

Bugs on the Windshield: Fuzzing the Windows Kernel



@YoavAlon



@NetanelBenSimon



Who Are We?

Yoav Alon (@yoavalon)

CTO @ Orca Security

MSRC MVSR 2019

Netanel Ben Simon (@NetanelBenSimon)

Researcher @ Check Point Research

MSRC MVSR 2019

We ❤️ Fuzzing

Motivation

We had a successful fuzzing campaign against **binary format** parsers

- Adobe Reader & MS Edge

WinAFL has been successfully used to identify bugs in Windows software, such as

- [Adobe] CVE-2018-4985, CVE-2018-5063, CVE-2018-5064, CVE-2018-5065, CVE-2018-5068, CVE-2018-5069, CVE-2018-5070, CVE-2018-12754, CVE-2018-12755, CVE-2018-12764, CVE-2018-12765, CVE-2018-12766, CVE-2018-12767, CVE-2018-12768, CVE-2018-12848, CVE-2018-12849, CVE-2018-12850, CVE-2018-12840, CVE-2018-15956, CVE-2018-15955, CVE-2018-15954, CVE-2018-15953, CVE-2018-15952, CVE-2018-15938, CVE-2018-15937, CVE-2018-15936, CVE-2018-15935, CVE-2018-15934, CVE-2018-15933, CVE-2018-15932, CVE-2018-15931, CVE-2018-15930, CVE-2018-15929, CVE-2018-15928, CVE-2018-15927, CVE-2018-12875, CVE-2018-12874, CVE-2018-12873, CVE-2018-12872, CVE-2018-12871, CVE-2018-12870, CVE-2018-12869, CVE-2018-12867, CVE-2018-12866, CVE-2018-12865, CVE-2018-12864, CVE-2018-12863, CVE-2018-12862, CVE-2018-12861, CVE-2018-12860, CVE-2018-12859, CVE-2018-12857, CVE-2018-12839 - found by Yoav Alon and Netanel Ben-Simon from Check Point Software Technologies

<https://research.checkpoint.com/2018/50-adobe-cves-in-50-days/>

Motivation

- We wanted a bigger challenge
 - fuzzing Windows Kernel seemed hard enough
- We can build a **full chain**

Obligatory slides - What is fuzzing?

A method for automatic software testing

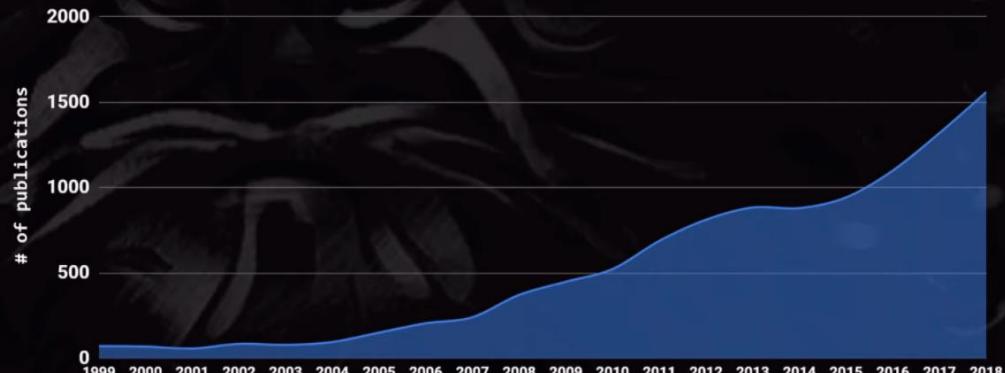
- Inputs: target software + CPU time
- Outputs: bugs

Example of a basic fuzzer:



```
$ ./test_program < /dev/urandom
```

Active area of research:



INTRODUCING FUZZING AT OFFENSIVECON

Ain't nobody got time for that

4

הציג בכנס אבטחה של HAZARALAFEL.COM

1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018

<https://www.youtube.com/watch?v=4BkAxMfHSzI> - @MShudrak



We have experience with **AFL**

We have experience with **AFL**

Can we use **AFL** to attack the Windows kernel?

kAFL

AFL with a “k”

kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels

Sergej Schumilo
Ruhr-Universität Bochum

Cornelius Aschermann
Ruhr-Universität Bochum

Robert Gawlik
Ruhr-Universität Bochum

Sebastian Schinzel
Münster University of Applied Sciences

Thorsten Holz
Ruhr-Universität Bochum

kAFL

AFL with a “k”



kAFL

AFL with a “k”

Trophies

- Linux keyctl null pointer dereference ([CVE-2016-8650](#))
- Linux EXT4 memory corruption
- Linux EXT4 denial of service
- macOS APFS memory corruption ([CVE-2017-13800](#))
- macOS HFS memory corruption ([CVE-2017-13830](#))



kAFL

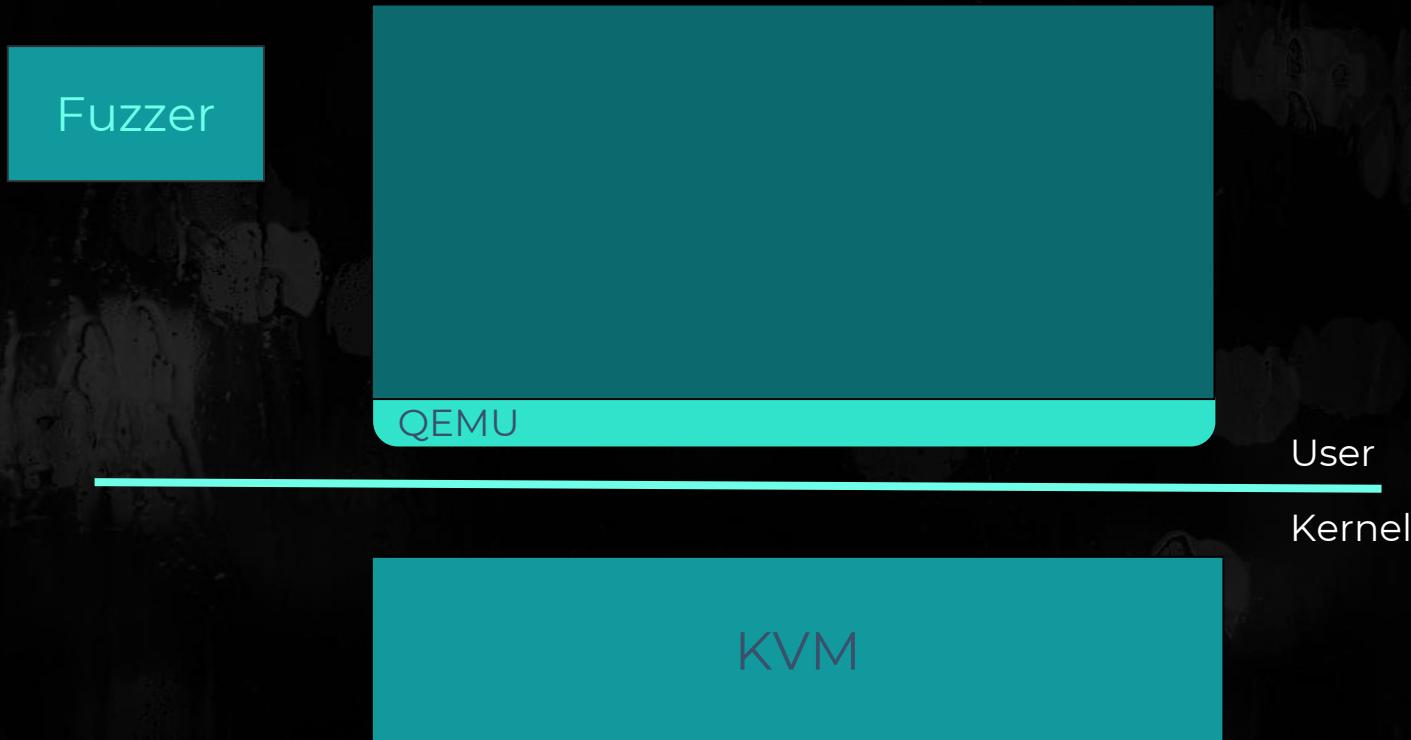
Architecture

Fuzzer

User

Kernel

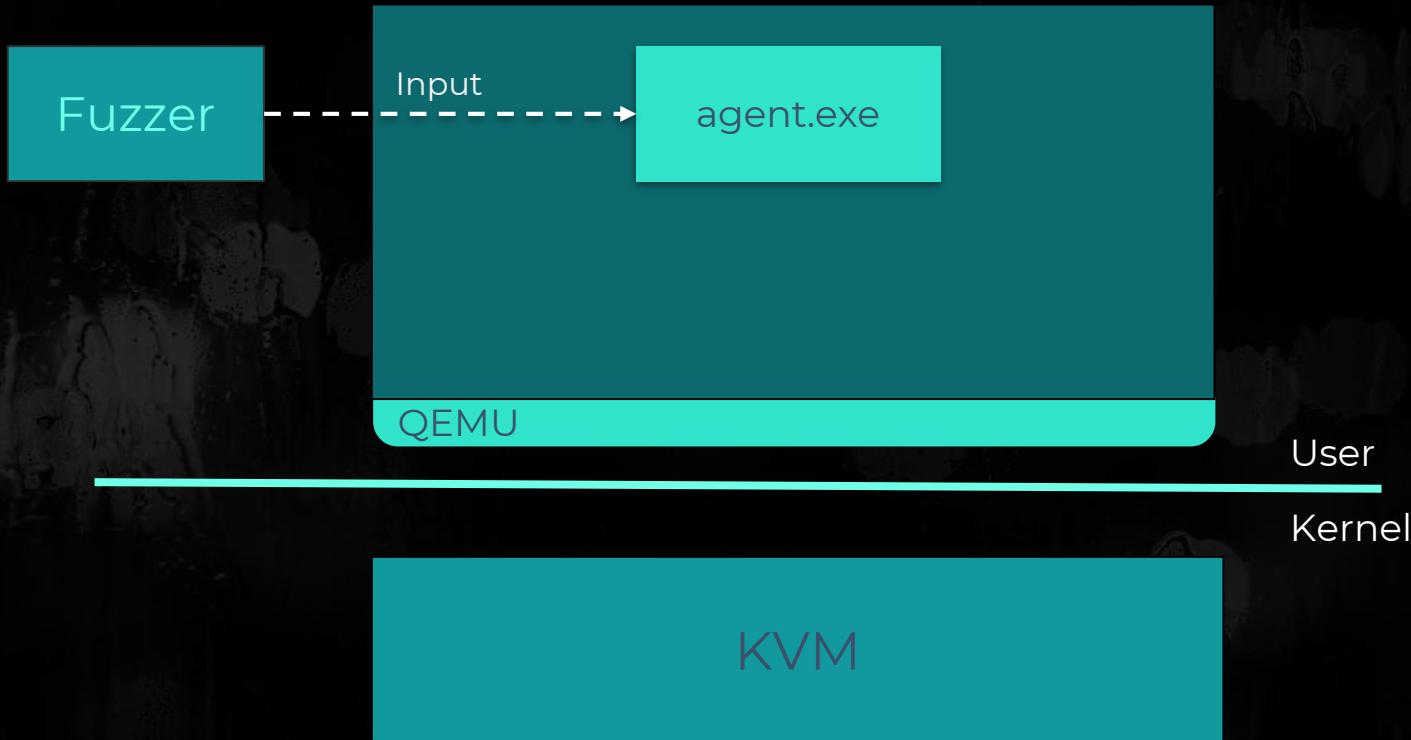
kAFL Architecture



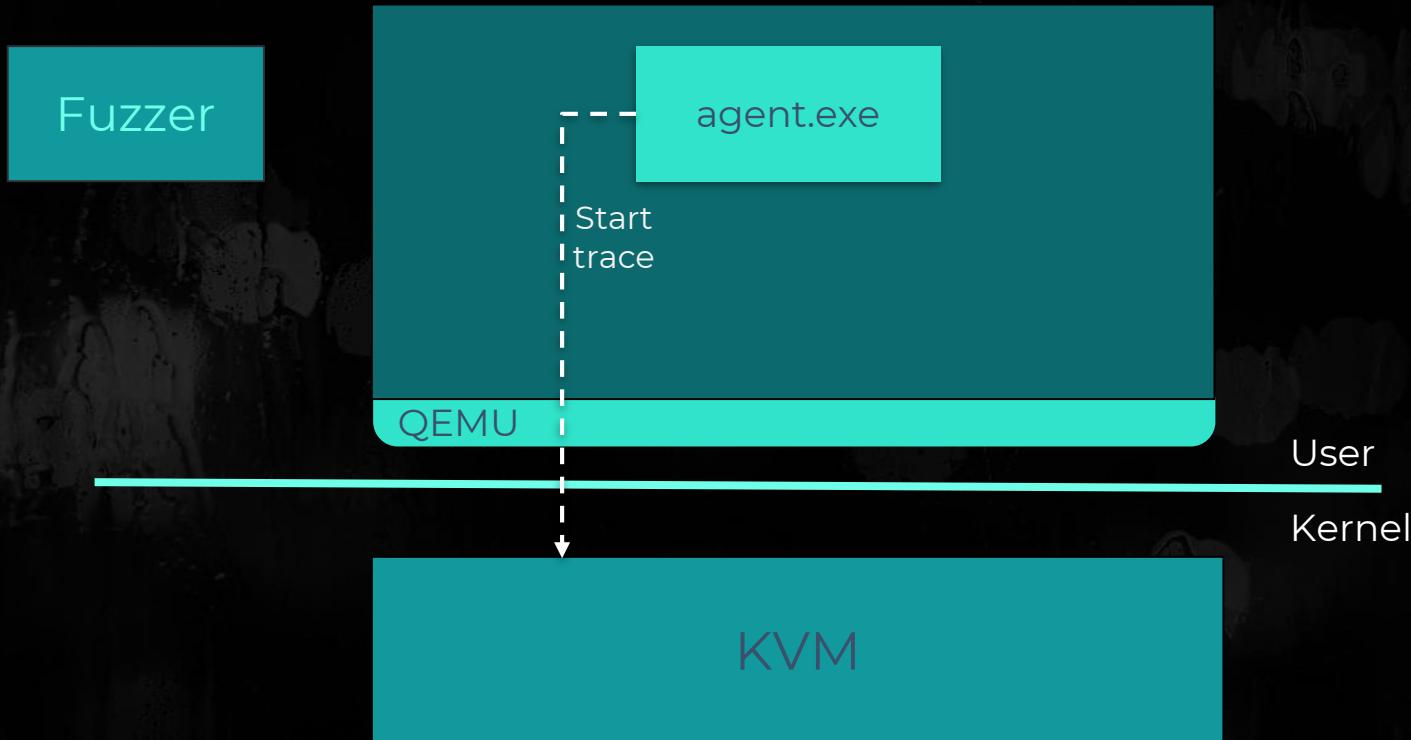
kAFL Architecture



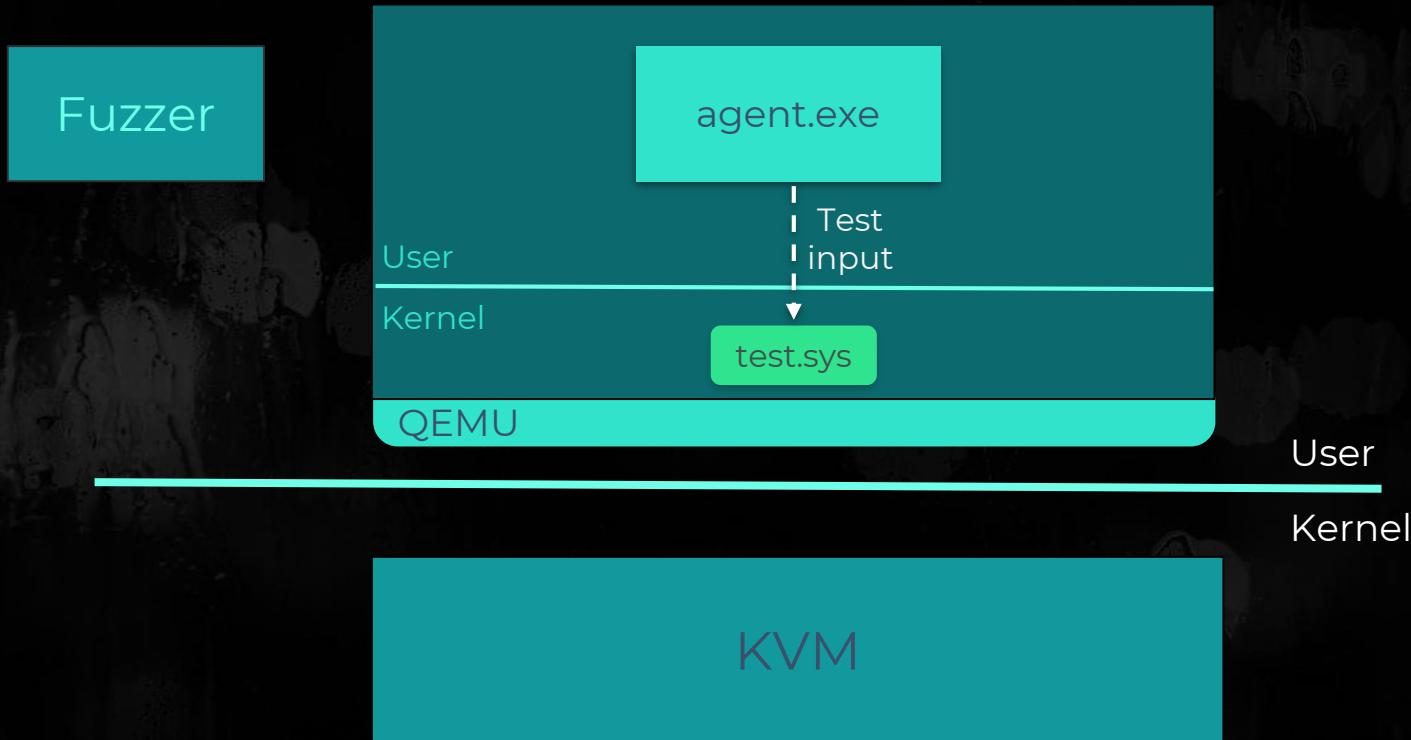
kAFL Architecture



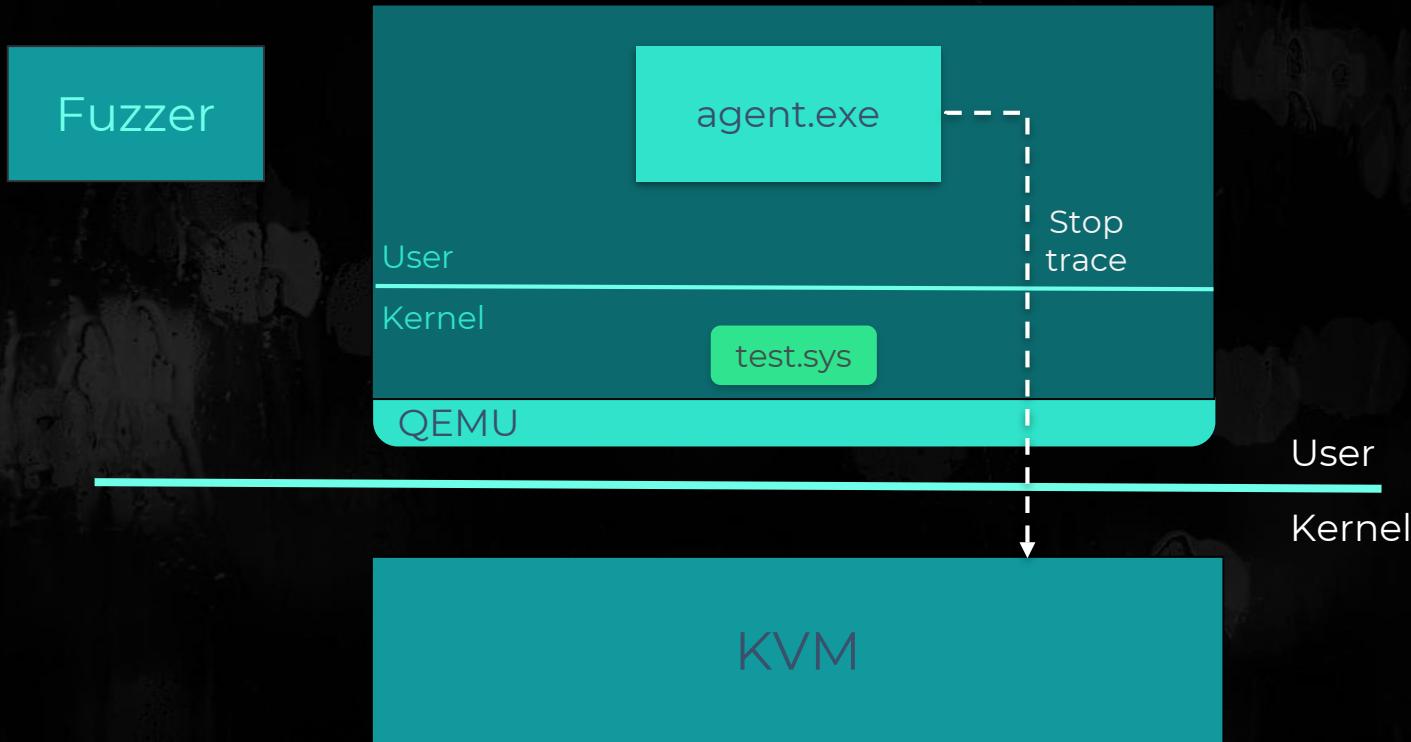
kAFL Architecture



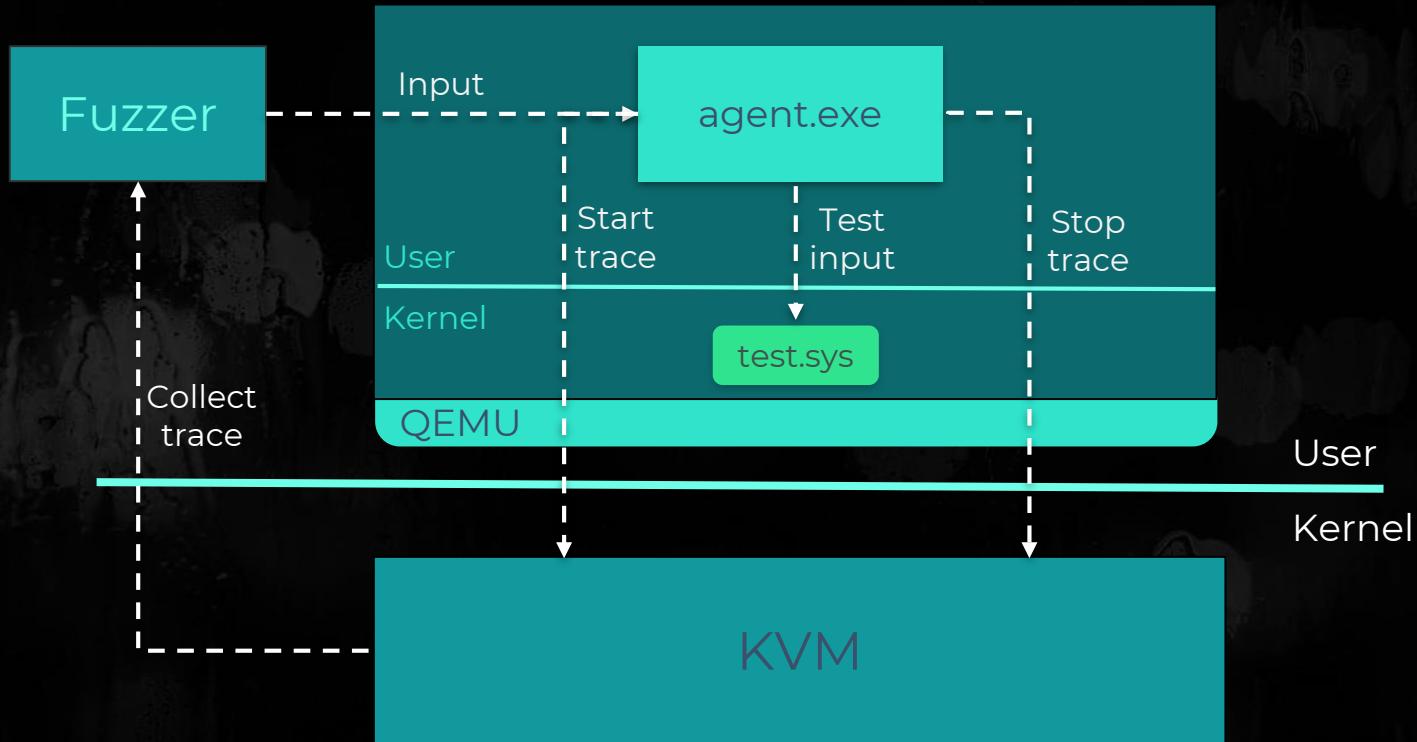
kAFL Architecture



kAFL Architecture



kAFL Architecture



kAFL

Coverage - Intel Processor Trace

Low-overhead **hardware** execution tracing feature

The trace information is written in a compressed form to **physical memory**

kAFL uses a fast decoder to generate full traces

kAFL

How to detect crashes?



