

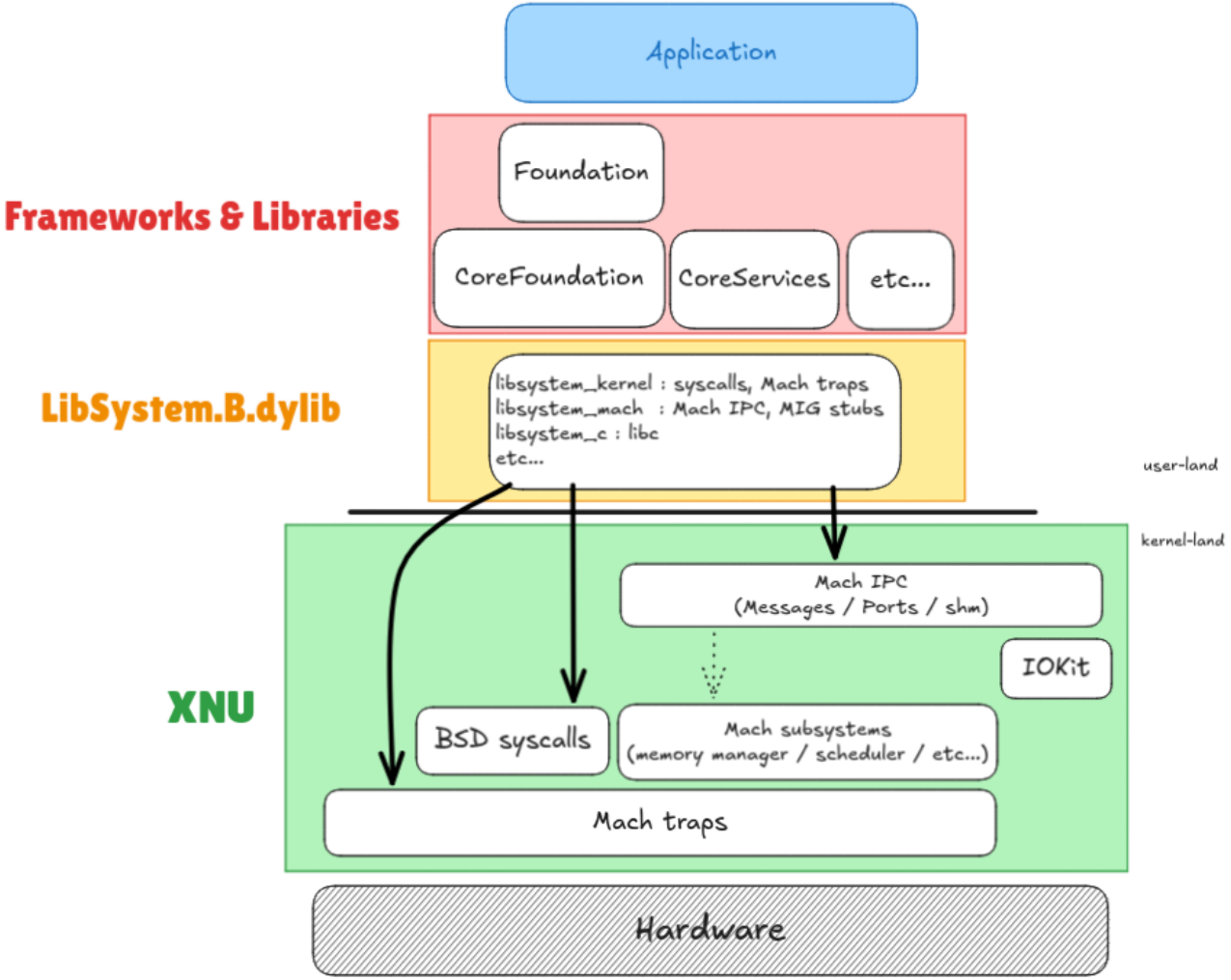

Introduction
Windows
Linux
macOS

Architecture




```
macOS / iOS
 ├─ Proprietary frameworks (GUI, graphics, etc.)
 └─ Darwin (Open-source OS)
      ├─ XNU (hybrid kernel)
      │    ├─ Mach (microkernel) : threads, IPC, VM
      │    ├─ BSD                : POSIX, processes, networking
      │    └─ IOKit              : drivers
      └─ BSD userland (shell, coreutils)
```