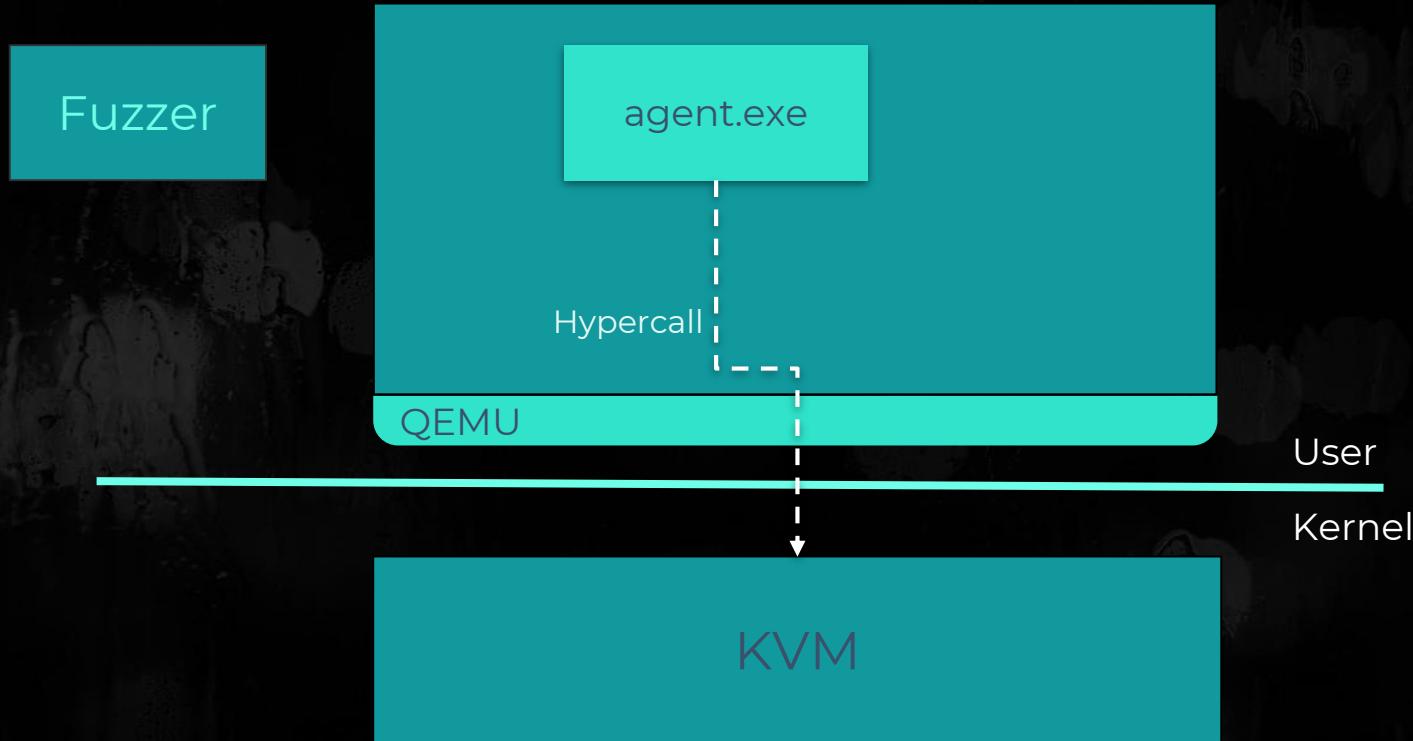
kAFL

How to detect crashes?



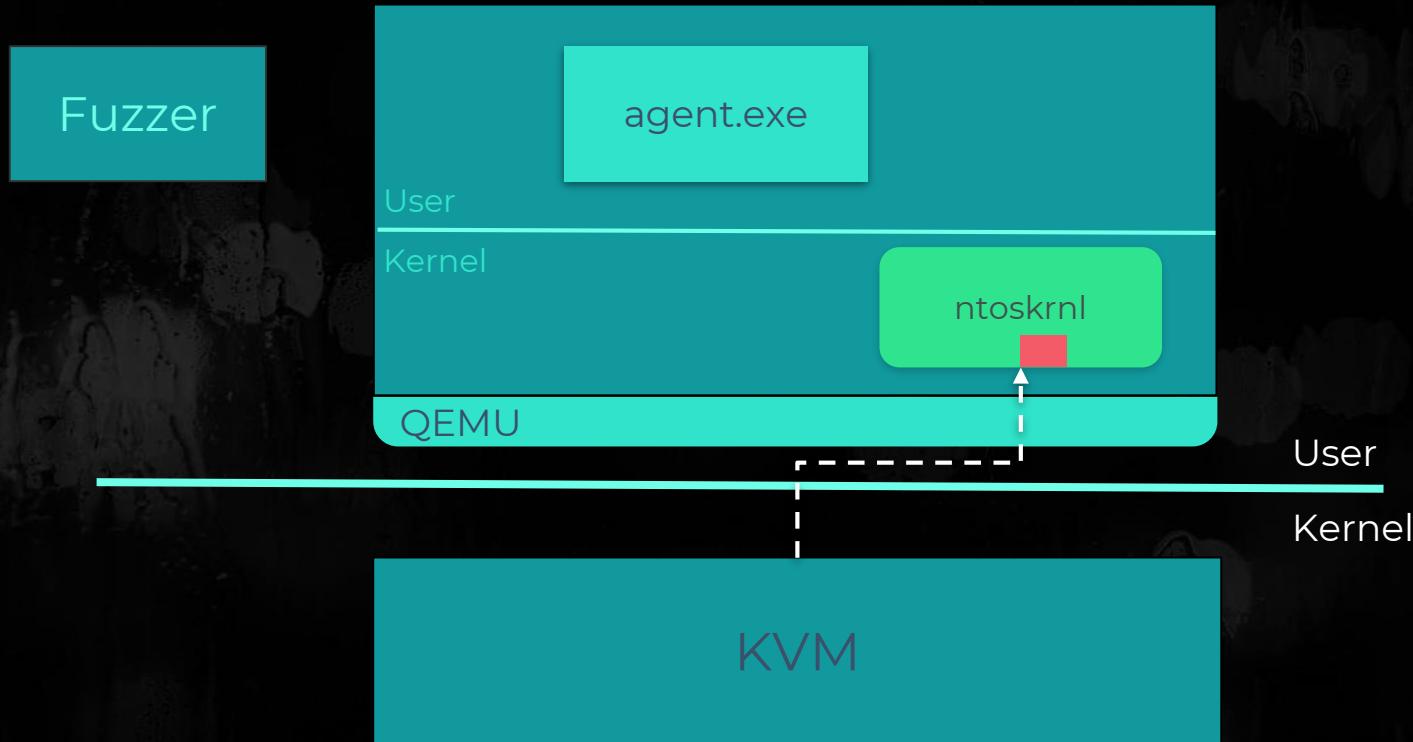
kAFL

How to detect crashes?



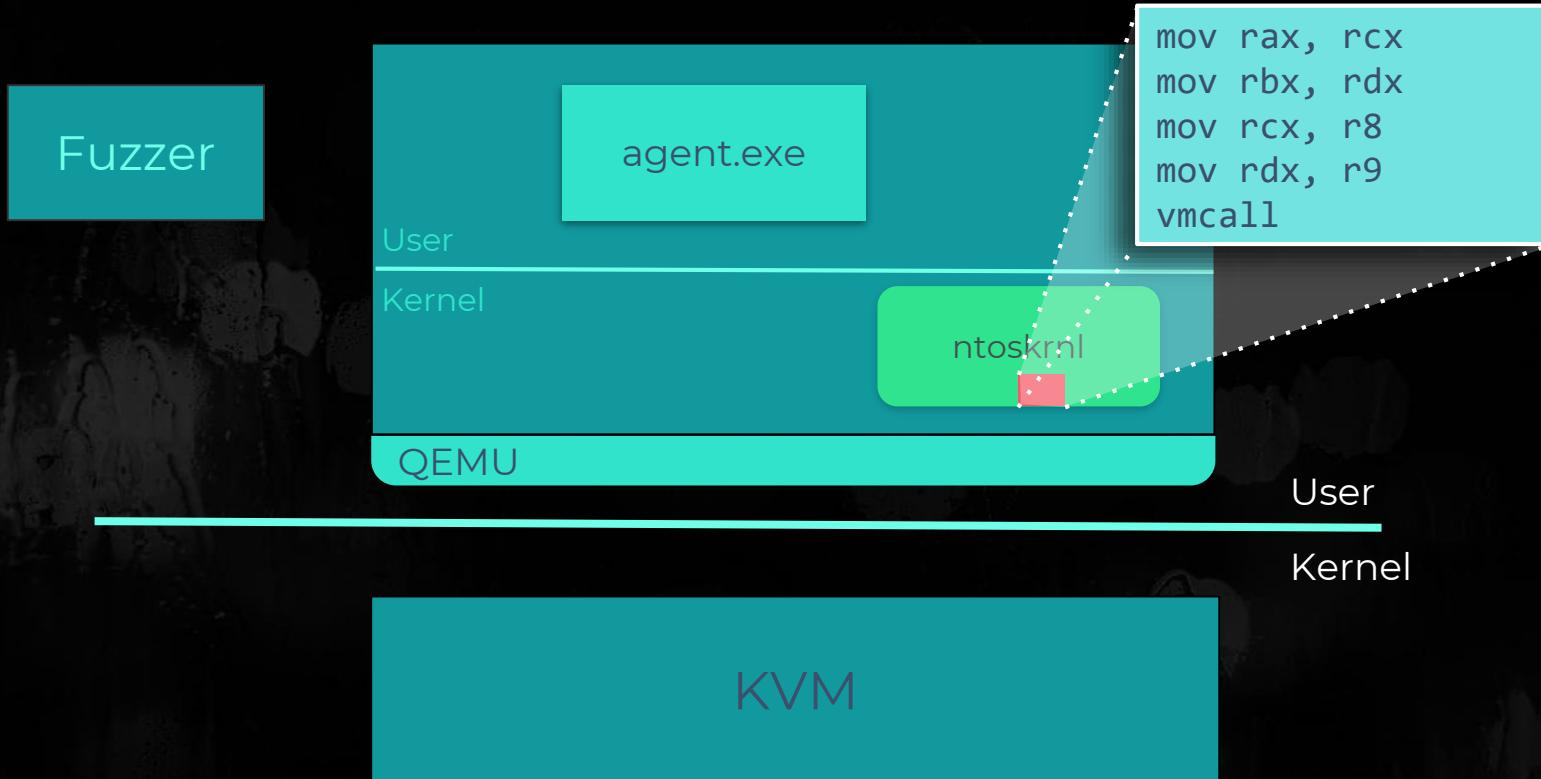
kAFL

How to detect crashes?



kAFL

How to detect crashes?



kAFL

How to detect crashes?

Fuzzer



Your PC ran into a problem and needs to restart. We'll restart for you.

For more information about this issue and possible fixes, visit <https://www.windows.com/stopcode>

If you call a support person, give them this info:
Stop code: SYSTEM SERVICE EXCEPTION

QEMU

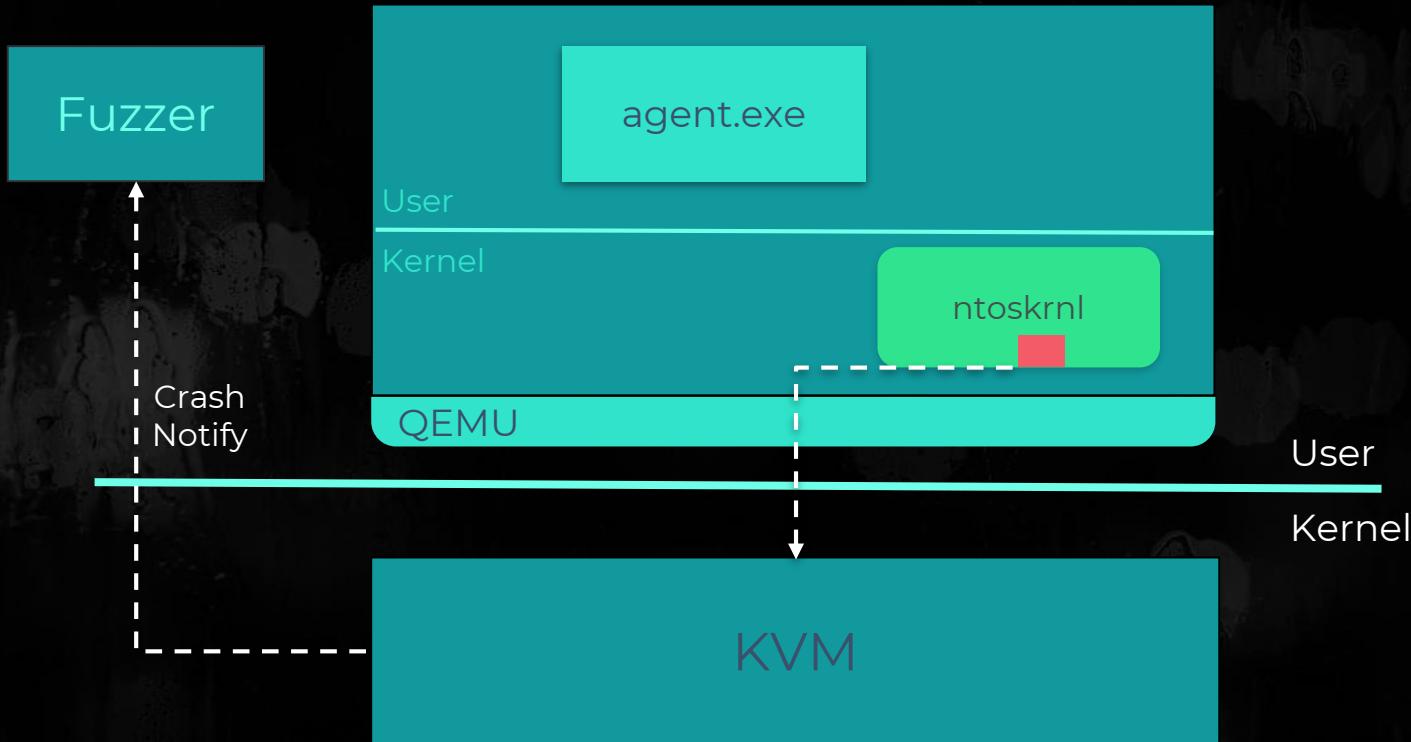
User

Kernel

KVM

kAFL

How to detect crashes?



kAFL harness

```
kAFL_hypercall(HYPERCALL_KAFL_SUBMIT_CR3, 0);

kAFL_hypercall(HYPERCALL_KAFL_GET_PAYLOAD, payload_buffer);

while (TRUE) {
    kAFL_hypercall(HYPERCALL_KAFL_NEXT_PAYLOAD, 0);

    kAFL_hypercall(HYPERCALL_KAFL_ACQUIRE, 0);

    DeviceIoControl(vuln_device_handle, ...);

    kAFL_hypercall(HYPERCALL_KAFL_RELEASE, 0);
}
```

kAFL

Dashboard

* x86-64 kernel AFL * (1 Processes)			
Runtime: 000:00:26:23	Performance: [██████████] 659 t/s		
Last Path: 000:00:07:33	Fuzzing Technique Progress		
Bitmap: 01.0b/ 00.0%	Bitflipping: [*****] 778		
Blacklisted: 0/ 0	Arithmetic: [*****] 10K		
Cycles: 22	Interesting: [*****] 1.4K		
Level: 7/ 8	Havoc: [*****] 4.7K		
Favs: 8/ 11 (72.7%)	Splicing: [*] 4.7K		
Pending: 0/ 0	Panic: 78 (2)		
Skipped: 0/ 0	CPU: 02.7%		
Payload-Size: 34B	KASan: 0 (0)	RAM: 02.7%	
Total: 960K	Timeout: 0 (0)	SPLICING	

kAFL

What to attack with kAFL?

Good targets for AFL/kAFL

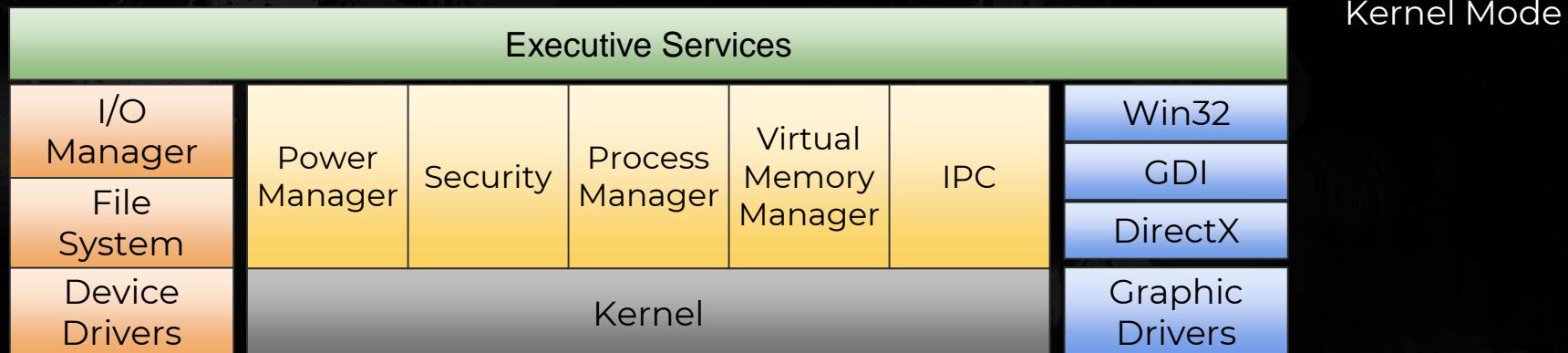
- Fast (>100 iter/s)
- Parsers, especially for binary formats

kAFL

What to attack with kAFL?

Good targets for AFL/kAFL

- Fast (>100 iter/s)
- Parsers, especially for binary formats

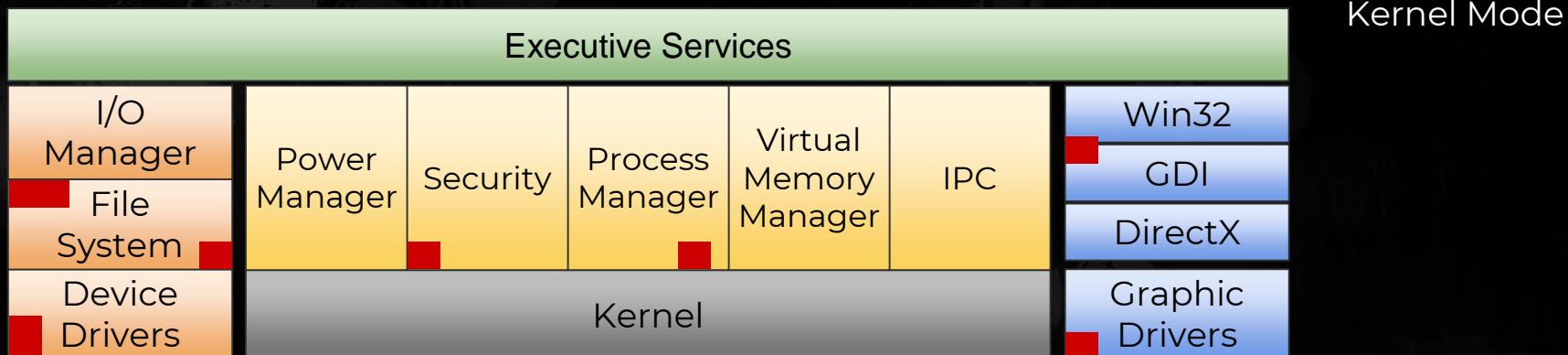


kAFL

What to attack with kAFL?

Good targets for AFL/kAFL

- Fast (>100 iter/s)
- Parsers, especially for binary formats



Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize          = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc     = DefWindowProc;
    WindowClass.lpszClassName   = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
}
```

Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize          = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc     = DefWindowProc;
    WindowClass.lpszClassName   = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
}
```

Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize          = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc     = DefWindowProc;
    WindowClass.lpszClassName   = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
}
```

Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize          = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc     = DefWindowProc;
    WindowClass.lpszClassName   = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
}
```

Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize          = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc     = DefWindowProc;
    WindowClass.lpszClassName   = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
}
```

Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize          = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc     = DefWindowProc;
    WindowClass.lpszClassName   = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
}
```

Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize          = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc     = DefWindowProc;
    WindowClass.lpszClassName   = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
}
```

Typical kernel bug: CVE-2018-0744

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize          = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc     = DefWindowProc;
    WindowClass.lpszClassName   = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, CW_USEDEFAULT,
                           0, 128, 128, NULL, NULL, NULL, NULL);
}
```

kAFL

syscall fuzzing

VS

kAFL

syscall fuzzing



VS

100
001

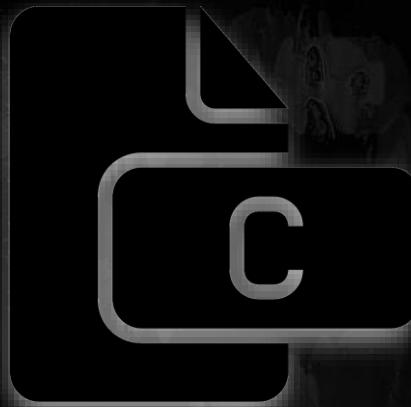
```
00000000  15 b7 2c 56 4f 3c 8f 74  9e d7 8a 10 ca 63 f7 81 |...,VO<.t.....c..|
00000010  c7 11 0b 11 cc 1e 81 15  4f e6 0f 06 b8 de 3a 52 |.....0.....:R|
00000020  a2 0a 2e 00 75 9e d5 2e  51 70 9e 5e 13 ae c8 ba |....u...Qp.^....|
00000030  d3 27 37 dc 14 d0 f3 e8  20 c0 e8 94 69 b2 33 4f |.'7..... .i.30|
00000040  e7 4c dd 0b f3 4c f0 47  d3 09 c9 0b 95 83 37 70 |.L...L.G.....7p|
00000050  bf 7a d0 2a 88 d0 2c ab  72 3a 95 56 86 69 88 55 |.z.*...,r:.V.i.U|
```

kAFL

syscall fuzzing



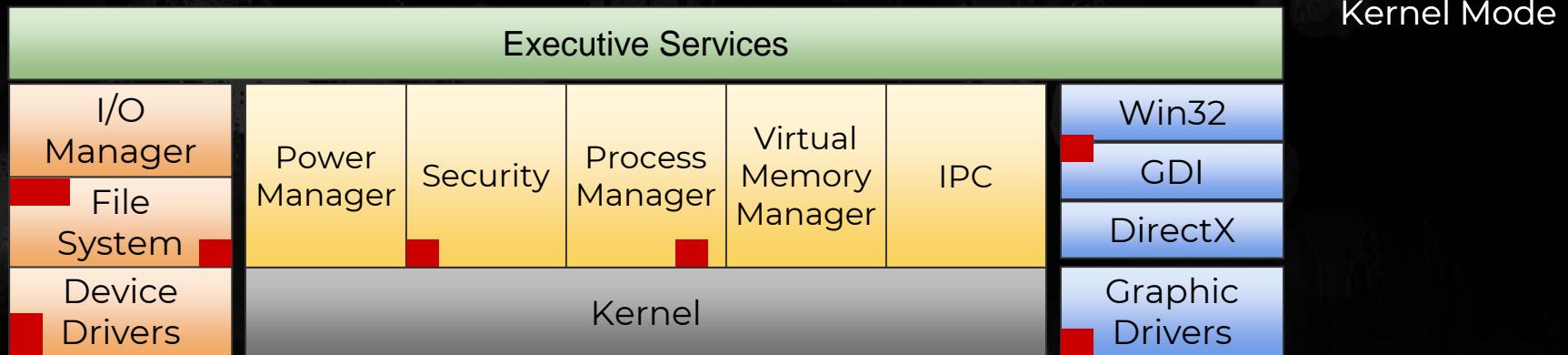
VS



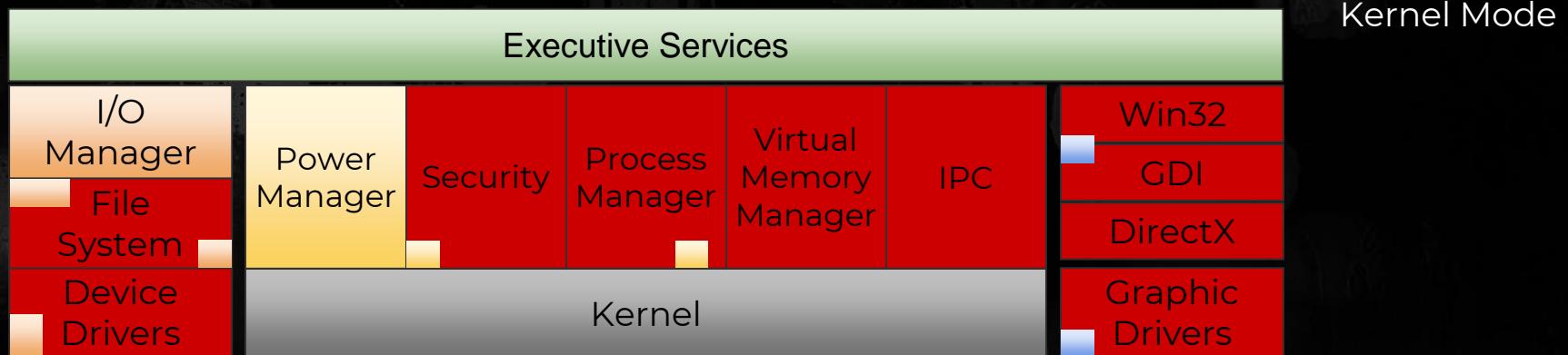
```
00000000 15 b7 2c 56 4f 3c 8f 74 9e d7 8a 10 ca 63 f7 81 |...,VO<.t.....c..|
00000010 c7 11 0b 11 cc 1e 81 15 4f e6 0f 06 b8 de 3a 52 |.....0.....:R|
00000020 a2 0a 2e 00 75 9e d5 2e 51 70 9e 5e 13 ae c8 ba |....u...Qp.^....|
00000030 d3 27 37 dc 14 d0 f3 e8 20 c0 e8 94 69 b2 33 4f |.'7..... .i.30|
00000040 e7 4c dd 0b f3 4c f0 47 d3 09 c9 0b 95 83 37 70 |.L...L.G.....7p|
00000050 bf 7a d0 2a 88 d0 2c ab 72 3a 95 56 86 69 88 55 |.z.*...,r:.V.i.U|
```

```
void main() {
    fd = open("/proc/self/mem");
    lseek(fd, 0x13337, SEEK_SET);
    read(fd, buffer, _SC_PAGE_SIZE)
    close(obj);
}
```

Kernel attack surface using kAFL

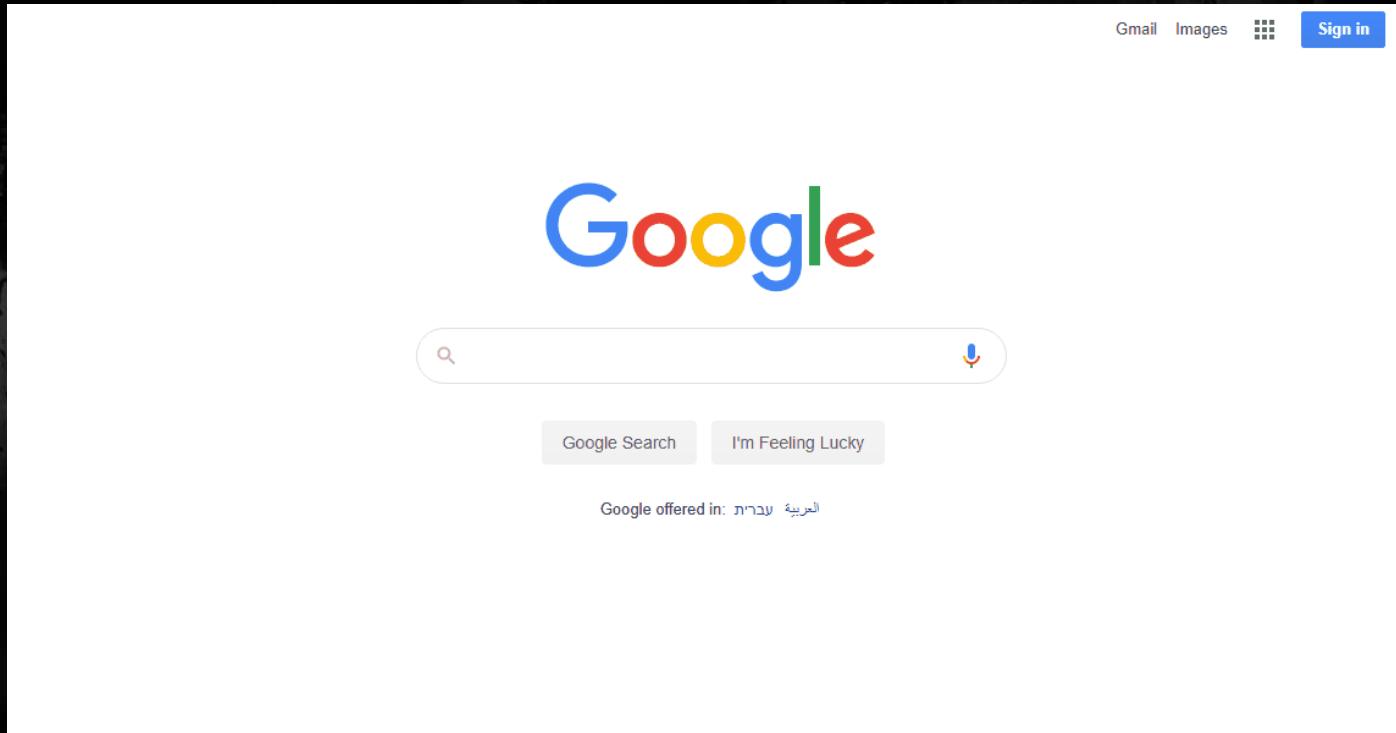


Kernel attack surface using syscall fuzzer



So we decided to dump kAFL
And move to a **syscall** fuzzer

So we decided to dump kAFL
And move to a syscall fuzzer



Syzkaller

Coverage-guided structure-aware kernel fuzzer

Syzkaller

Coverage-guided structure-aware kernel fuzzer

- “AKA” Smart syscall fuzzer

Syzkaller

Coverage-guided structure-aware kernel fuzzer

- “AKA” Smart syscall fuzzer
- OS (Linux, BSD, Fuchsia, ...)



Syzkaller

Coverage-guided structure-aware kernel fuzzer

- “AKA” Smart syscall fuzzer
- OS (Linux, BSD, Fuchsia, ...)
- Machines (QEMU, GCE, Mobile phones)



Syzkaller

Coverage-guided structure-aware kernel fuzzer

- “AKA” Smart syscall fuzzer
- OS (Linux, BSD, Fuchsia, ...)
- Machines (QEMU, GCE, Mobile phones)
- Architecture (x86-64, aarch64, ...)



Syzkaller

The **hardest working researcher** in the Linux Kernel community

Syzkaller

The **hardest working researcher** in the Linux Kernel community

Found more than **3700** bugs in the linux kernel (!)

Syzkaller

The **hardest working researcher** in the Linux Kernel community

Found more than **3700** bugs in the linux kernel (!)

CVE-2019-2215

fixed (39):

Title	Repro	Count	Last	Reported	Closed	Patch
BUG: bad unlock balance in ipmr_mfc_seq_stop	C	7493	692d	771d	692d	7d3d60ef ip6mr: fix stale iterator
general protection fault in skb_release_data		1	877d	877d	805d	304b4101 ipv6: fix out of bound writes in __ip6_append_data()
KASAN: slab-out-of-bounds Read in keychord_write	syz	2	899d	897d	892d	913d980e ANDROID: keychord: Fix a slab out-of-bounds read.
KASAN: use-after-free Read in __lock_acquire	C	1161	700d	770d	692d	550c01d0 UPSTREAM: ANDROID: binder: remove waitqueue when thread exits.
KASAN: use-after-free Read in bio_copy_user iov	syz	73	859d	890d	805d	4099ac93 scsi: sg: protect accesses to 'reserved' page array
KASAN: use-after-free Read in fanout_demux_rollover	C	5	833d	854d	791d	6f7cdd4a packet: hold bind lock when rebinding to fanout hook
KASAN: use-after-free Read in parse_ipsecrequests	C	7	897d	897d	867d	3c17d418 UPSTREAM: af_key: Fix sadb_x_ipsecrequest parsing

Syzkaller

The **hardest working researcher** in the Linux Kernel community

Found more than **3700** bugs in the linux kernel (!)

CVE-2019-2215

fixed (39):

Title	Repro	Count	Last	Reported	Closed	Patch
BUG: bad unlock balance in ipmr_mfc_seq_stop	C	7493	692d	771d	692d	7d3d60ef ip6mr: fix stale iterator
general protection fault in skb_release_data		1	877d	877d	805d	304b4101 ipv6: fix out of bound writes in __ip6_append_data()
KASAN: slab-out-of-bounds Read in keychord_write	syz	2	899d	897d	892d	913d980e ANDROID: keychord: Fix a slab out-of-bounds read.
KASAN: use-after-free Read in __lock_acquire	C	1161	700d	770d	692d	550c01d0 UPSTREAM: ANDROID: binder: remove waitqueue when thread exits.
KASAN: use-after-free Read in bio_copy_user iov	syz	73	859d	890d	805d	4099ac93 scsi: sg: protect accesses to 'reserved' page array
KASAN: use-after-free Read in fanout_demux_rollover	C	5	833d	854d	791d	6f7cdd4a packet: hold bind lock when rebinding to fanout hook
KASAN: use-after-free Read in parse_ipsecrequests	C	7	897d	897d	867d	3c17d418 UPSTREAM: af_key: Fix sadb_x_ipsecrequest parsing

Test case generator/mutator

Feedback mechanism

Bug oracle

Test case generator/mutator

Feedback mechanism

Bug oracle

Syzkaller Generated Program

```
r0 = open(&(0x7f0000000000)=".file0", 0x3, 0x9)
```

Syzkaller Generated Program

```
r0 = open(&(0x7f0000000000)=".file0", 0x3, 0x9)
```

Syzkaller Generated Program

```
r0 = open(&(0x7f0000000000)=".file0", 0x3, 0x9)  
read(r0, &(0x7f0000000010), 57)  
close(r0)
```

Syscall Descriptions

exit(error_code int32)

close(fd fd)

resource fd[int32]

open(file filename, flags flags[open_flags], mode flags[open_mode]) fd

open_mode = S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, ...

read(fd fd, buf buffer[out], count len[buf])

Syscall Descriptions

```
exit(error_code int32)
```

```
close(fd fd)
```

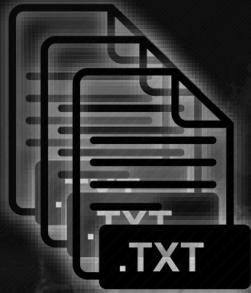
```
resource fd[int32]
```

```
open(file filename, flags flags[open_flags], mode flags[open_mode]) fd
```

```
open_mode = S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, ...
```

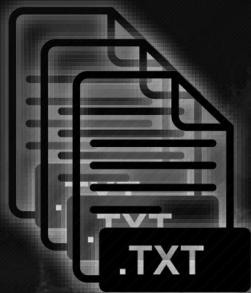
```
read(fd fd, buf buffer[out], count len[buf])
```

Generation



Generation

syz-sygen



Generation

syz-sygen



Generation

syz-sygen

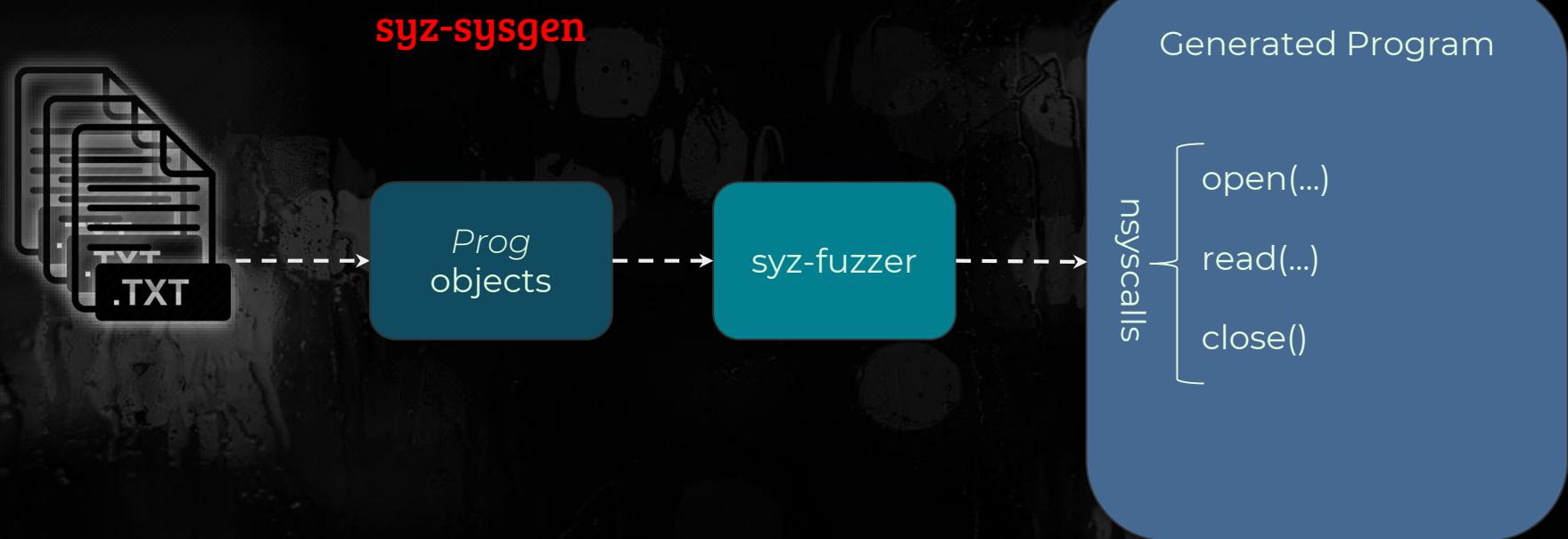


Prog
objects

syz-fuzzer



Generation



Mutations - insertCall

```
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
write(r0, &(0x7f0000000000)="41414141", 4)
close(r0)
```

Mutations - insertCall

```
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
write(r0, &(0x7f0000000000)="41414141", 4)
close(r0)
write(r0, &(0x7f0000000000)="61", 1)
```

Mutations - mutateArg

```
r0 = open(&(0x7f0000000000)=".file0", 0x3, 0x9)
write(r0, &(0x7f0000000010)="41414141", 4)
close(r0)
```

Mutations - mutateArg

```
r0 = open(&(0x7f0000000000)="./RaNdFilEnAmE", 0x3, 0x9)
write(r0, &(0x7f0000000010)="41414141", 4)
close(r0)
```

Mutations - splice

```
r0 = open(&(0x7f0000000000)=".file0", 0x3, 0x9)
write(r0, &(0x7f0000000010)="41414141", 4)
close(r0)
```

Mutations - splice

```
r0 = open(&(0x7f0000000000)=".file0", 0x3, 0x9)
r1 = open(&(0x7f0000000020)=".file1", 0x3, 0x9)
write(r0, &(0x7f0000000010)="41414141", 4)
close(r0)
```

Mutations - squashAny

```
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
write(r0, &(0x7f0000000010)="41414141", 4)
close(r0)
```

Mutations - squashAny

```
r0 = open(&(0x7f0000000000)=".file0", 0x3, 0x9)
write(r0, &(0x7f0000000010)="414130304141", 6)
close(r0)
```

Test case generator/mutator

Feedback mechanism

Bug oracle

Test case generator/mutator

Feedback mechanism

Bug oracle

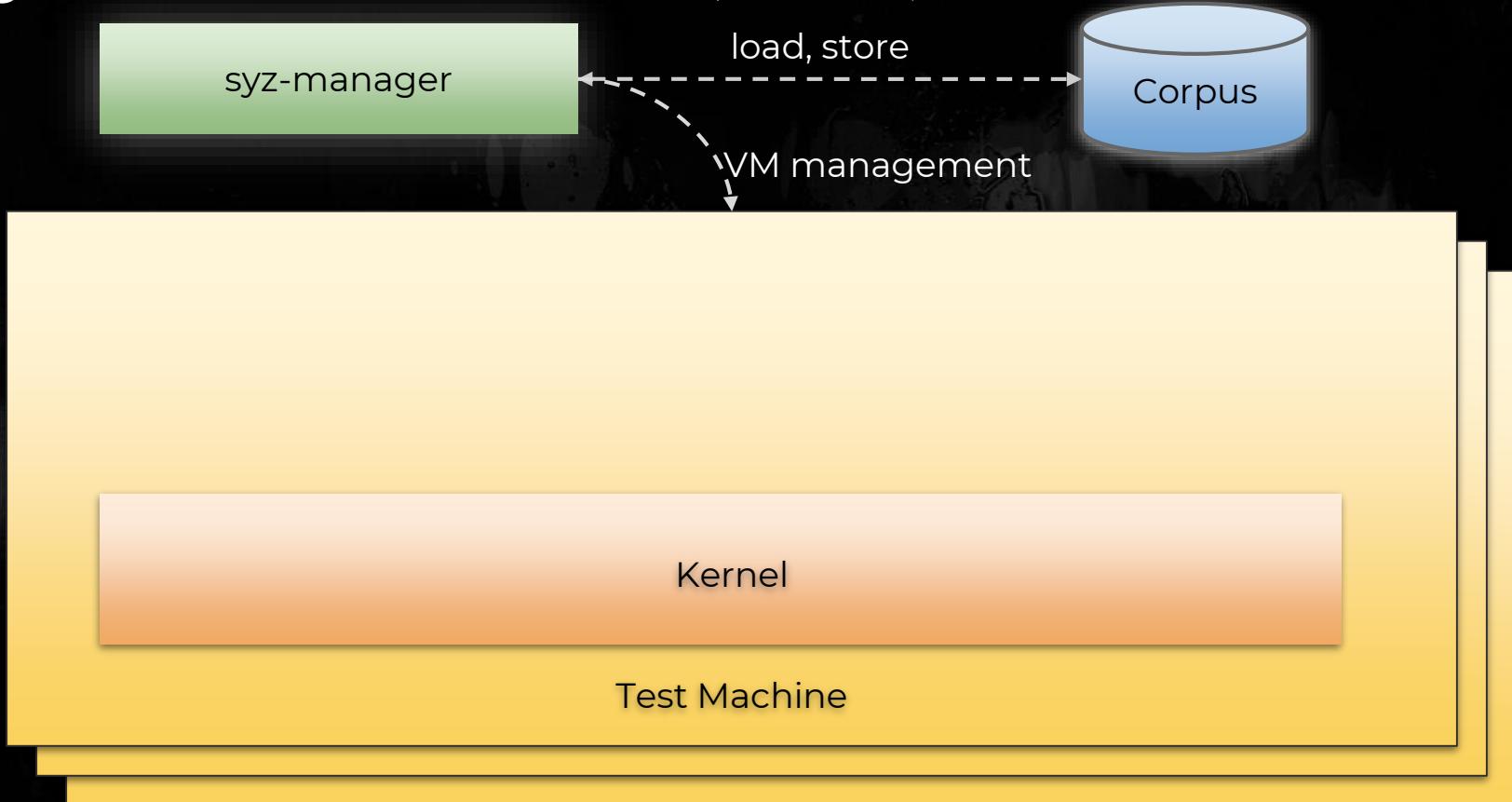
Syzkaller - Architecture (Linux)

syz-manager

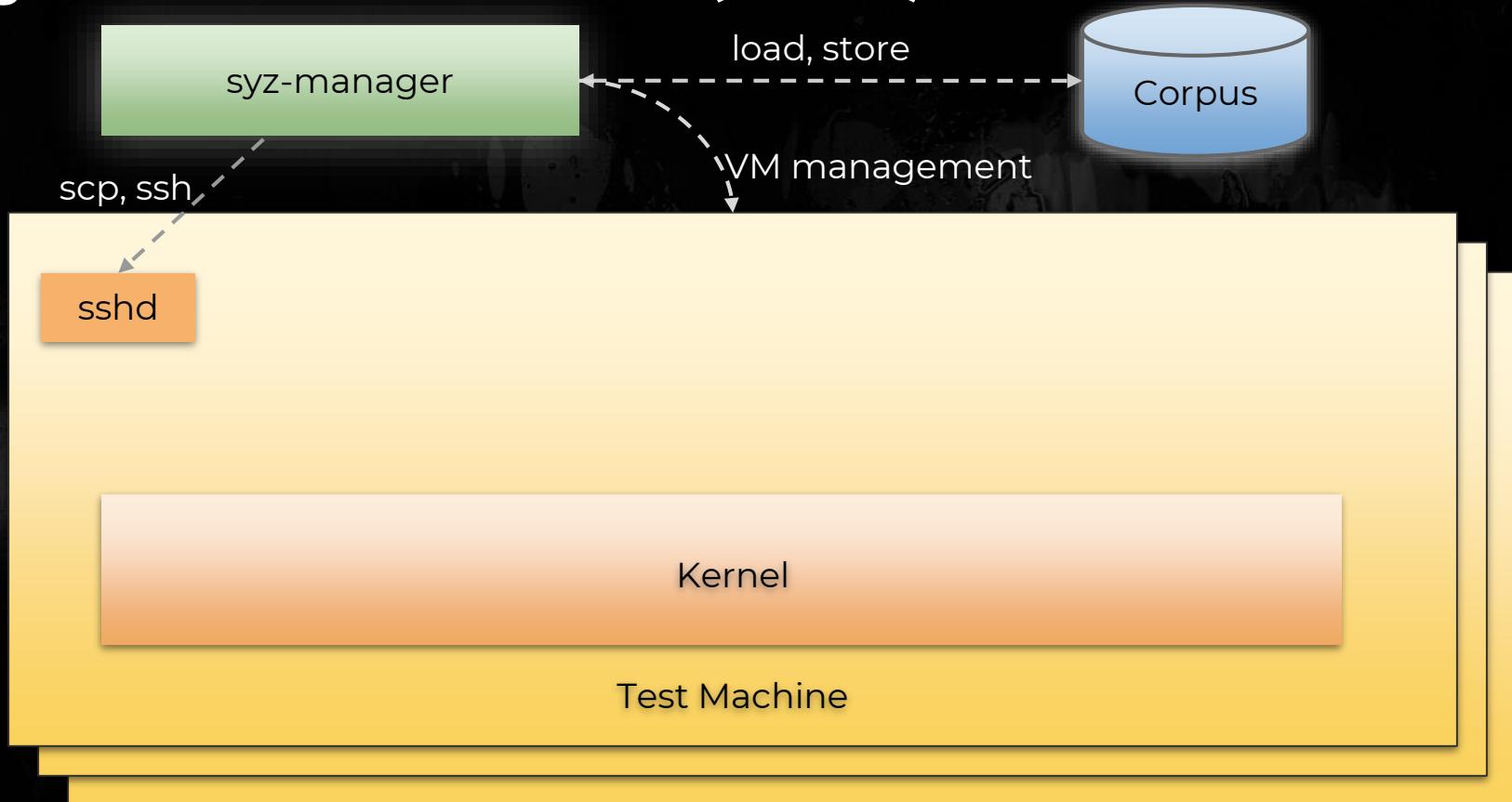
Syzkaller - Architecture (Linux)