Internals


Application
Frameworks & Libraries
Foundation
CoreFoundation
CoreServices
etc...
LibSystem.B.dylib
libsystem_kernel : syscalls, Mach traps
libsystem_mach : Mach IPC, MIG stubs
libsystem_c : libc
etc...
user-land
kernel-land
XNU
Mach IPC
(Messages / Ports / shm)
IOKit
BSD syscalls
Mach subsystems
(memory manager / scheduler / etc...)
Mach traps
Hardware

Process Scanning

task_for_pid()

Get a **task** on a remote process

```
kern_return_t kr = task_for_pid(mach_task_self(), pid, &task);
if (kr != KERN_SUCCESS) {
    printf("task_for_pid failed: %s\n", mach_error_string(kr));
    return 1;
}

...

if (info.protection & VM_PROT_READ) {

    vm_offset_t data;
    mach_msg_type_number_t data_count;

    kr = mach_vm_read(task, address, region_size, &data, &data_count);

    if (kr == KERN_SUCCESS) {

        search_in_buffer((uint8_t *)data, data_count, address);

        vm_deallocate(mach_task_self(), data, data_count);
    }
}
```

mach_vm_read()

Iterate over each readable region

Try to read it’s memory

Then search for pattern matching

```
sh-3.2$ sudo ./scanner 15932
Found "NOTMALICIOUS" at address: 0x104257ec8
Found "NOTMALICIOUS" at address: 0x104274000
```

We accessed the memory !

Taskgated

Has been heavily restricted

BENEFIT :

- In-memory secrets
- Remote process injection
- TCC bypass
- Sandbox escape

CAVEATS:

- Memory scanning is dead

Root can only access non-hardened and non-Apple process

`com.apple.security.get-task-allow` → same-user access

`com.apple.system-task-ports` → access other processes (except kernel)

`com.apple.private.cs.debugger` → inject same-user, non-hardened processes

SO ADVERSARIES LIVES IN MEMORY

Restoring Reflective Code Loading on macOS

Apple silently 'broke' in-memory code loading on macOS ...let's restore it!

by: Patrick Wardle / December 16, 2024

https://objective-see.org/blog/blog_0x7C.html

RESTORING DYLD MEMORY LOADING

https://blog.xpnsec.com/restoring-dyld-memory-loading

macOS Maldev

MACH_TASK_SELF()

You can still obtain a Mach port
to your own address space.

HOOKING SLEEP

get sleep address
and patch LibSystem.B.dylib

SHELLCODE INJECTION

Inject in your own address space

Sleep based fluctuation is working !

```
int main(int argc, char **argv) {
    const char *path = argv[1];

    int fd = open(path, O_RDONLY);

    sleepAddr = (void *)sleep;
    hookedSleepAddr = (void *)hookedSleep;

    resolvePatchPlaceholder(hookedSleepAddr);
    backupOriginalSleep();

    printf("HookSleep...\n");
    hookSleep();
    printf("Calling sleep...\n");

    return selfInjectShellcode(fd);
}
```

```
sh-3.2$ ./run ./shellcode.bin
HookSleep...
Calling sleep...
HookedSleep triggered !
unhooking...
calling the real sleep...
sleep done, hooking again...
```

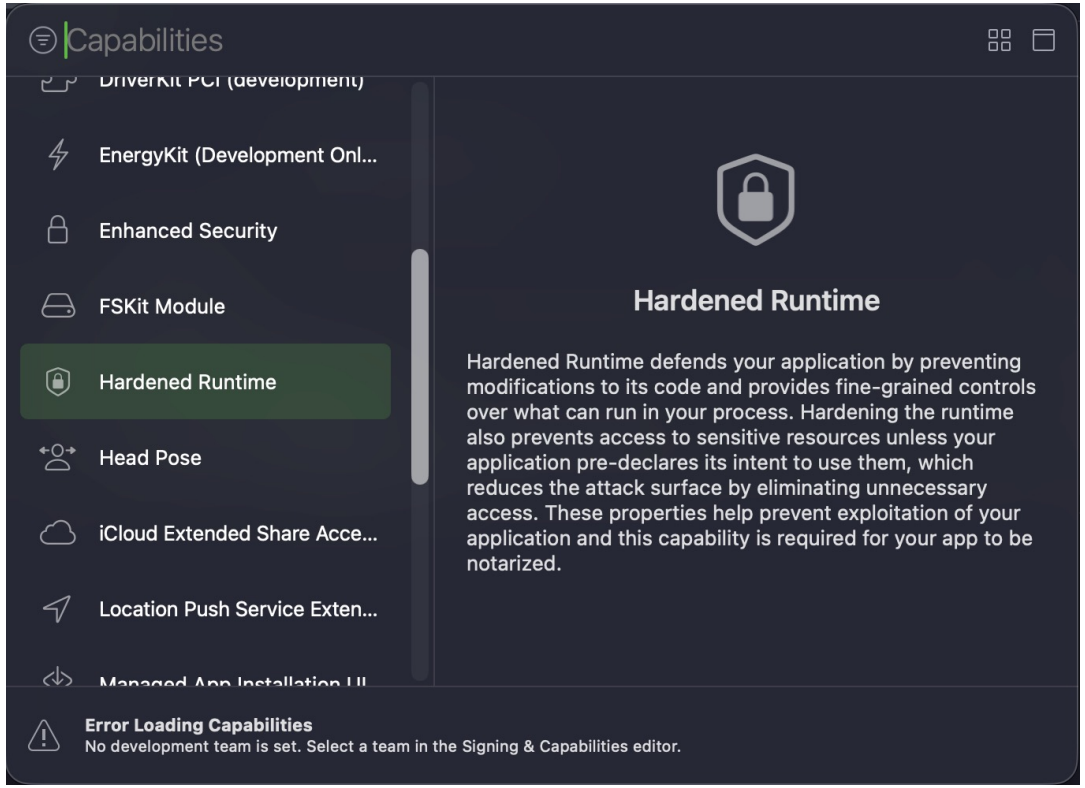
Hardened Runtime

It **validates the signature of all loaded code in the process,** which means the attacker cannot:

- Inject code at runtime.
- Modify libraries at rest
- Even root can't get a Task Port on the malware

Unless they use entitlements !

```
sh-3.2$ codesign -dv /Applications/Dozer.app 2>&1 | grep -i runtime
CodeDirectory v=20500 size=5406 flags=0x10000(runtime) hashes=162+3
Runtime Version=10.14.0
```



Live Analysis

You cannot dump the memory anymore

- Kext are deprecated
- Taskgated is enforced (no tfp0)

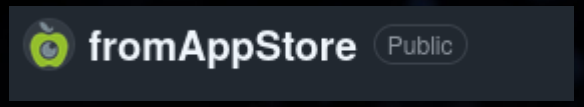
Except for VMs (opencore).

ARM unified memory breaks traditional Volatility structures.