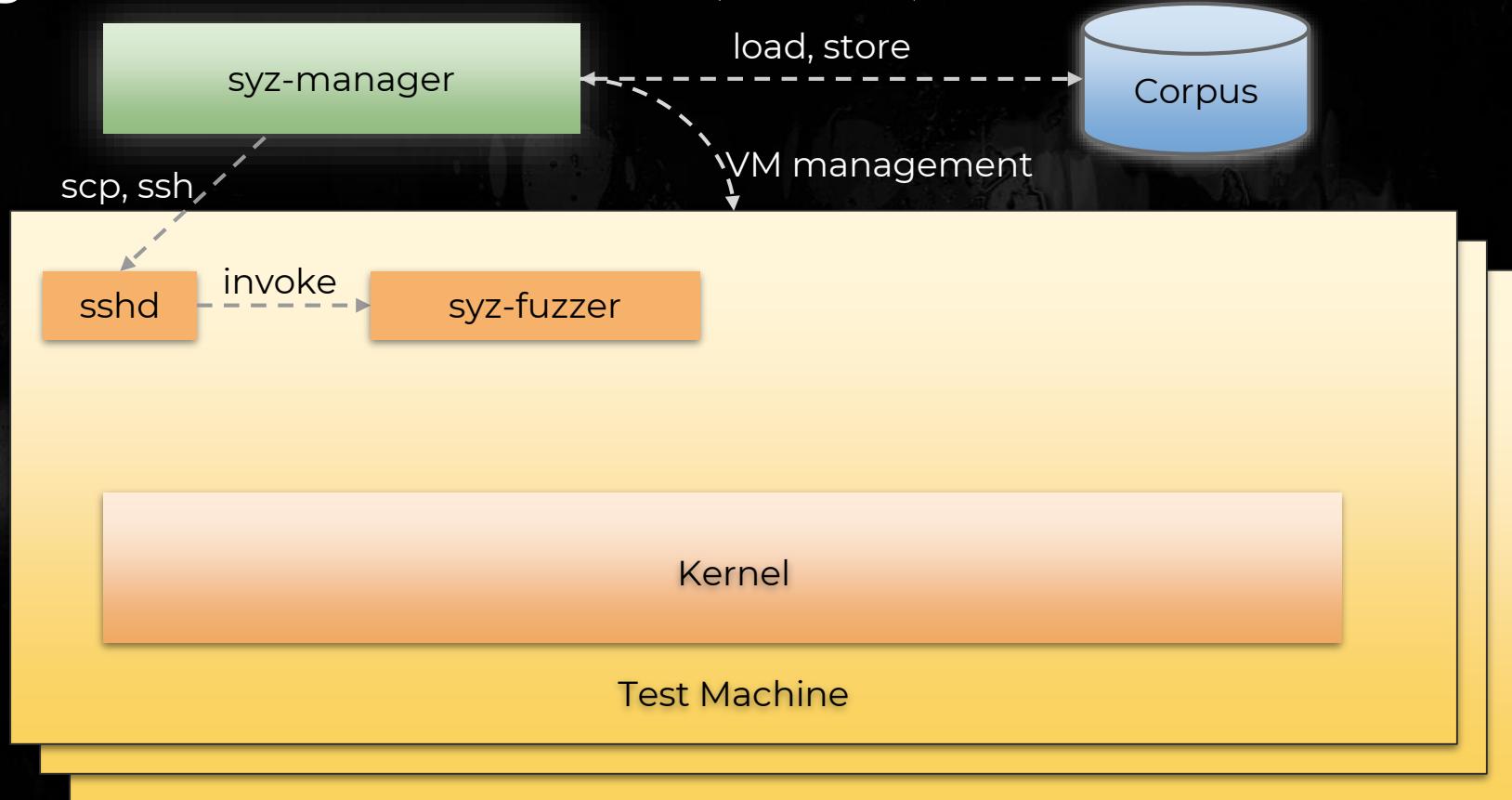
Syzkaller - Architecture (Linux)



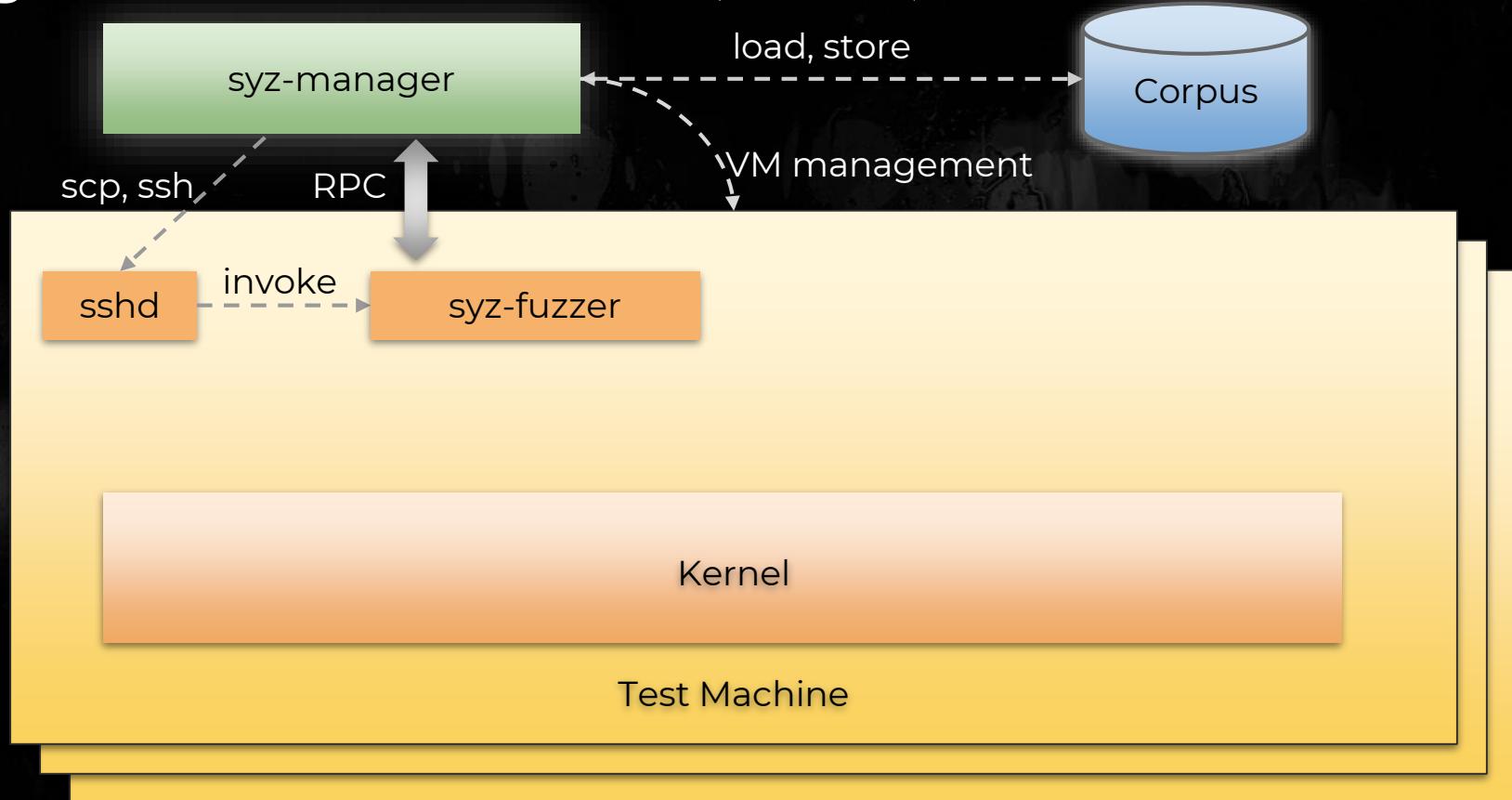
Syzkaller - Architecture (Linux)



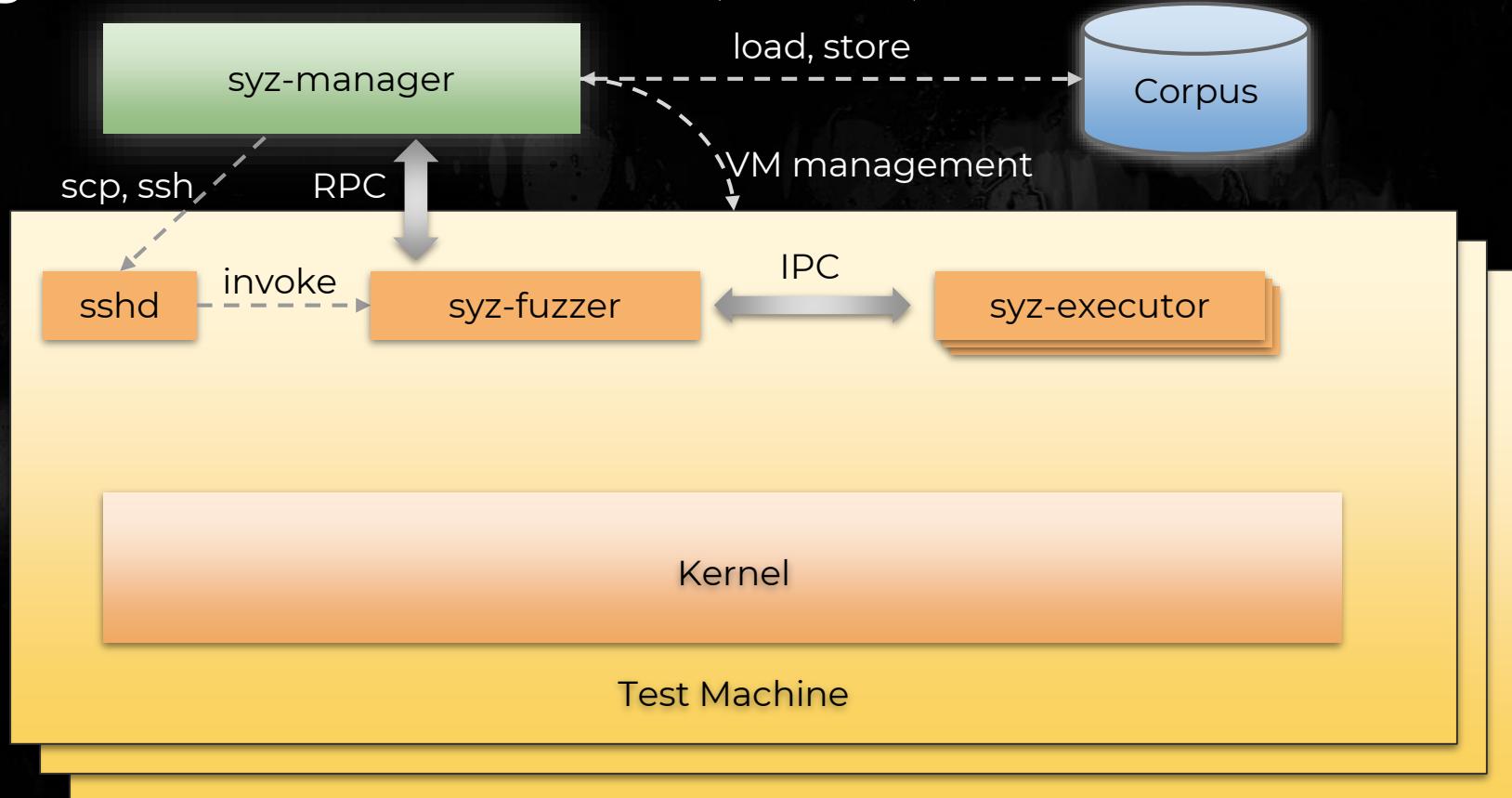
Syzkaller - Architecture (Linux)



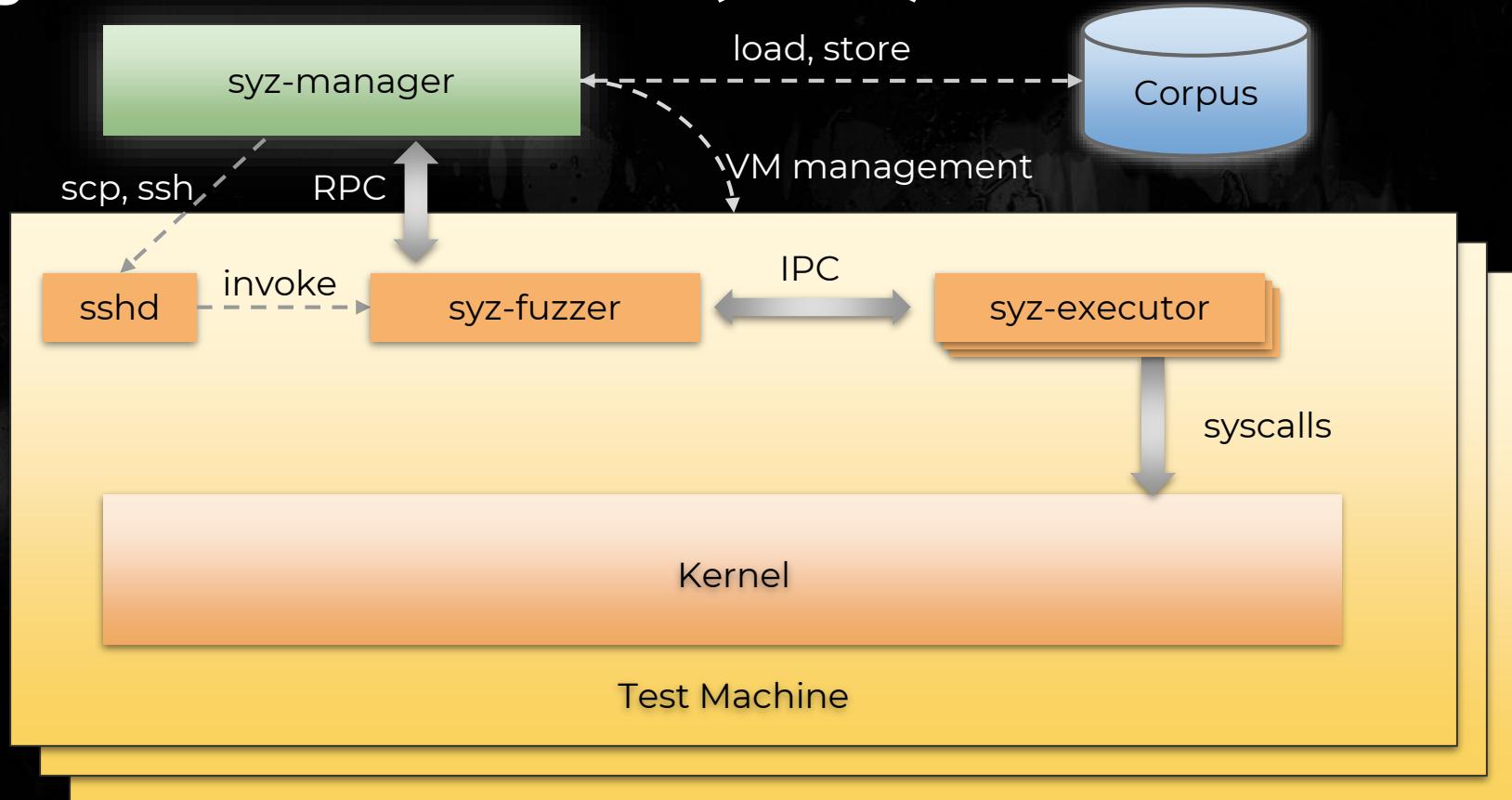
Syzkaller - Architecture (Linux)



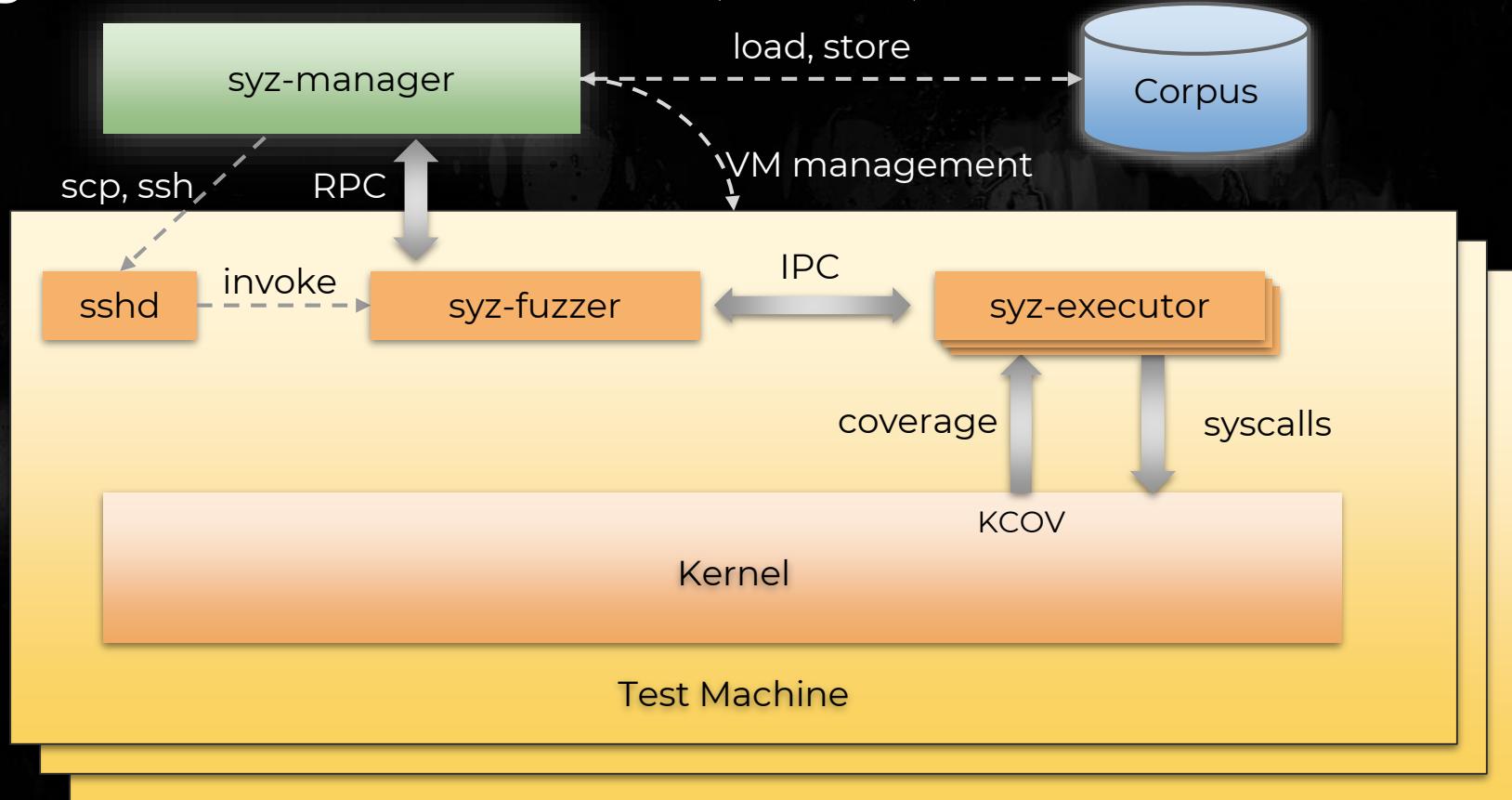
Syzkaller - Architecture (Linux)



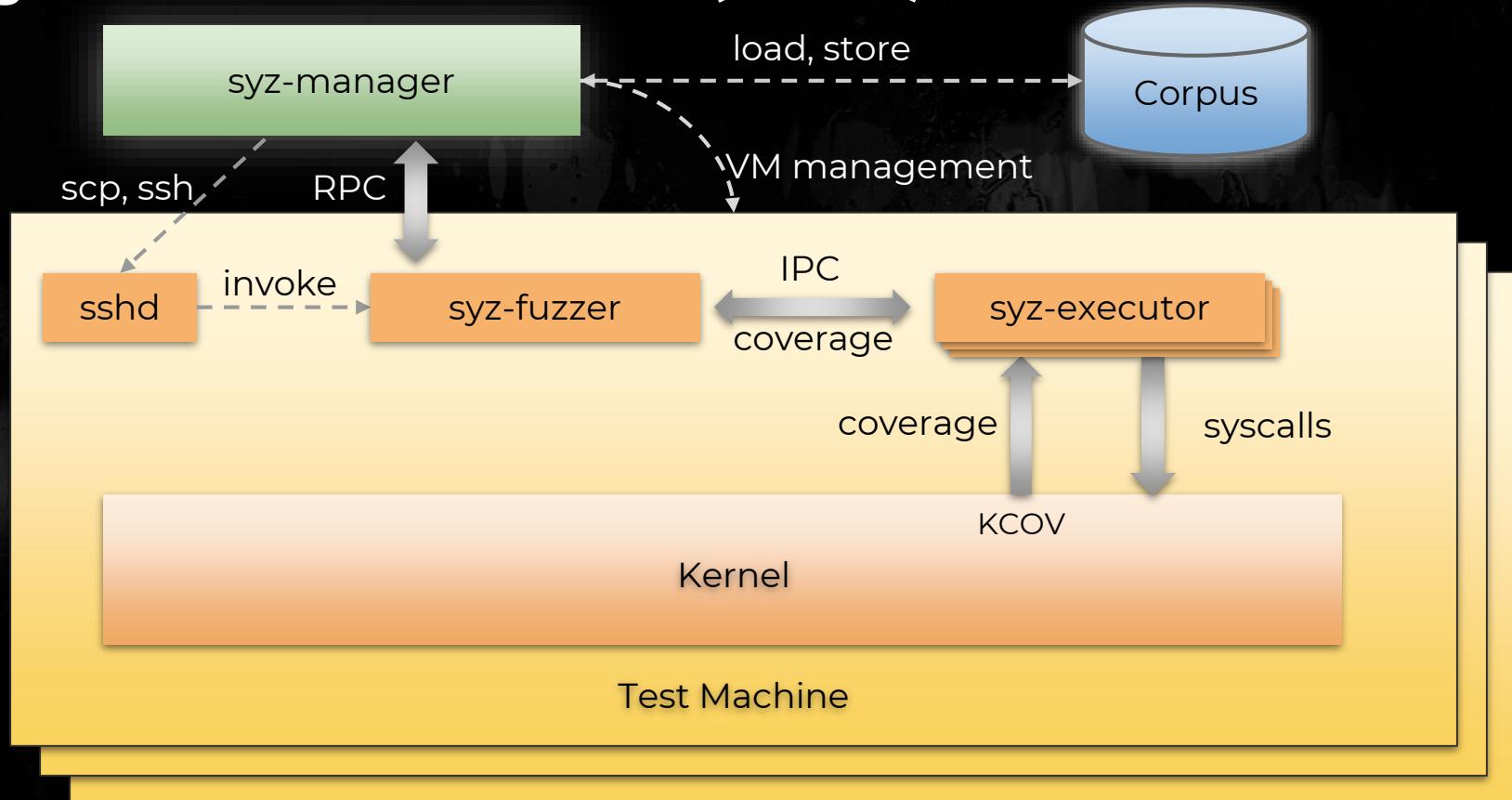
Syzkaller - Architecture (Linux)



Syzkaller - Architecture (Linux)



Syzkaller - Architecture (Linux)



KCOV

```
void kernel_func() {  
    if (...) {  
        ...  
    } else {  
        ...  
    }  
}
```

KCOV

```
void kernel_func() {  
    if (...) {  
        ...  
    } else {  
        ...  
    }  
}
```



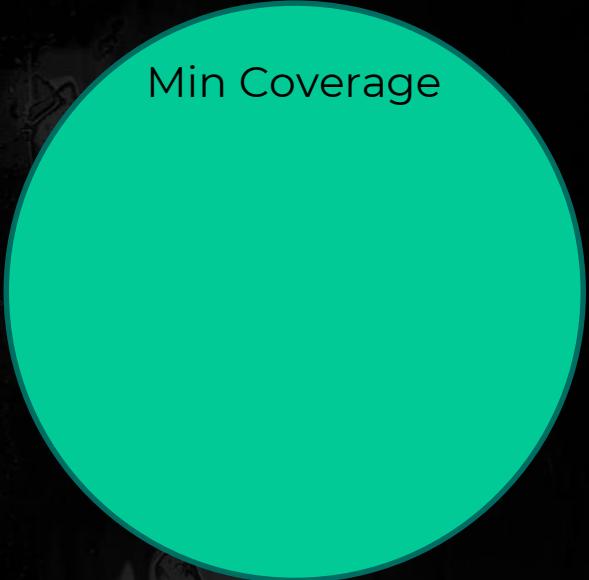
```
void kernel_func() {  
    __sanitizer_cov_trace_pc();  
    if (...) {  
        __sanitizer_cov_trace_pc();  
        ...  
    } else {  
        __sanitizer_cov_trace_pc();  
        ...  
    }  
    __sanitizer_cov_trace_pc();  
}
```