Memory analysis now relies on live analysis

Triage FromAppStore

Application distributed on the AppStore have **hardened runtime + sandboxing** enforced



fromAppStore

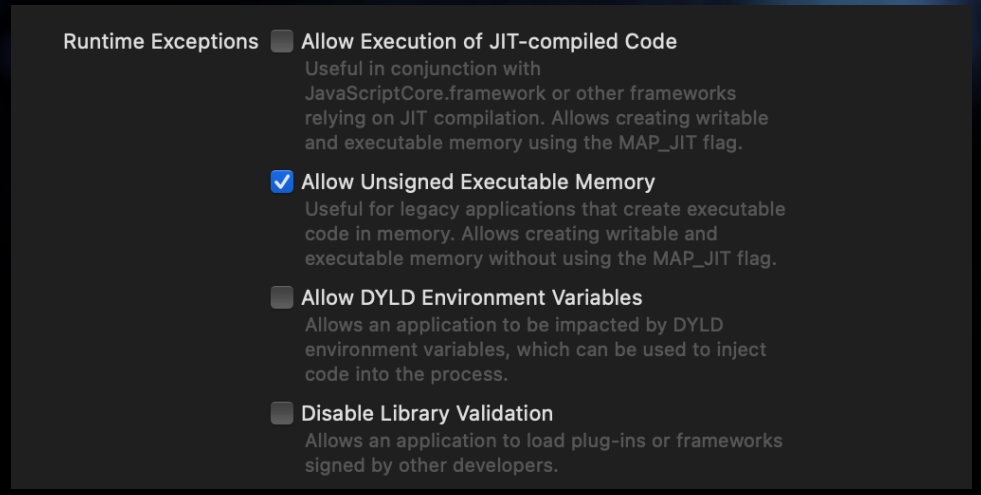
Checks if an application is pristine (untampered) and from the official Mac App Store.
For technical details, see the blog post; "Are you from the Mac App Store?"

```
find /Applications -maxdepth 1 -name "*.app" -type d -exec ./fromAppStore {} \;
```

Triage Hardened Process

HUNT HARDENED RUNTIME APP WITH SENSIBLE ENTITLEMENT :

- com.apple.security.cs.allow-jit
- com.apple.security.cs.allow-unsigned-executable-memory
- com.apple.security.cs.disable-executable-page-protection
- com.apple.security.cs.debugger




```
sh-3.2$ sudo ./triage-hardened
Password:
PID     USER    STATUS  PATH
4350    root    NON-HARDENED    /Users/yunor/Desktop/kitchen/triage-hardened
4349    root    SIP     /usr/bin/sudo
4330    yunor   HARDENED/SUSPICIOUS     /Users/yunor/.local/share/mise/installs/node/22.6.0/bin/node
4329    yunor   HARDENED/SUSPICIOUS     /Users/yunor/.local/share/mise/installs/node/22.6.0/bin/node
4328    root    SIP     /usr/bin/sudo
4292    yunor   SIP     /System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/Metadata.framework/Versions/A/Support/mdworker_shared
3938    yunor   HARDENED/SUSPICIOUS     /Applications/Zen Browser.app/Contents/MacOS/plugin-container.app/Contents/MacOS/plugin-container
3664    yunor   SIP     /System/Library/Frameworks/NetFS.framework/Versions/A/XPCServices/PlugInLibraryService.xpc/Contents/MacOS/PlugInLibraryService
3473    yunor   HARDENED/SUSPICIOUS     /Applications/Zen Browser.app/Contents/MacOS/plugin-container.app/Contents/MacOS/plugin-container
2973    yunor   HARDENED/SUSPICIOUS     /Applications/Zen Browser.app/Contents/MacOS/plugin-container.app/Contents/MacOS/plugin-container
91399   yunor   SIP     /bin/zsh
```

Memory Mappings

vmmap

We can detect basic shellcode injection thanks to memory mappings

It's the conceptual equivalent of /proc/<pid>/maps + /proc/<pid>/smaps on Linux,

You have to specify a target

executable ? **non-backed ?** **mach_vm_allocate()**

```
sh-3.2$ vmmap Injector | grep 'r-x\|rwx' | grep 'SM=PRV\|SM=NUL\|SM=COW' | grep 'VM_ALLOCATE'
Found process 80082 (basicShellcodeInjector) from partial name Injector
VM_ALLOCATE                 1026ac000-102710000    [  400K    16K    16K     0K] r-x/rwx SM=PRV
```