Kernel Coverage Issues

- Interrupts
- Background threads
- Nondeterminism
- Scheduler/allocator flakes

Kernel Coverage Issues

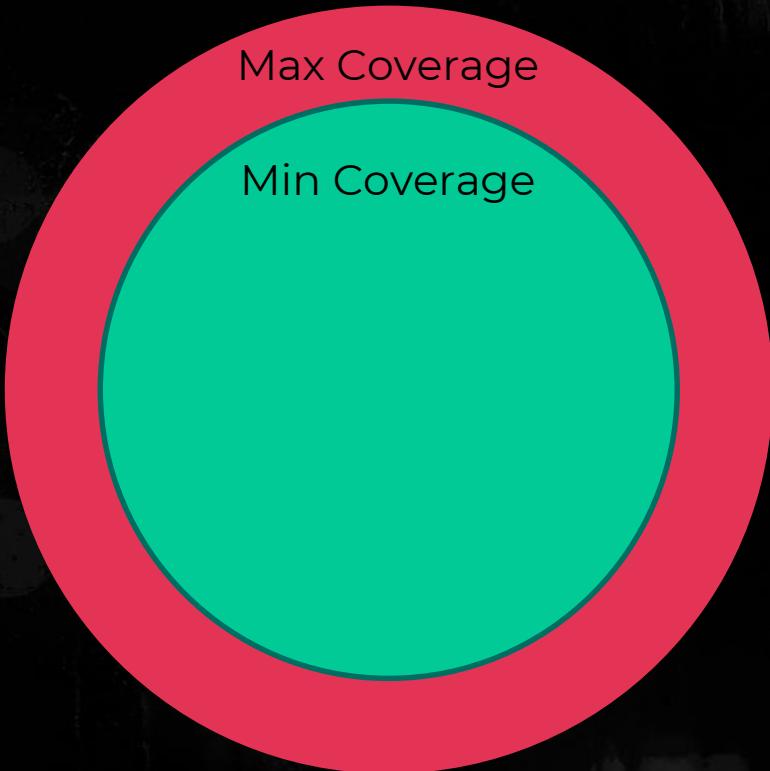
- Interrupts
- Background threads
- Nondeterminism
- Scheduler/allocator flakes



Min Coverage

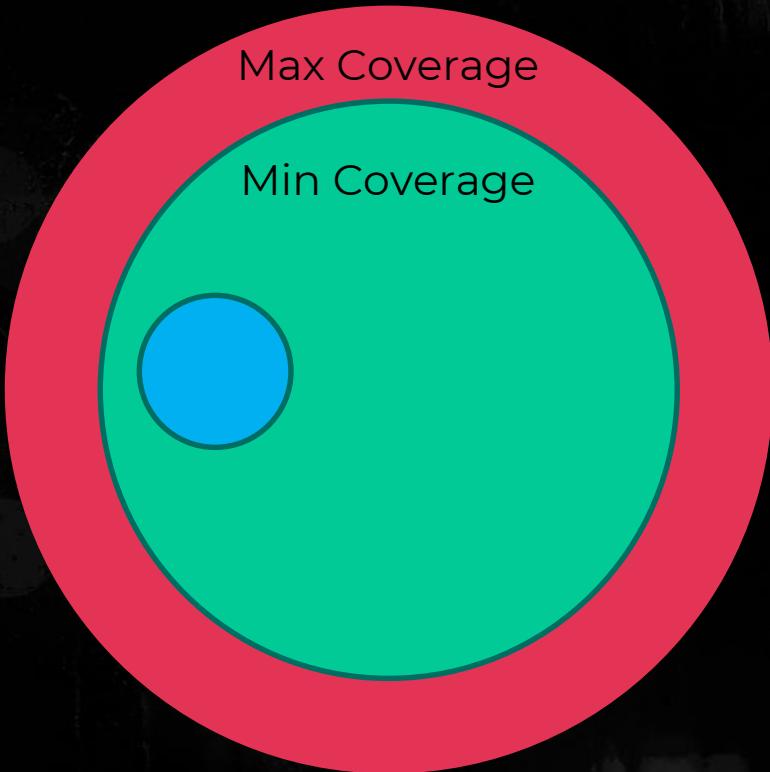
Kernel Coverage Issues

- Interrupts
- Background threads
- Nondeterminism
- Scheduler/allocator flakes



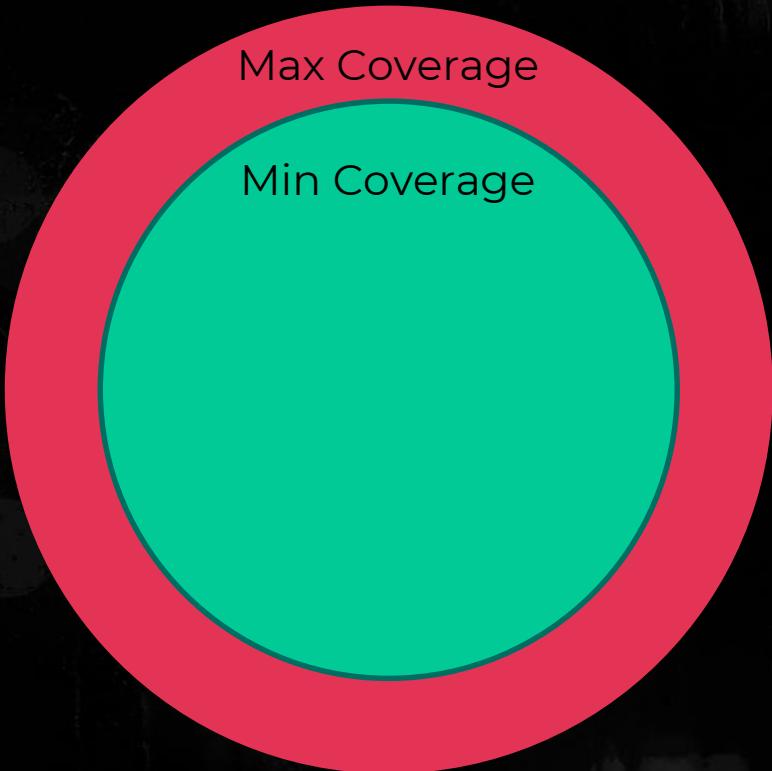
Kernel Coverage Issues

- Interrupts
- Background threads
- Nondeterminism
- Scheduler/allocator flakes



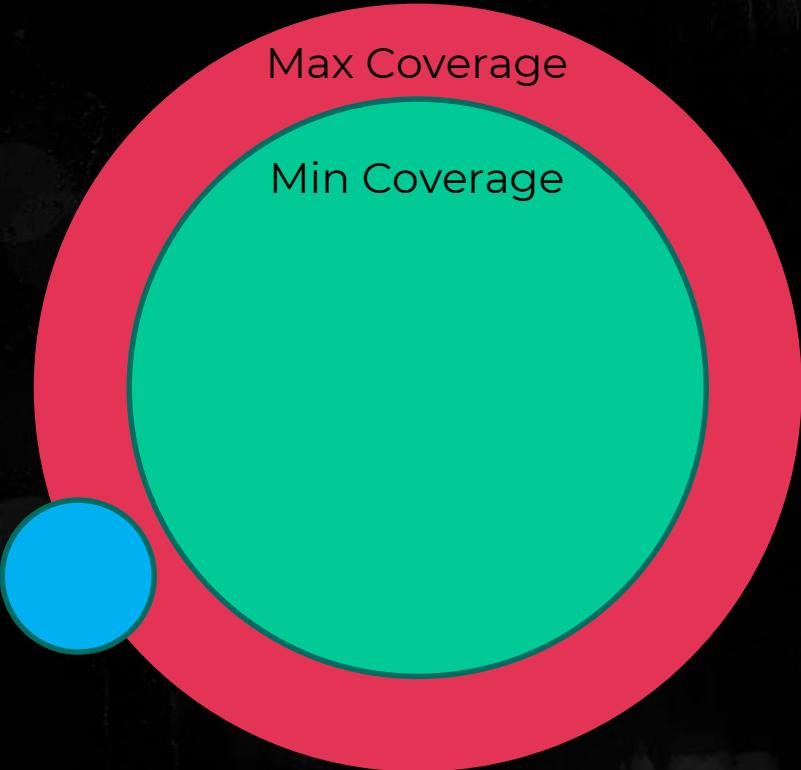
Kernel Coverage Issues

- Interrupts
- Background threads
- Nondeterminism
- Scheduler/allocator flakes



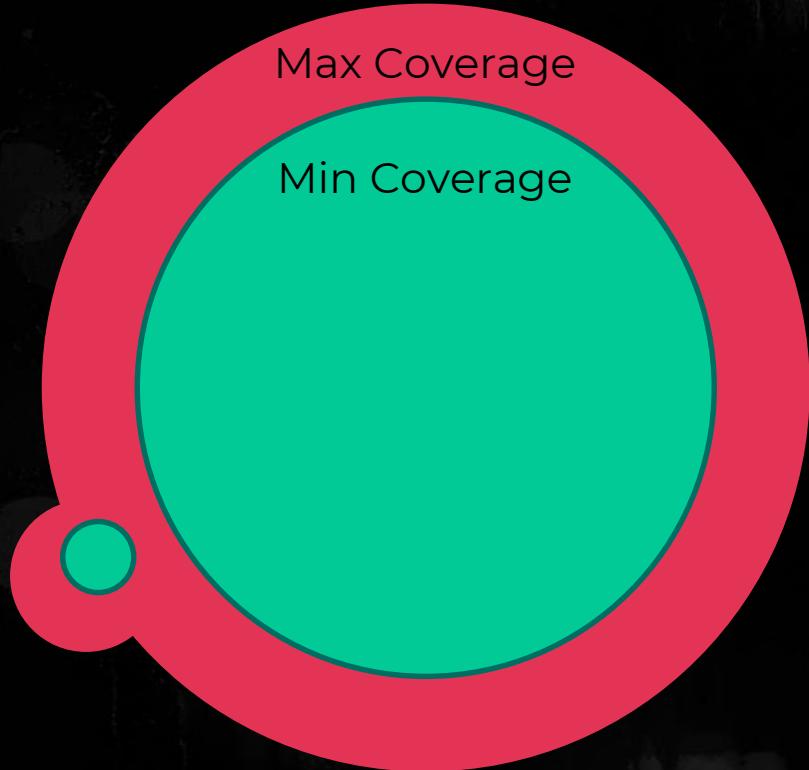
Kernel Coverage Issues

- Interrupts
- Background threads
- Nondeterminism
- Scheduler/allocator flakes



Kernel Coverage Issues

- Interrupts
- Background threads
- Nondeterminism
- Scheduler/allocator flakes



Test case generator/mutator

Feedback mechanism

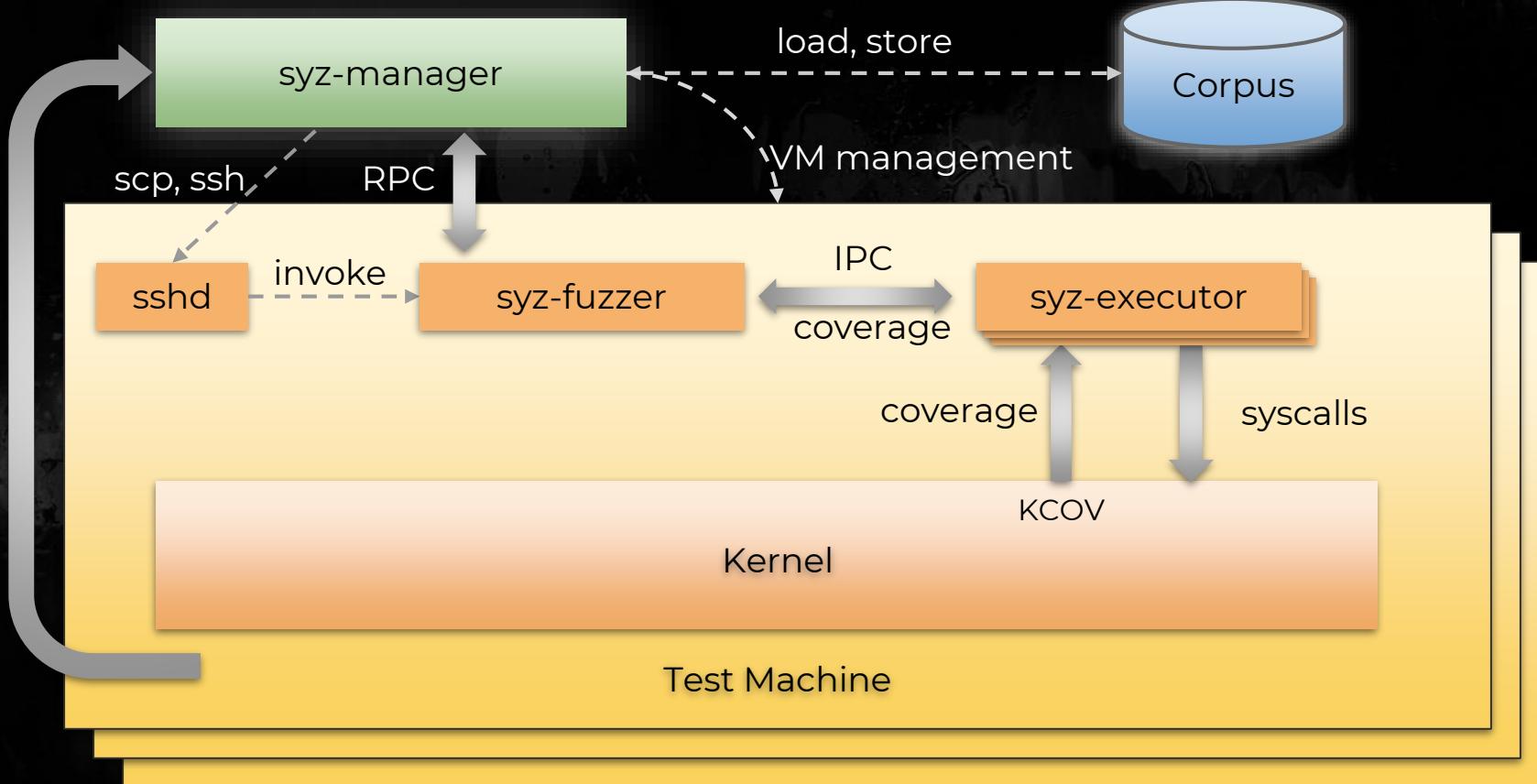
Bug oracle

Test case generator/mutator

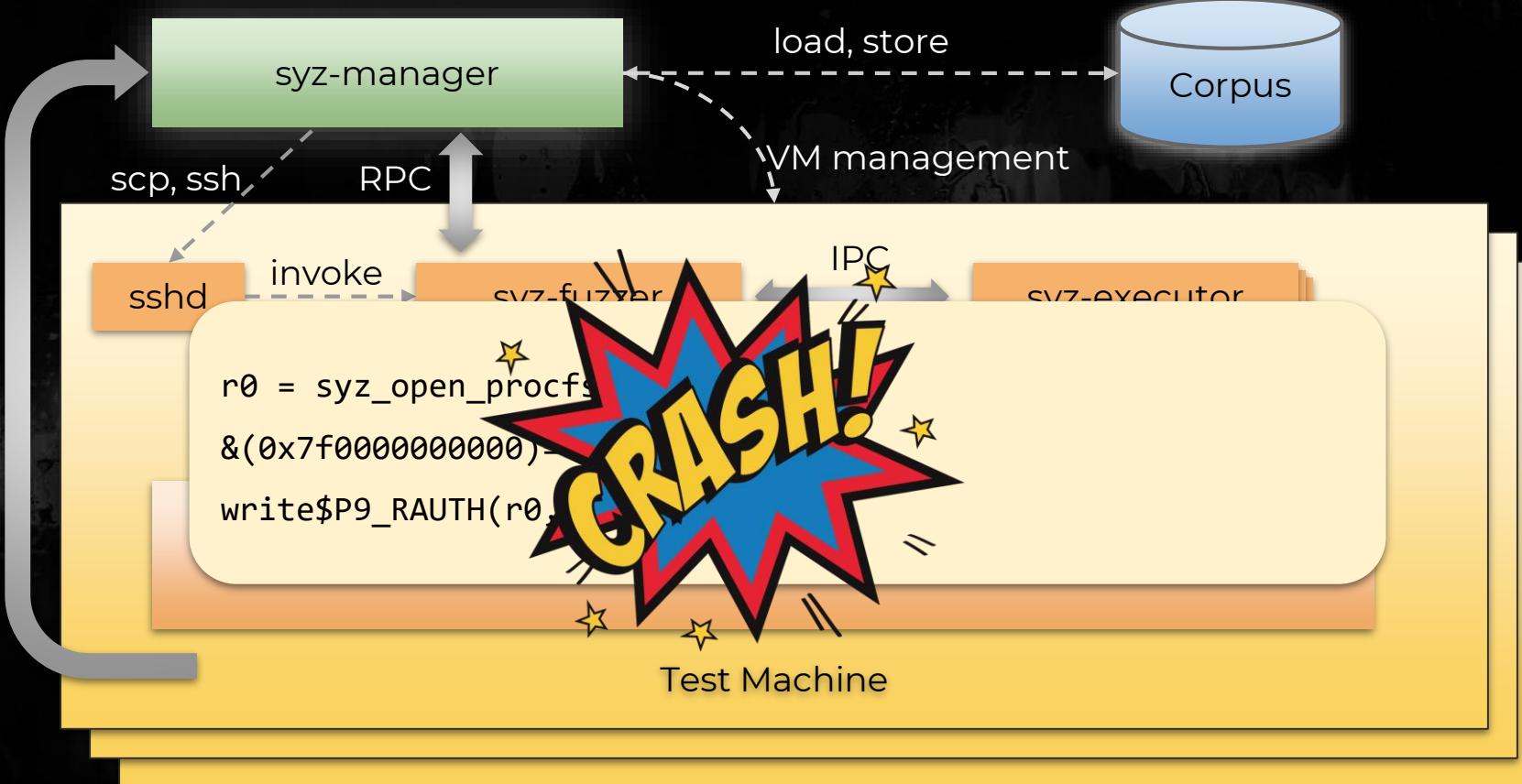
Feedback mechanism

Bug oracle

Syzkaller - Architecture (Linux)



Syzkaller - Architecture (Linux)