Fluctuation will bypass this detection

Hunt with Kernel Telemetry

DTrace / dtruss

```
sh-3.2$ sudo dtrace -ln 'syscall::entry'
Password:
dtrace: system integrity protection is on, some features will not be available

   ID   PROVIDER            MODULE                          FUNCTION NAME
dtrace: failed to match :syscall::entry: System Integrity Protection is on
```

- Modern tracing
- Similar to linux eBPF
- Useless if SIP enabled

ktrace ?

```
sh-3.2$ sudo ktrace trace -p $PID | grep "vm_protect"
2026-03-08 15:10:34.162268 CET         0.3             MSC_mach_vm_protect_trap
  2(AP-E) run(47618)
2026-03-08 15:10:34.162300 CET        32.4(32.4)       MSC_mach_vm_protect_trap
  2(AP-E) run(47618)
2026-03-08 15:10:34.162325 CET         0.2             MSC_mach_vm_protect_trap
  2(AP-E) run(47618)
```

we trace one pid for demo, but you can trace the whole system

- Old
- Read kernel tracepoints
- It works

Endpoint Security Framework

Leverage the endpoint security framework API.

Count of protection changes per page

We can use eslogger to query ESF

```
Starting mprotect monitoring with eslogger...
NOTE: Run this script with sudo: sudo python3 mprotect_monitor.py
Press Ctrl+C to stop and display final inventory
------------------------------------------------------------

!!! FLUCTUATION DETECTED !!! More than 5 times on page: 0x104848000
  Size: 16384, Current Prot: 3
  Process: notmalicious (PID: 69025)
  Total changes: 6
  Change history:
    1. prot=1, pid=69025
    2. prot=3, pid=69025
    3. prot=1, pid=69025
    4. prot=3, pid=69025
    5. prot=1, pid=69025
    6. prot=3, pid=69025
```

Note: eslogger is not designed for automation or large-scale data collection.

macOS Takeaways

Search for **behaviors** over **signatures**

DFIR

RAM dumping is dead, do live analysis :

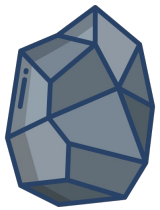
- eslogger for ESF telemetry
- vmmap search for anonymous pages
- Triage with objective-see/fromAppStore

EDR

Memory scanning is dead

Monitor via ESF

- mach_vm_allocate
- mach_vm_protect



Hardening

Enforce GateKeeper policy

Install MAC / App whitelisting :

- northpolesec/santa
- MDM

Conclusion

Strong Indicator: VirtualProtect / mprotect / mach_vm_protect

False Positives: JIT / NTDLL Hooks / DRM protected or packed applications

Reco: Allow only file backed memory to execute + Monitor memory page permission switches

WINDOWS

- Cradle of fluctuation
- Monitor VirtualProtect
- Use opensource scanners

LINUX

- Fluctuation is possible
- Use Auditd or eBPF
- Zircolite for flash analysis

MACOS

- Memory scanning in dead for now
  - So attackers doesn't need to use fluctuation
- But it could change...
  - Then we'll be ready

Blogpost & Repository

- https://github.com/OWNsecurity/Memory-Fluctuation
- https://www.own.security/en/ressources/blog/artefacts-de-fluctuation-memoire

References

- https://i.blackhat.com/Asia-23/AS-23-Uhlmann-You-Can-Run-But-You-Cant-Hide.pdf
- https://www.blackhillsinfosec.com/avoiding-memory-scanners
- https://github.com/mgeeky/ShellcodeFluctuation
- https://karol-mazurek.medium.com/list/snakeapple-50baea541374

OWN

+33 (0) 805 690 234

contact@**own.security**

WWW.**OWN.SECURITY**