Test case generator/mutator

Feedback mechanism

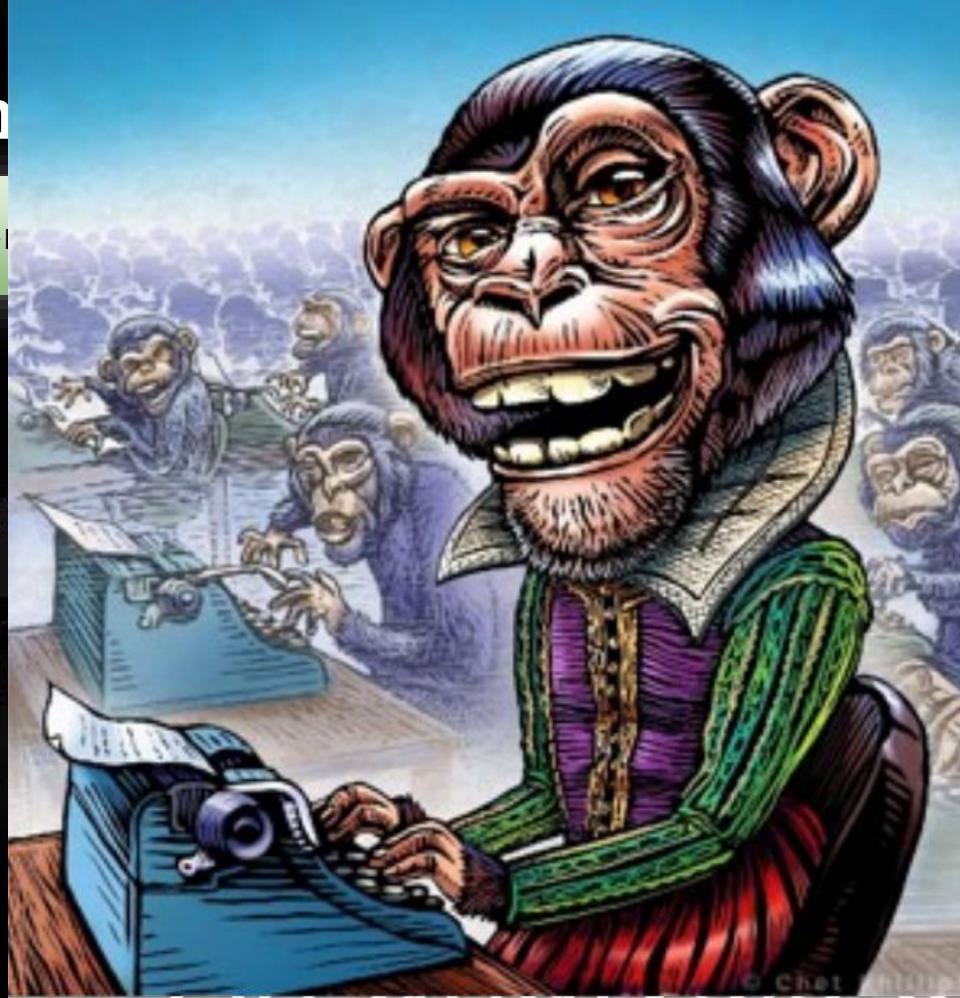
Bug oracle

Syzkaller - Unsupervised

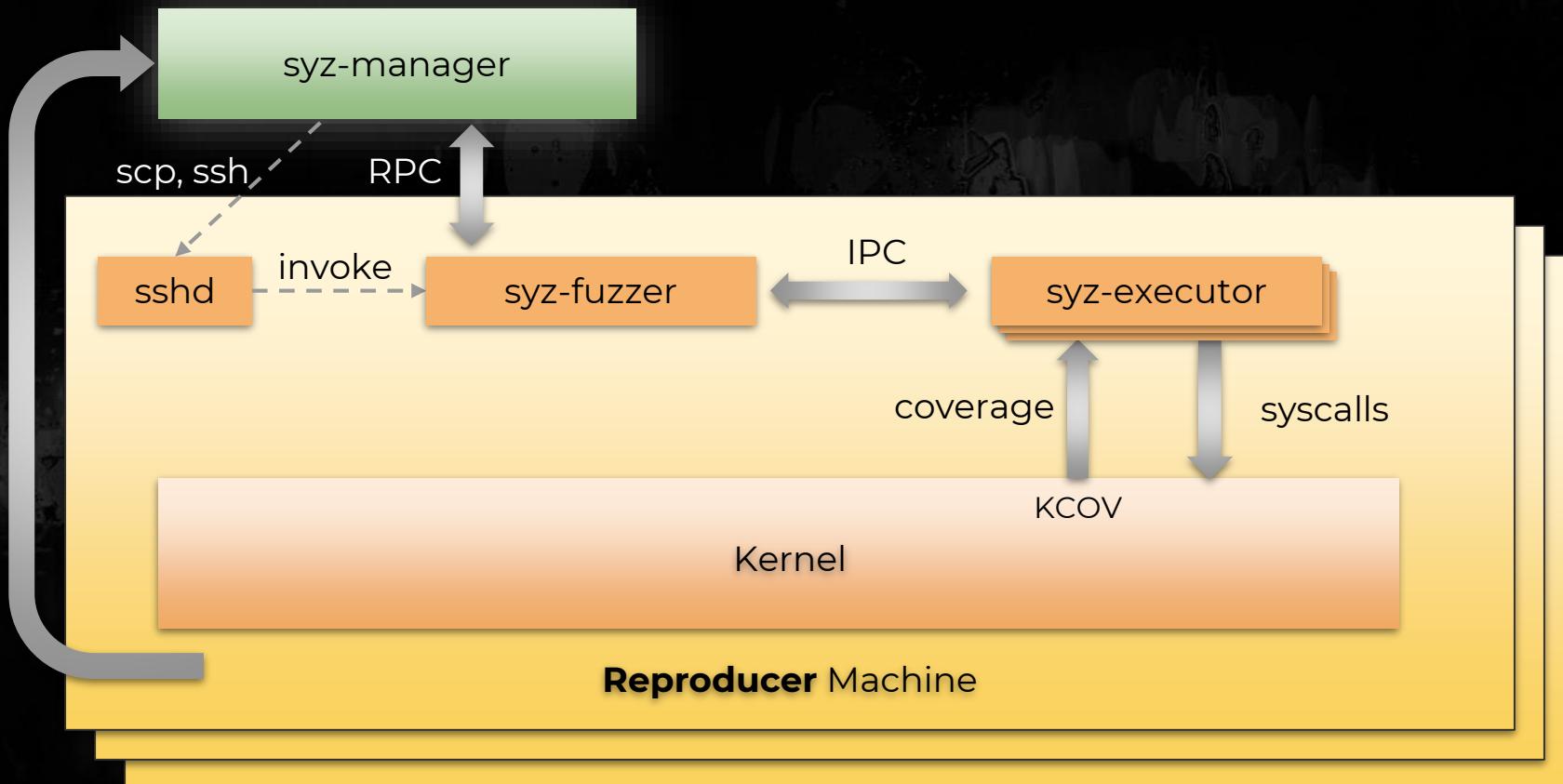
syz-manager

Syzkaller - Un

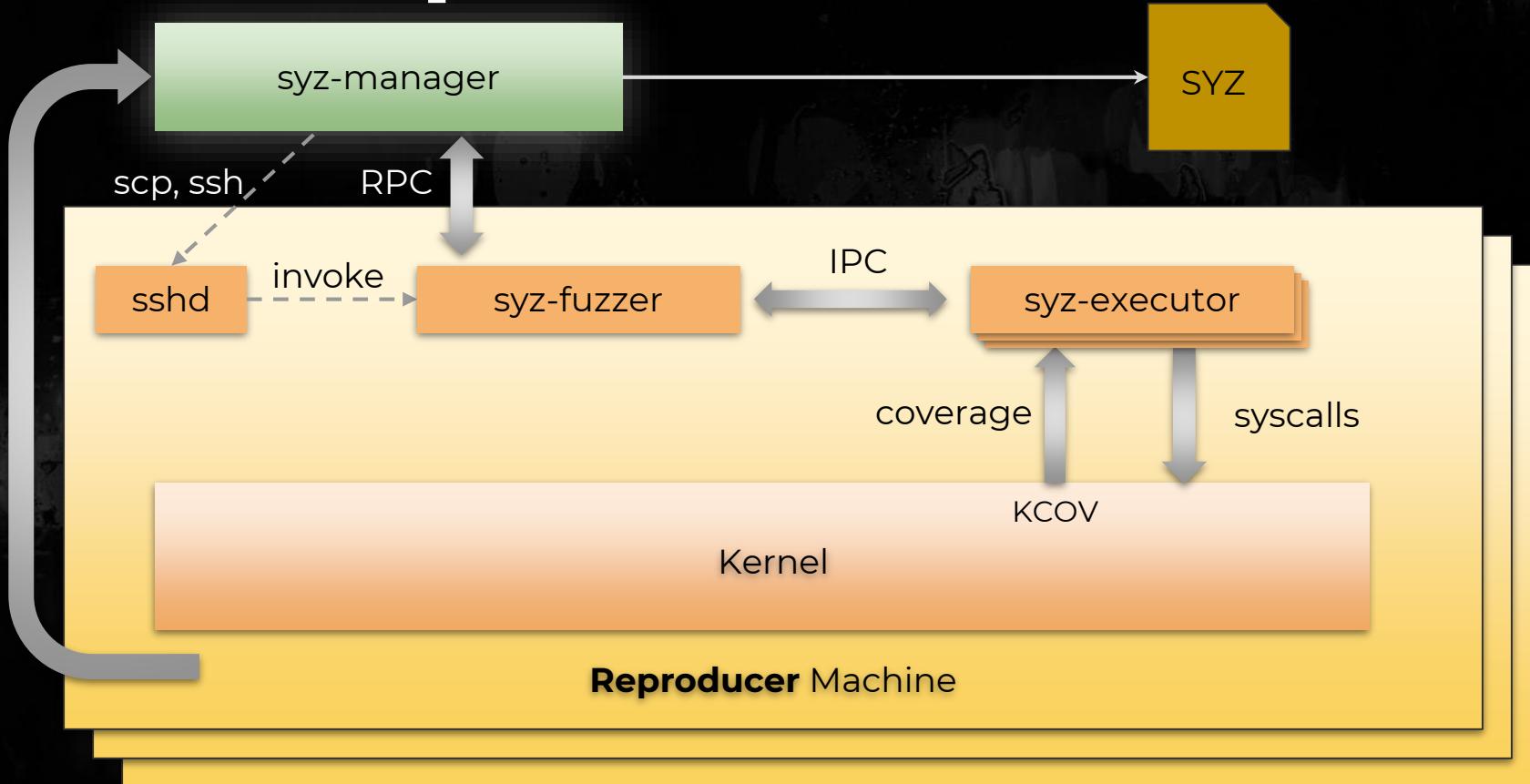
syz-



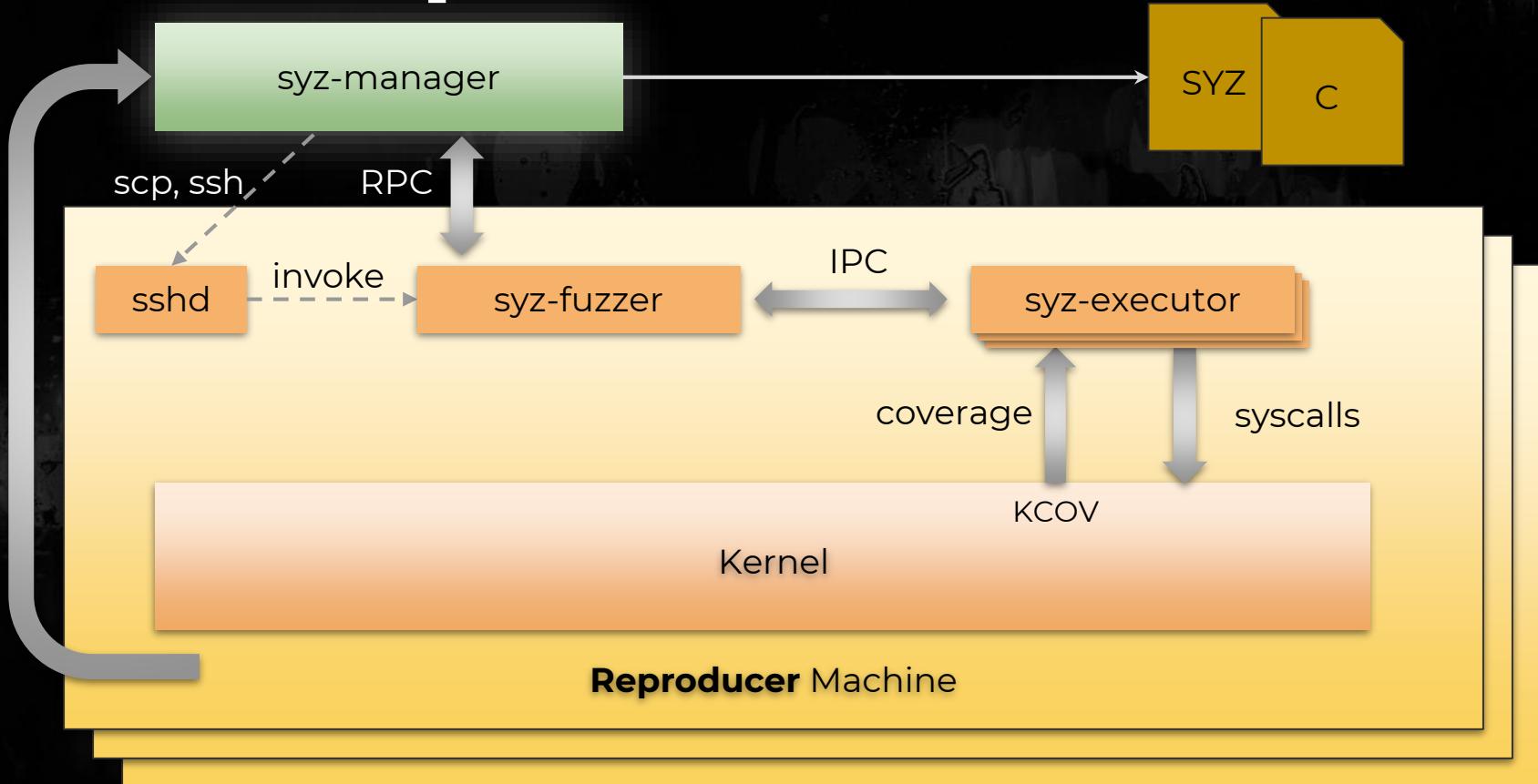
Syzkaller - Unsupervised



Syzkaller - Unsupervised



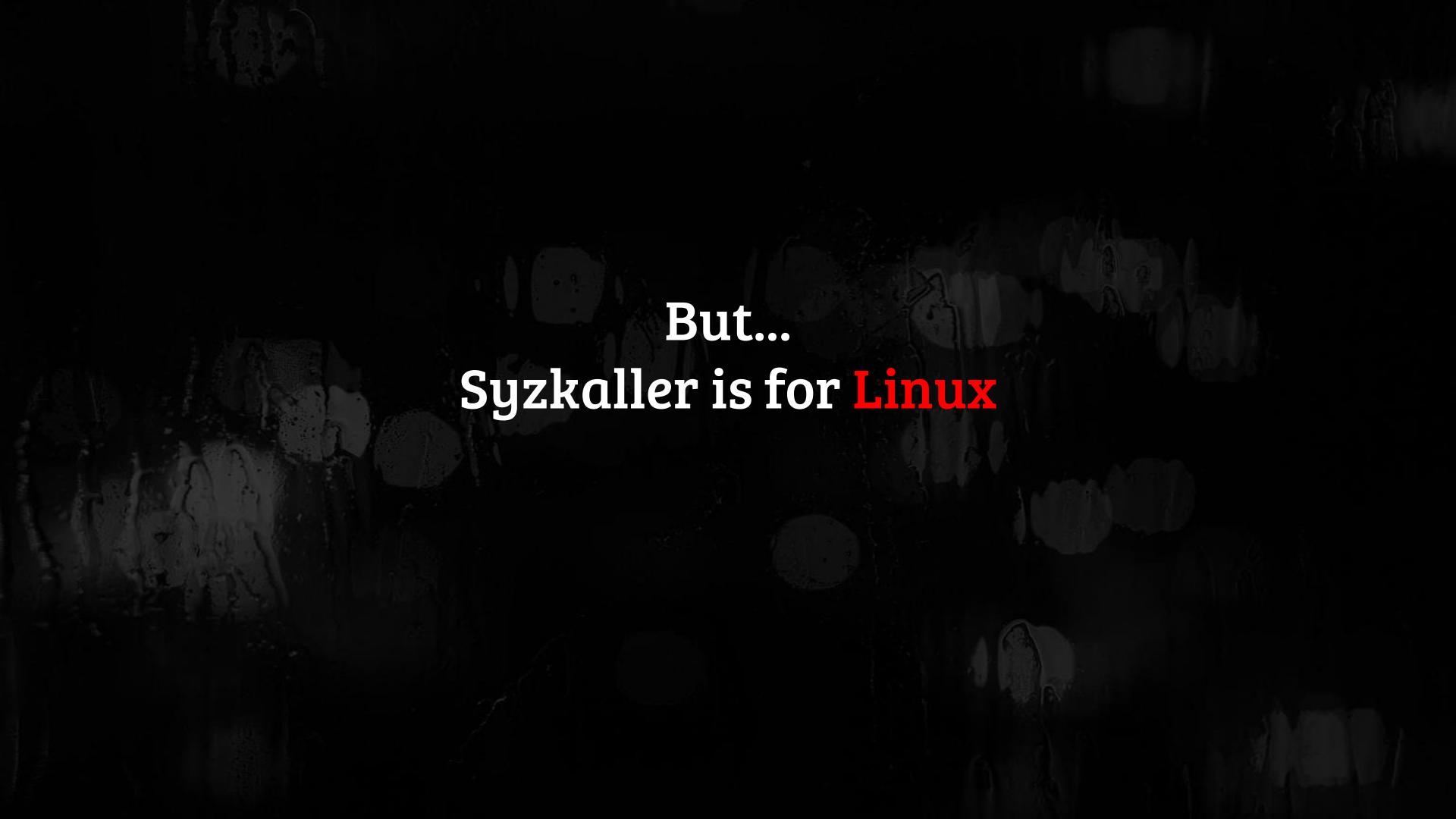
Syzkaller - Unsupervised





Syzkaller is awesome!

But...



But...
Syzkaller is for Linux

Dmitry Vyukov
(syzkaller's father)

One correction: syzkaller is not a Linux fuzzer for N years now.



google/syzkaller
github.com

It supports 6 full-fledged OSes, not counting Windows, Trusty and gVisor.

10:52 AM

But...

Syzkaller is for **Linux**

Dmitry Vyukov
(syzkaller's father)

One correction: syzkaller is not a Linux fuzzer for N years now.



google/syzkaller
github.com

It supports 6 full-fledged OSes, not counting Windows, Trusty and gVisor.

10:52 AM

But...

Syzkaller is for Linux

also

Run Linux on Windows

Install and run Linux distributions side-by-side on the Windows Subsystem for Linux (WSL).



WSL - Windows Subsystem for Linux

WSL is a compatibility layer for running Linux binaries natively on Windows

Allows to interop linux & windows binaries:

WSL - Windows Subsystem for Linux

WSL is a compatibility layer for running Linux binaries natively on Windows

Allows to interop linux & windows binaries:



```
bash -c "tasklist.exe | wc -l"
```

WSL - Windows Subsystem for Linux

WSL is a compatibility layer for running Linux binaries natively on Windows

Allows to interop linux & windows binaries:

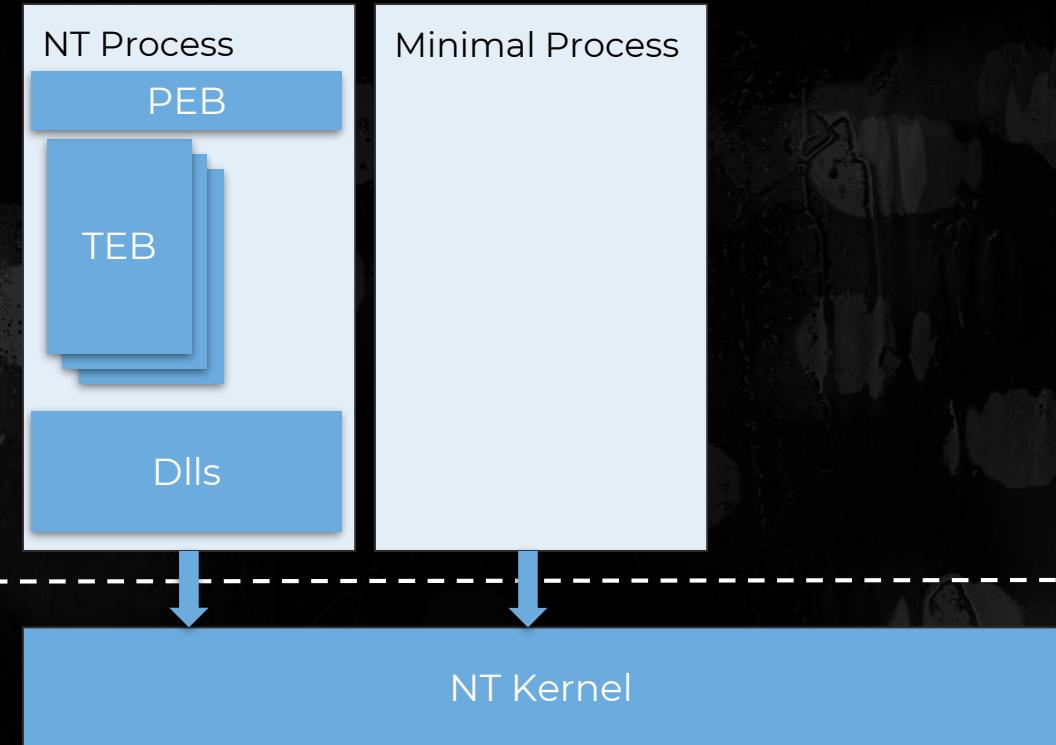


```
bash -c "tasklist.exe | wc -l"
```

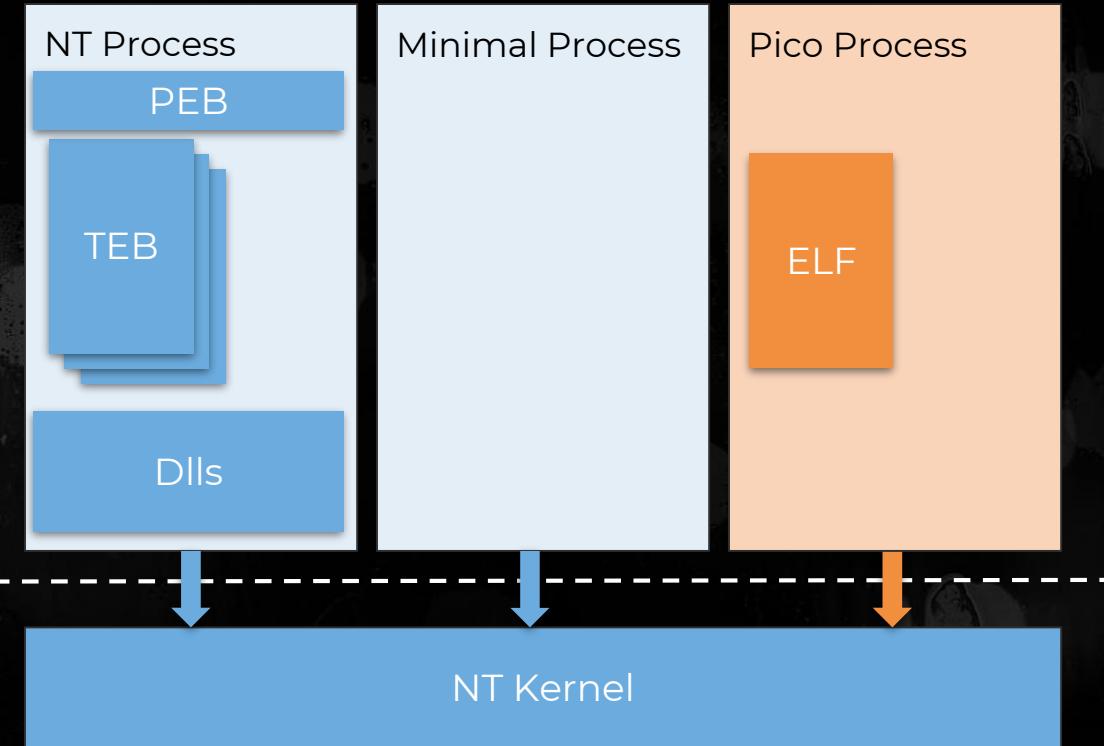
Require less resources than running a Virtual Machine

Aimed for running bash and core Linux commands

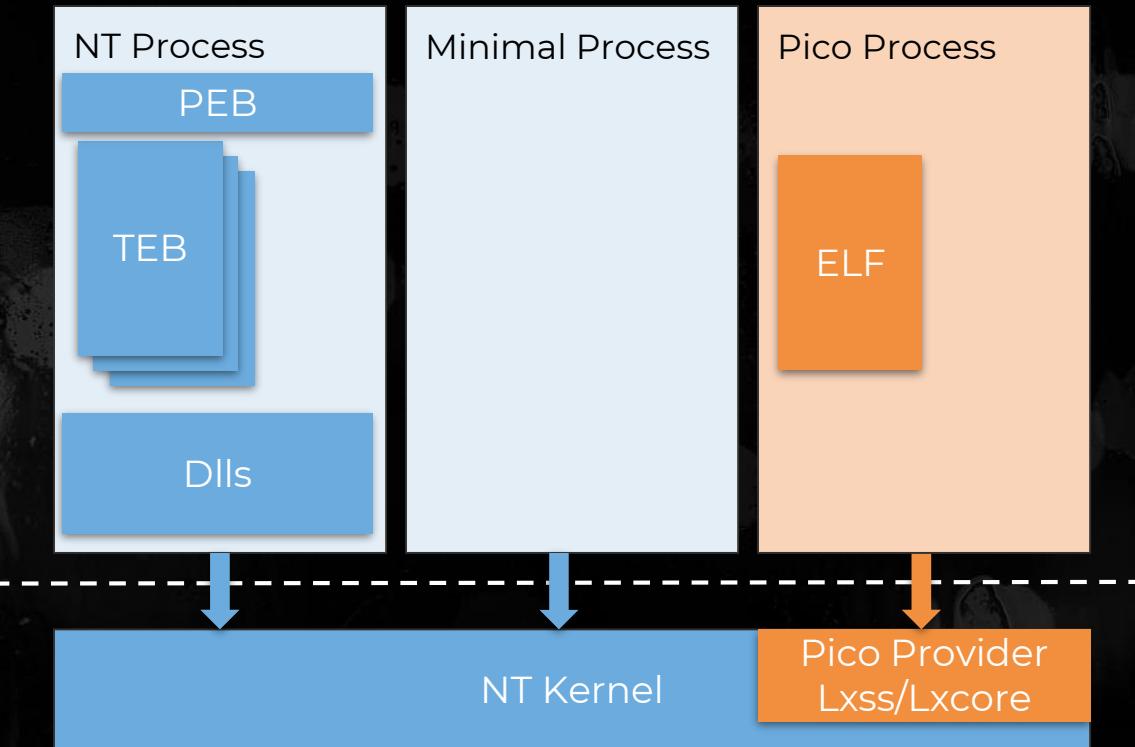
WSL - Bird's eye view



WSL - Bird's eye view



WSL - Bird's eye view



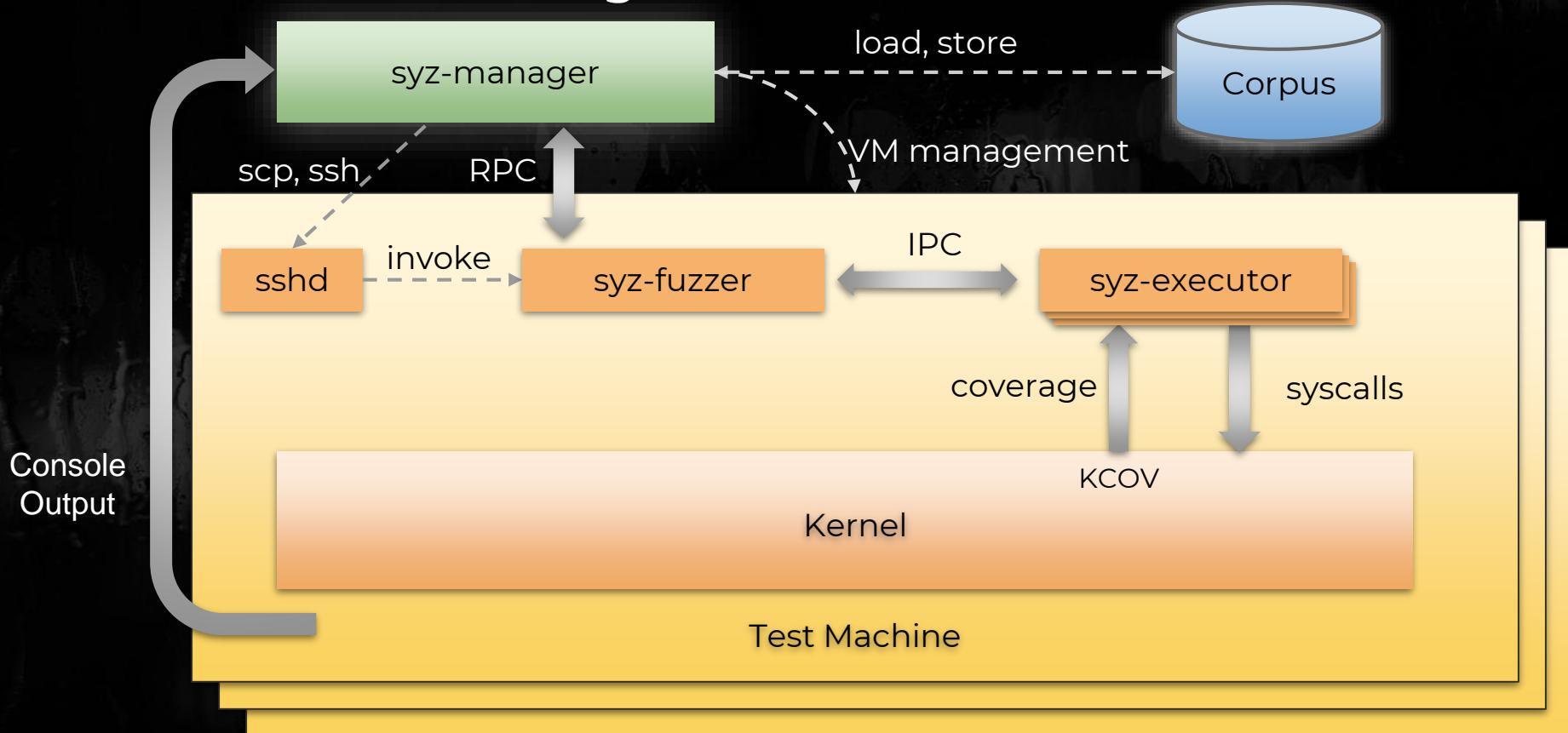
Why WSL?

Similar to fuzzing Linux

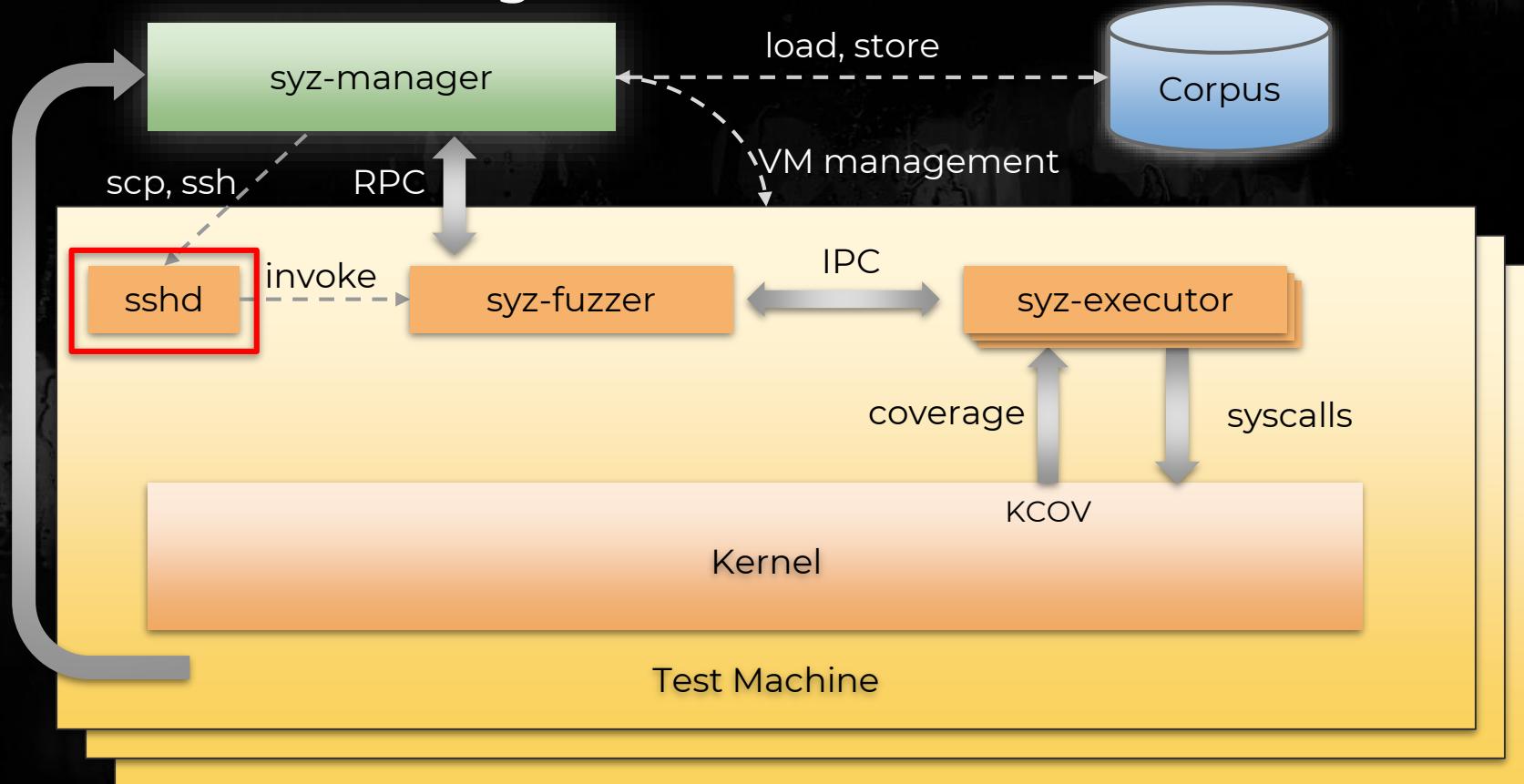
Relatively new (2 drivers, ~1MB in size)

First steps with `syzkaller`

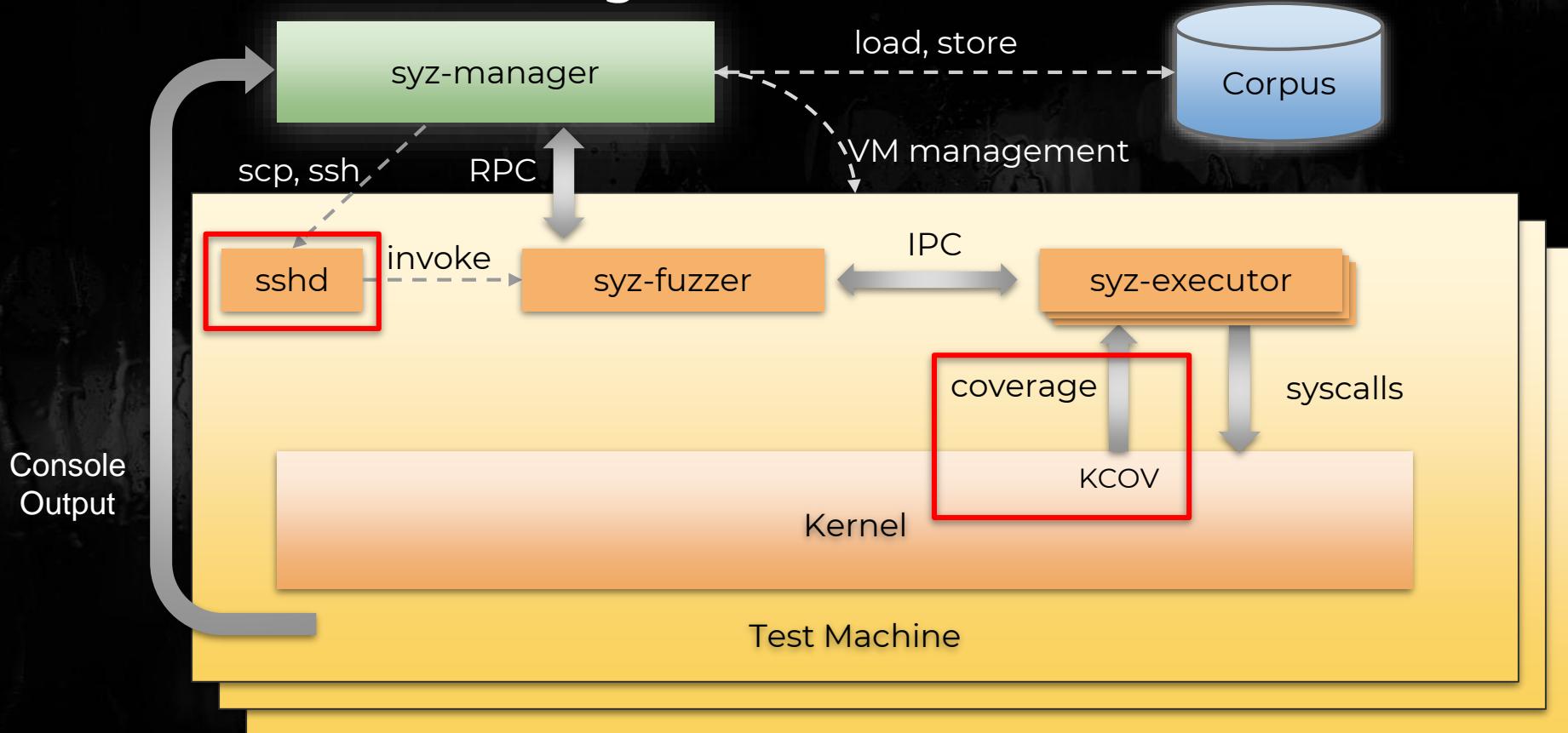
What do we need to get started



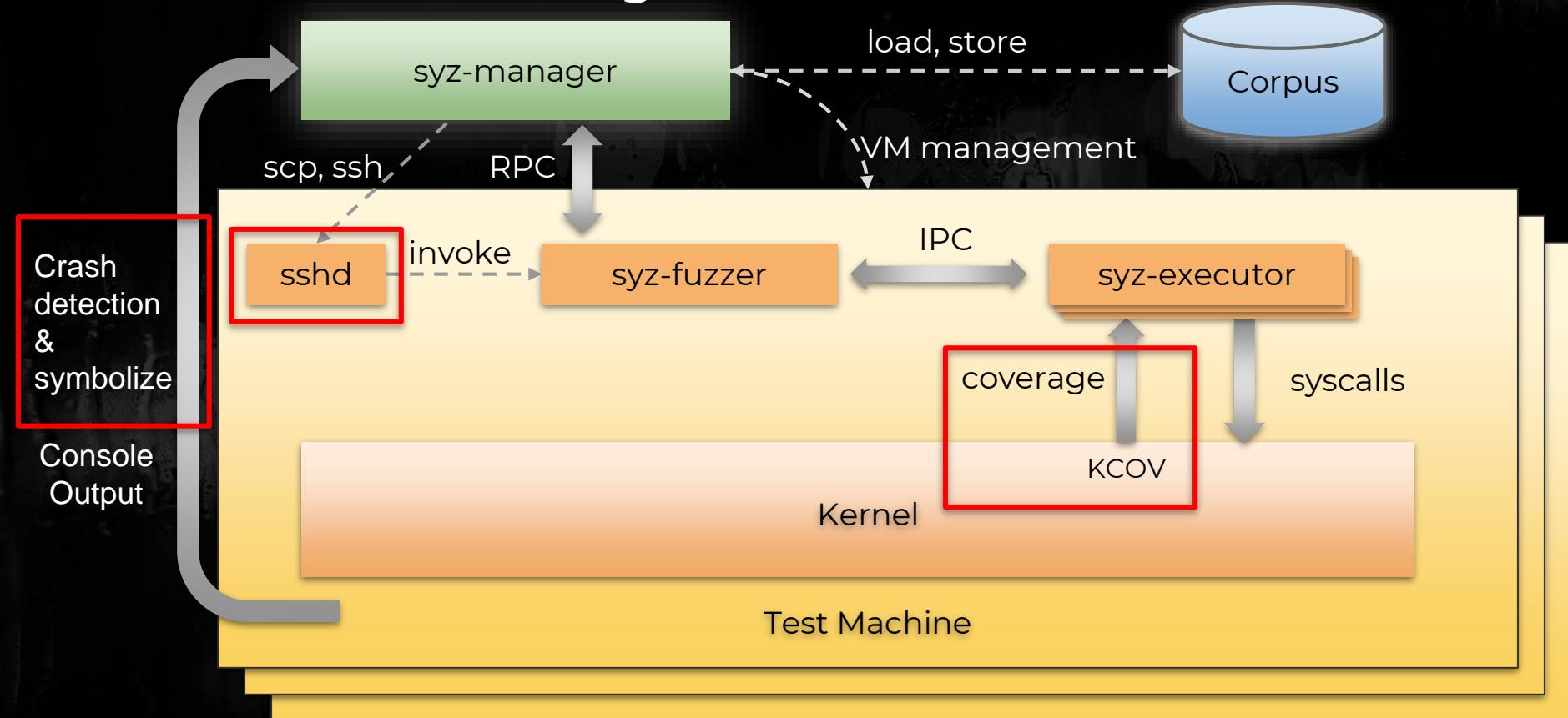
What do we need to get started



What do we need to get started



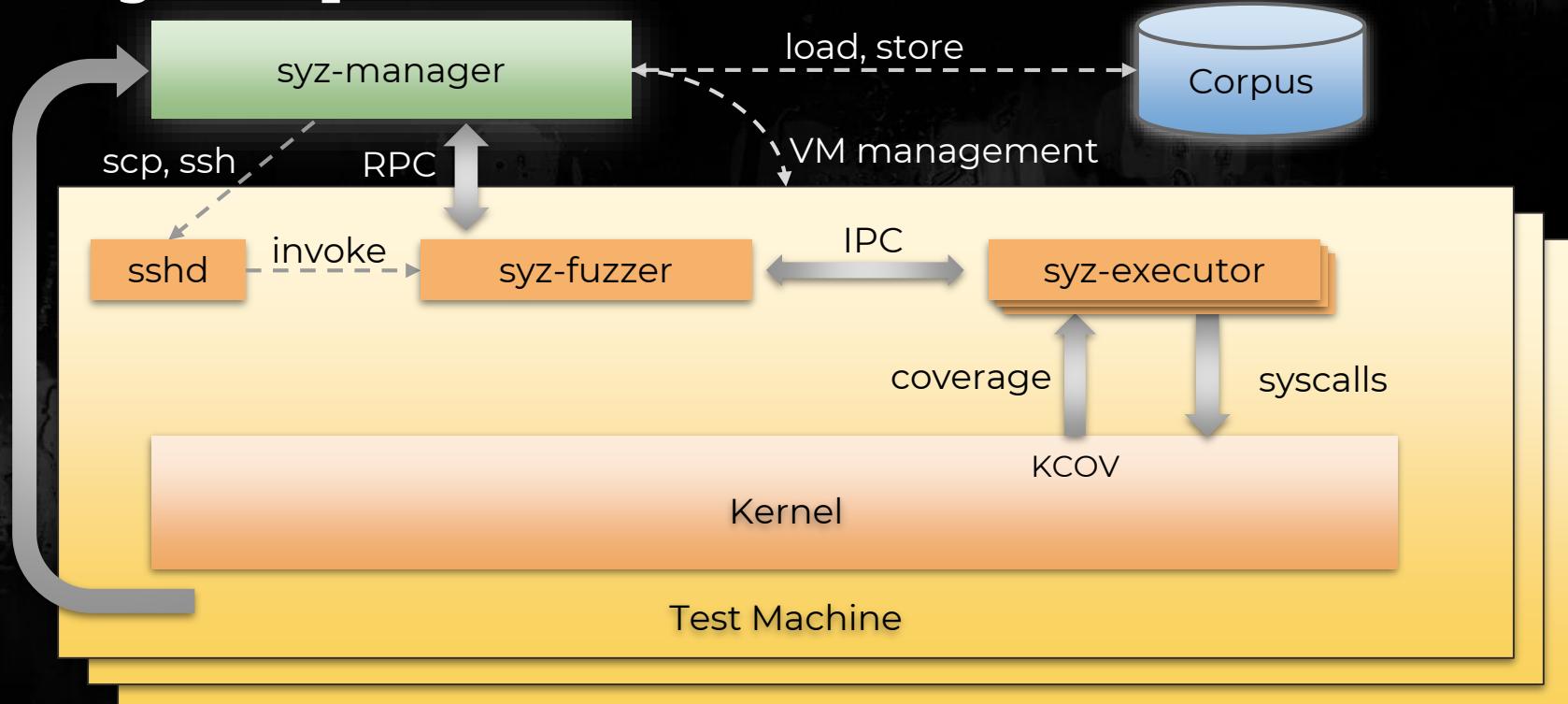
What do we need to get started



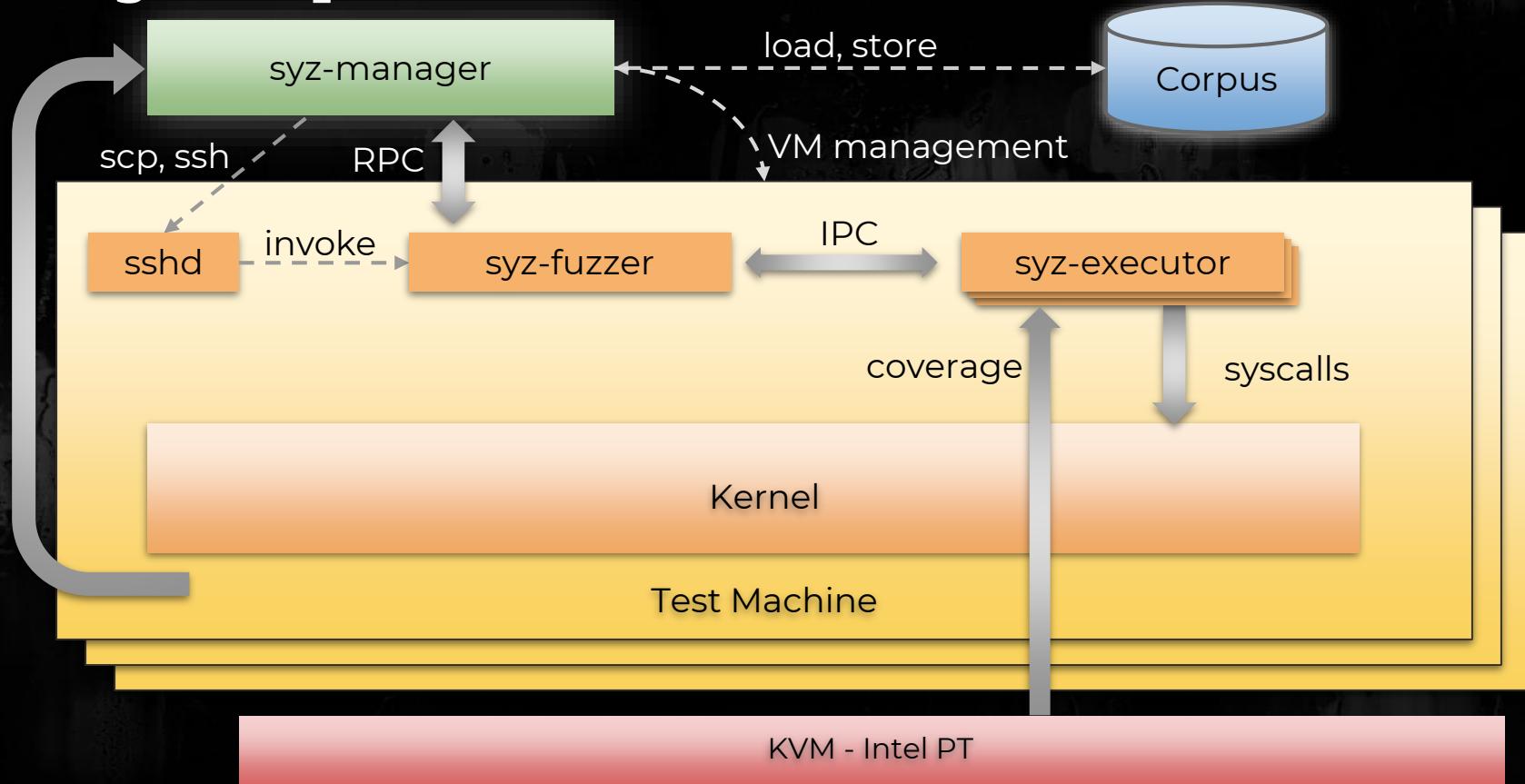
Syzkaller - WSL coverage

- Windows is a closed source compiled binary
 - We can't use -fsanitize=trace-pc (like KCOV)
- Options
 - Emulation - bochs
 - Static Binary Instrumentation
 - Hypervisor - like Apple Pie
 - **Hardware support - Intel PT**

Coverage - Implementation



Coverage - Implementation

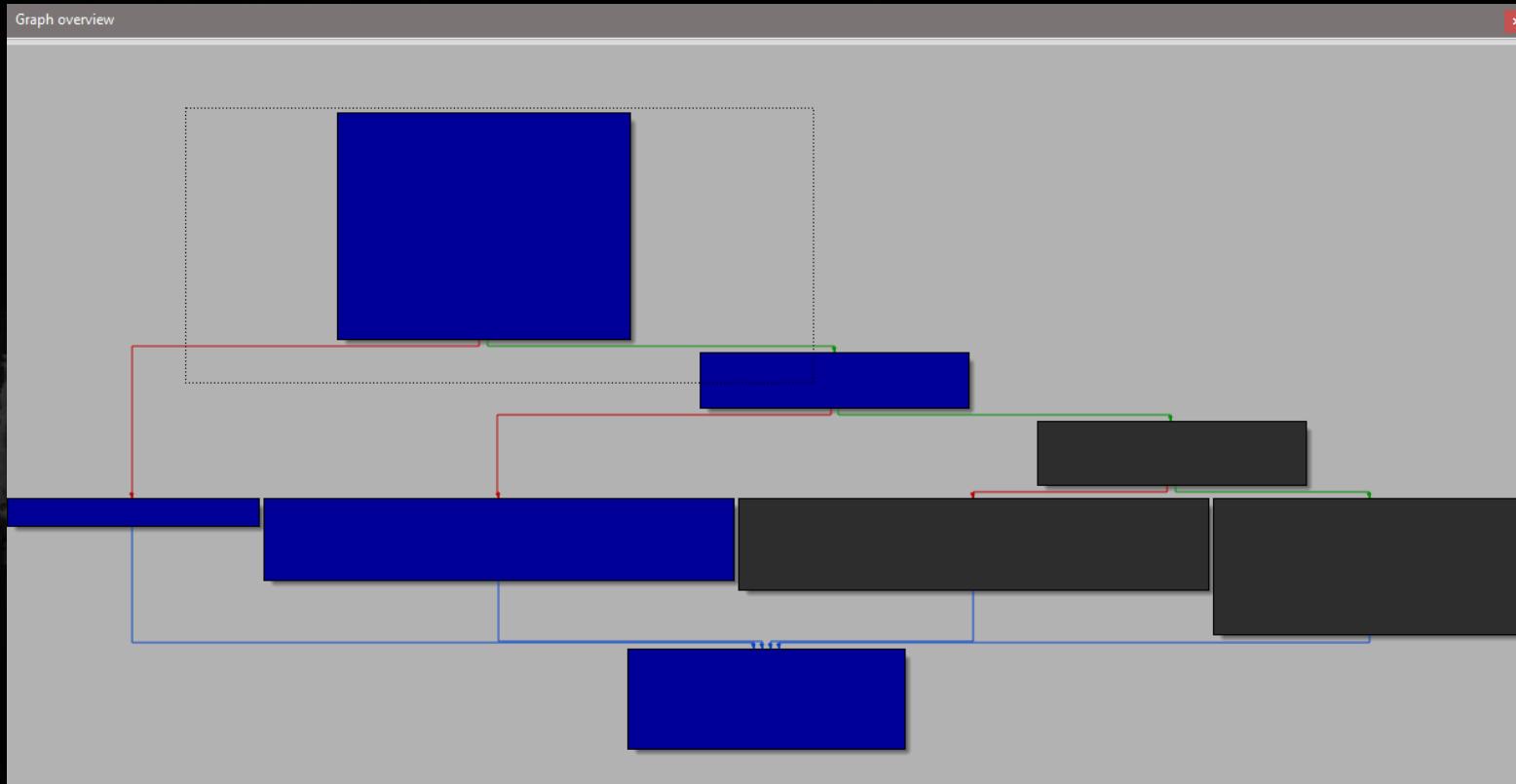


Syzkaller - WSL coverage

59.08	LxpEpollContextAddFileDescriptor	0x1C004DD5C	37 / 79	218 / 369	1353	45	
70.37	LxpMiFreeOverlappingRange	0x1C0086200	43 / 68	209 / 297	1137	46	
46.53	LxpDevTerminalDefaultDeviceControl	0x1C00FD6BC	37 / 89	208 / 447	1889	48	
65.13	LxpMmMremap	0x1C0088AEC	54 / 100	325 / 499	1889	49	
48.03	LxpEpollContextWait	0x1C004E768	53 / 108	256 / 533	2203	55	
0.00	LxpDevTtyInputWorker	0x1C0101070	0 / 80	0 / 359	1481	55	
0.00	LxpProcFsRootCpuInfoGenerateStringCommon	0x1C00AEADC	0 / 65	0 / 520	2245	56	
0.00	LxpProcFsPidMapsGenerateBufferCommon	0x1C00AA51C	0 / 97	0 / 544	2293	56	
30.02	LxpSemPerformOperations	0x1C00F33F0	19 / 114	142 / 473	1828	58	
81.39	VfsRename	0x1C00929F8	67 / 95	376 / 462	1807	59	
89.53	LxUtilNtStatusToLxError	0x1C00FA894	84 / 94	171 / 191	893	60	
0.00	LxpElfInfoParse	0x1C004CABC	0 / 127	0 / 541	2552	67	
0.00	LxpDevLogTraceLoggingWrite	0x1C00B5024	0 / 105	0 / 431	1626	67	
68.94	LxpMmAllocateMapVm	0x1C0086680	84 / 150	424 / 615	2407	82	
0.00	LxpProcFsPidStatusGenerateBuffer	0x1C00ABC50	0 / 110	0 / 664	2707	82	
80.59	LxpDevTerminalProcessInput	0x1C00FE72C	110 / 149	411 / 510	1859	89	
43.98	_output_1	0x1C0009560	67 / 182	252 / 573	2121	100	



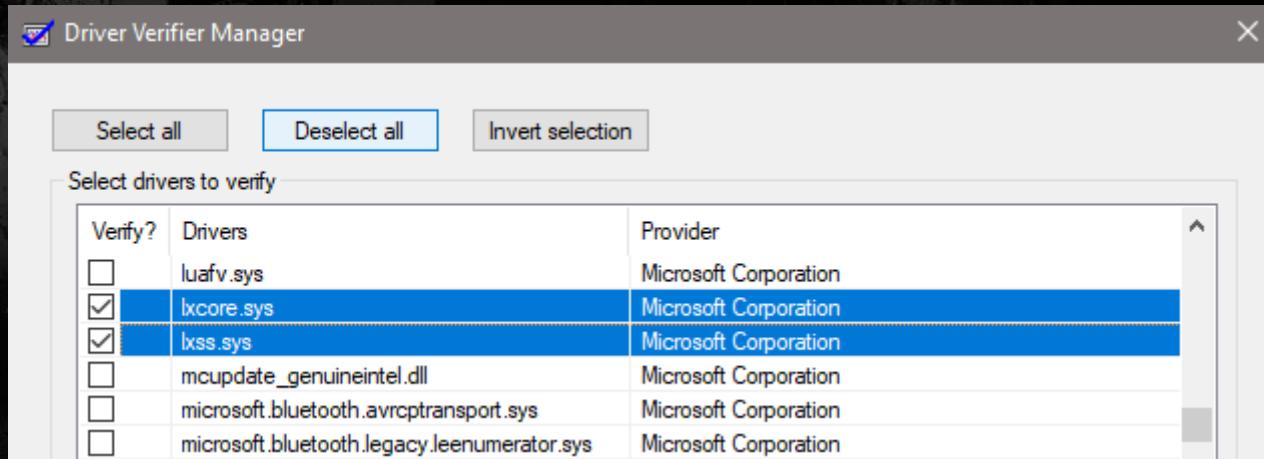
Syzkaller - WSL coverage



Bug Oracle

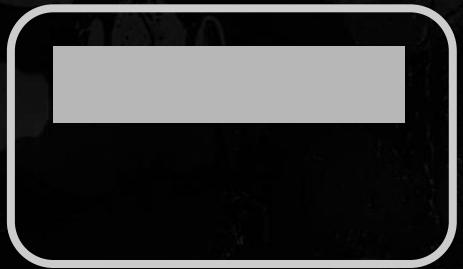
We used the same technique as kAFL

Enabled driver verifier for lxcore & lxss



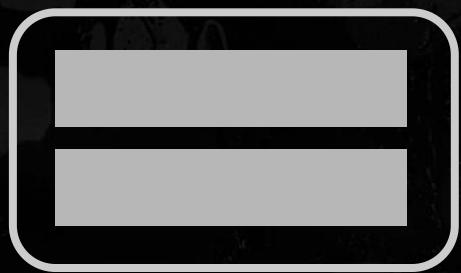
Symbolizing

crashes



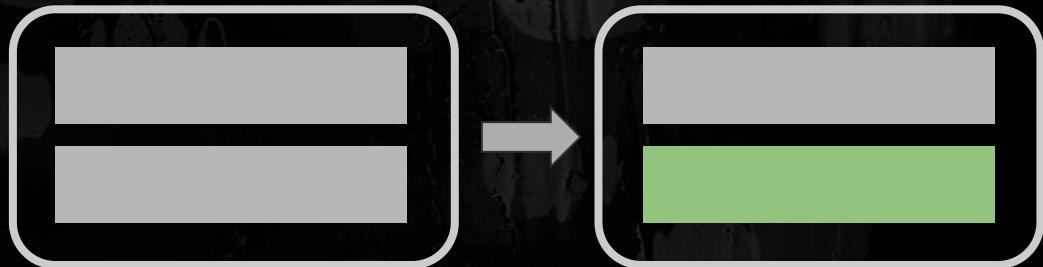
Symbolizing

crashes

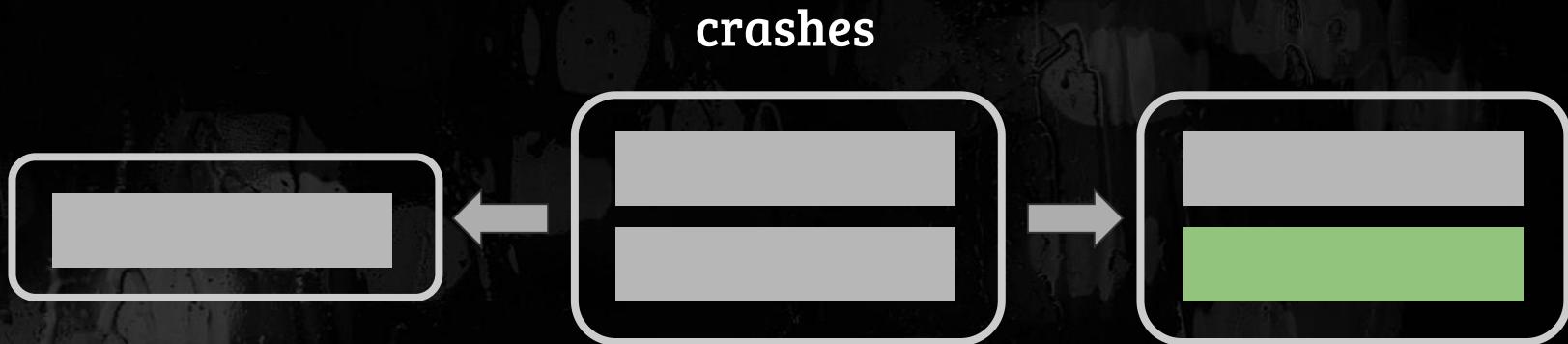


Symbolizing

crashes

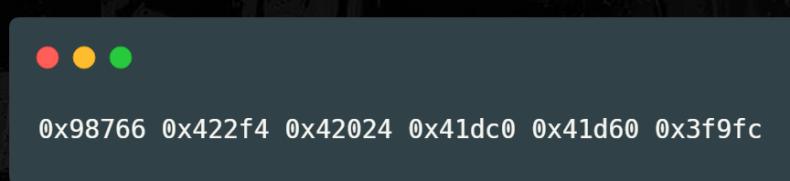


Symbolizing

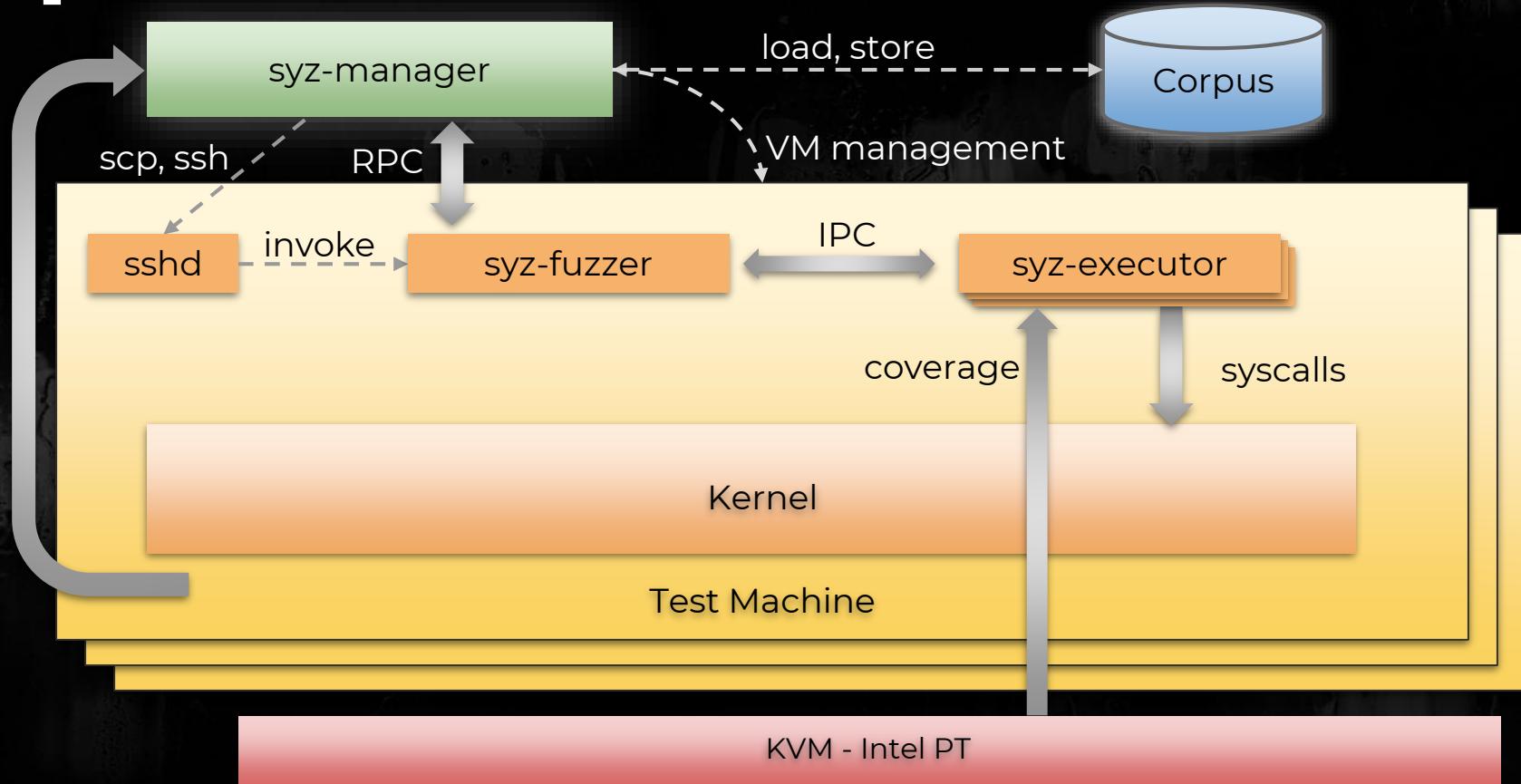


Crash De-duplication & symbolize

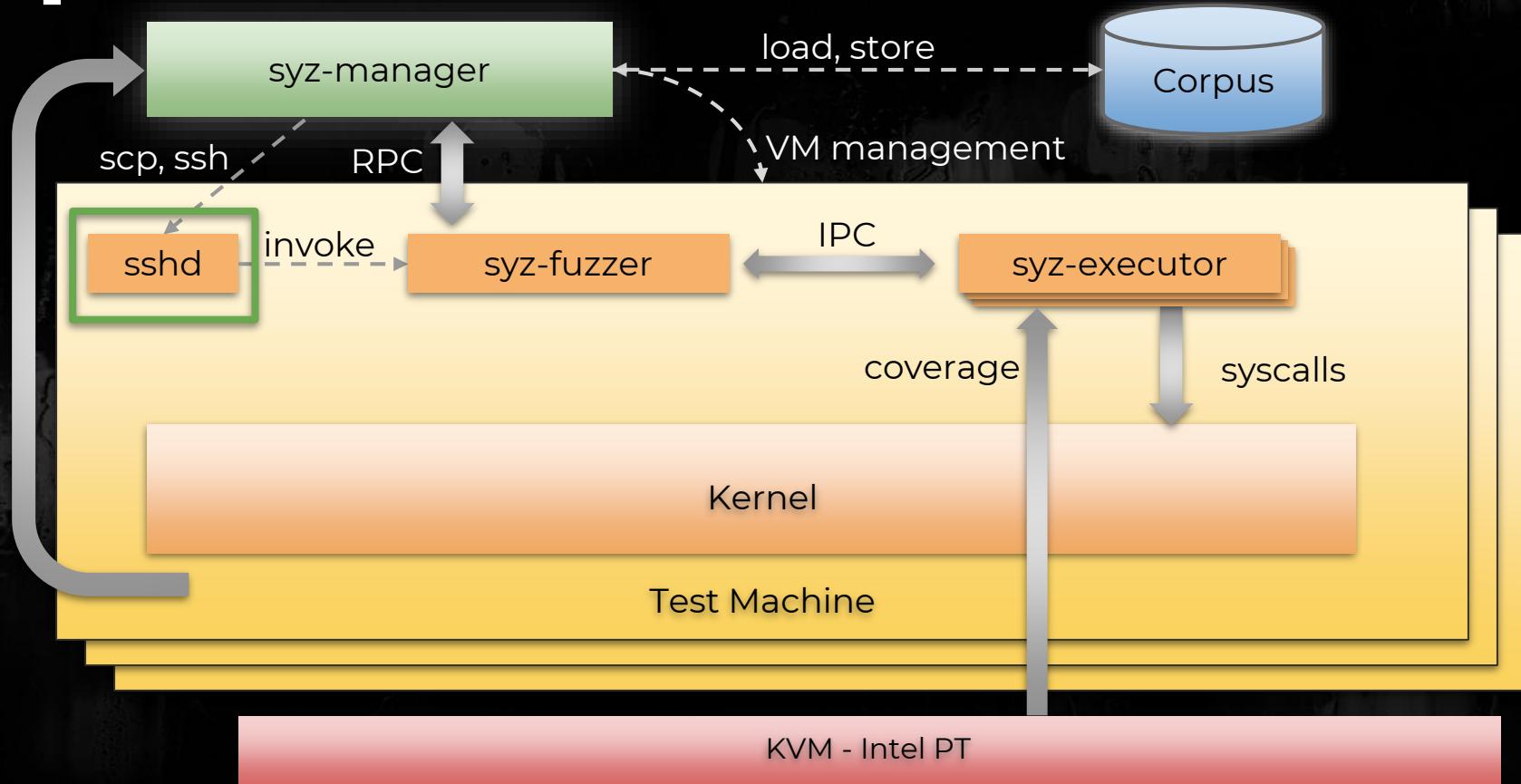
- Scanning the stack
- Search for addresses within the module
- Output the first few as offsets (ASLR)



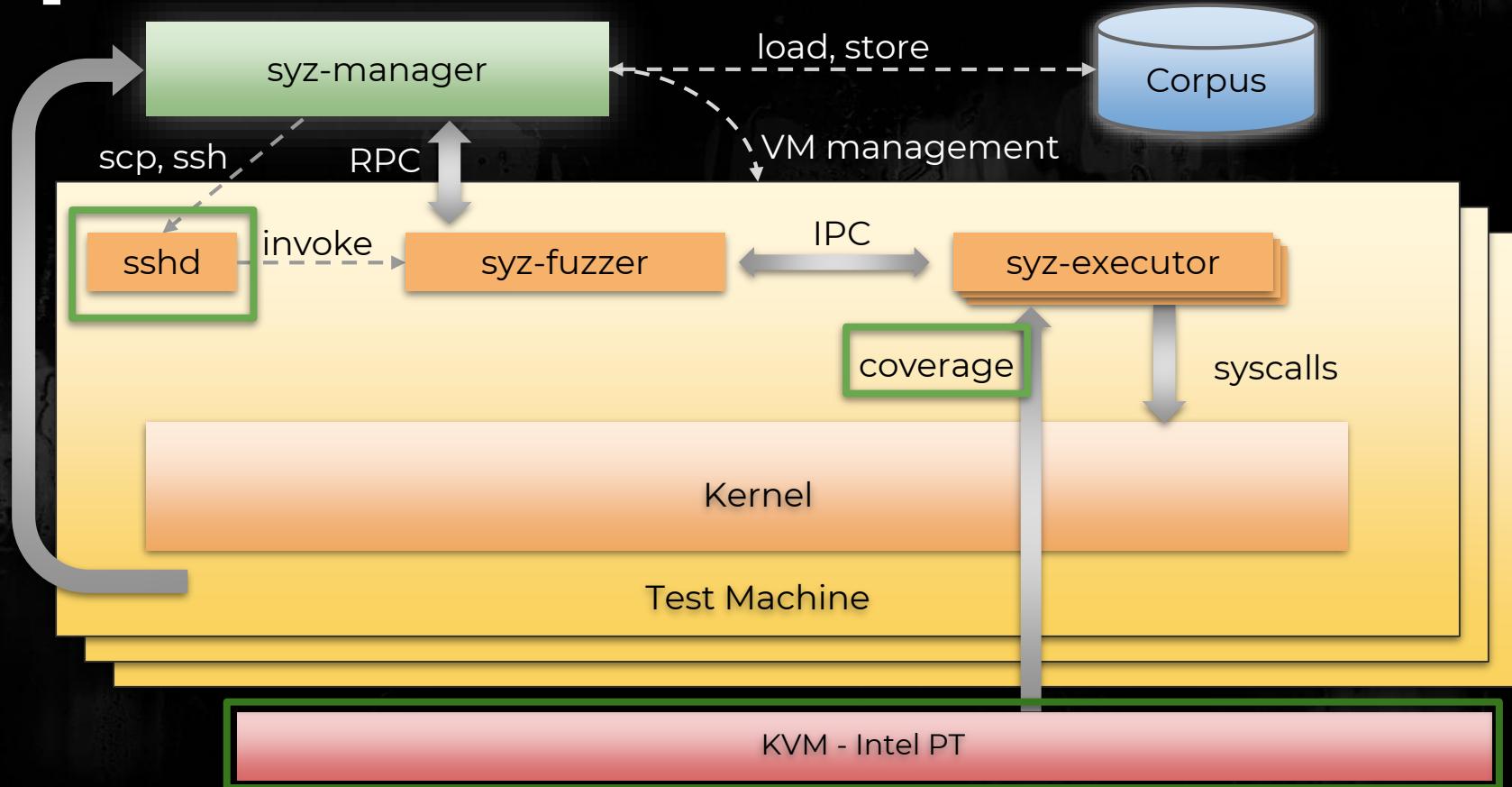
Recap



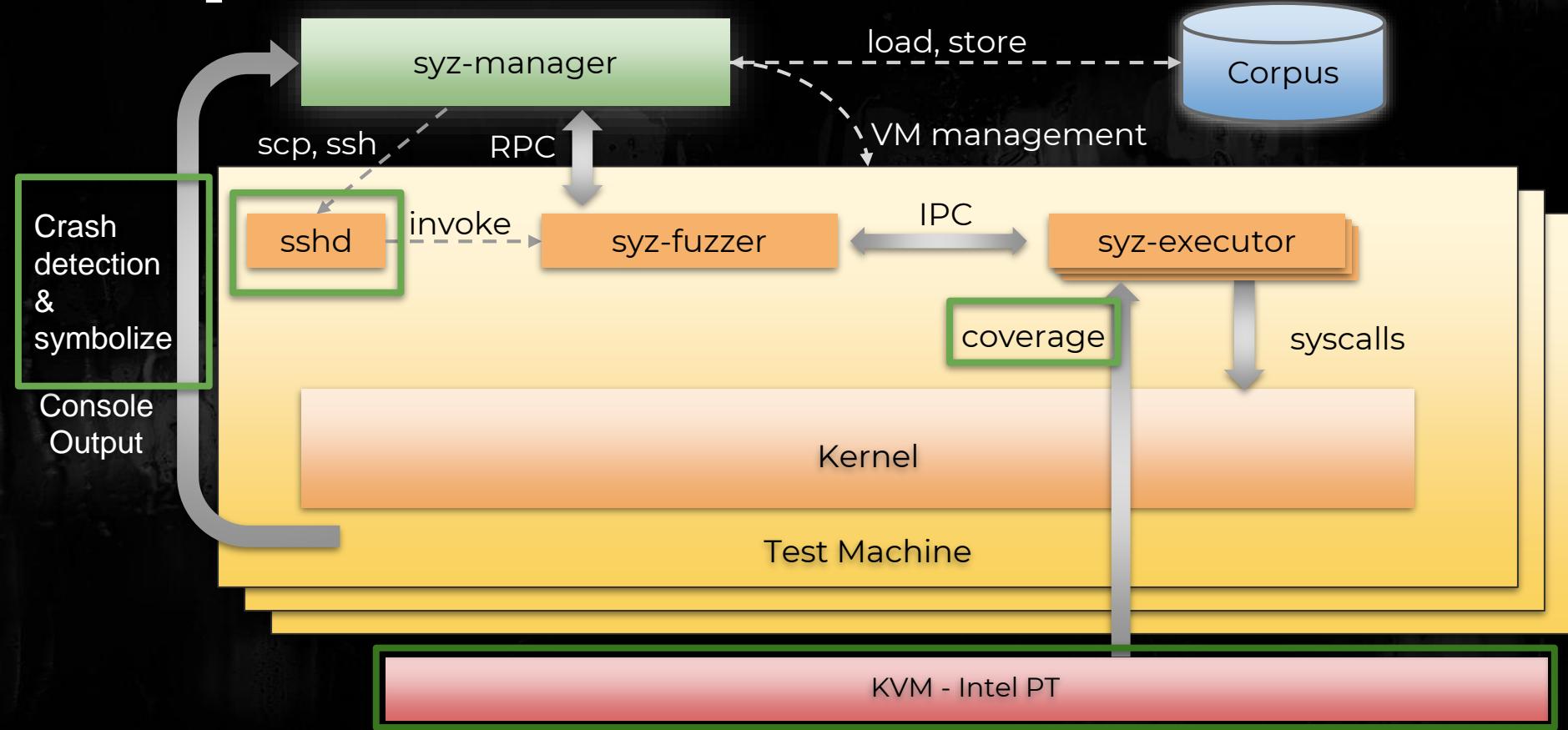
Recap



Recap



Recap



Sanity check

We used a hypercall to patch kernel to emulate a “bug”

We let the fuzzer run until the path with bug was reached

Syzkaller was able to find it

Let's run the fuzzer!

And then it rained bugs!





And then it rained bugs!

Not really

Not really

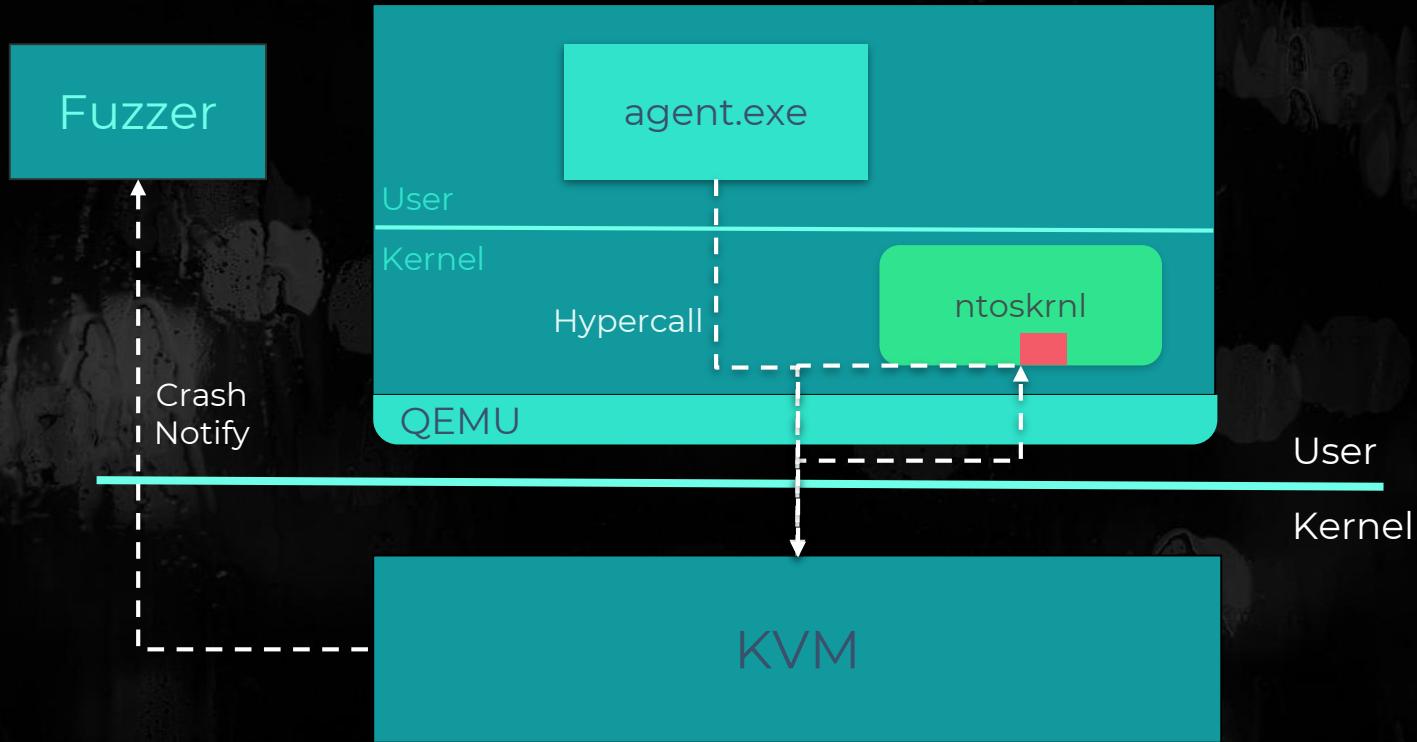
We noticed a crash with CRITICAL_STRUCTURE_CORRUPTION



Patch Guard

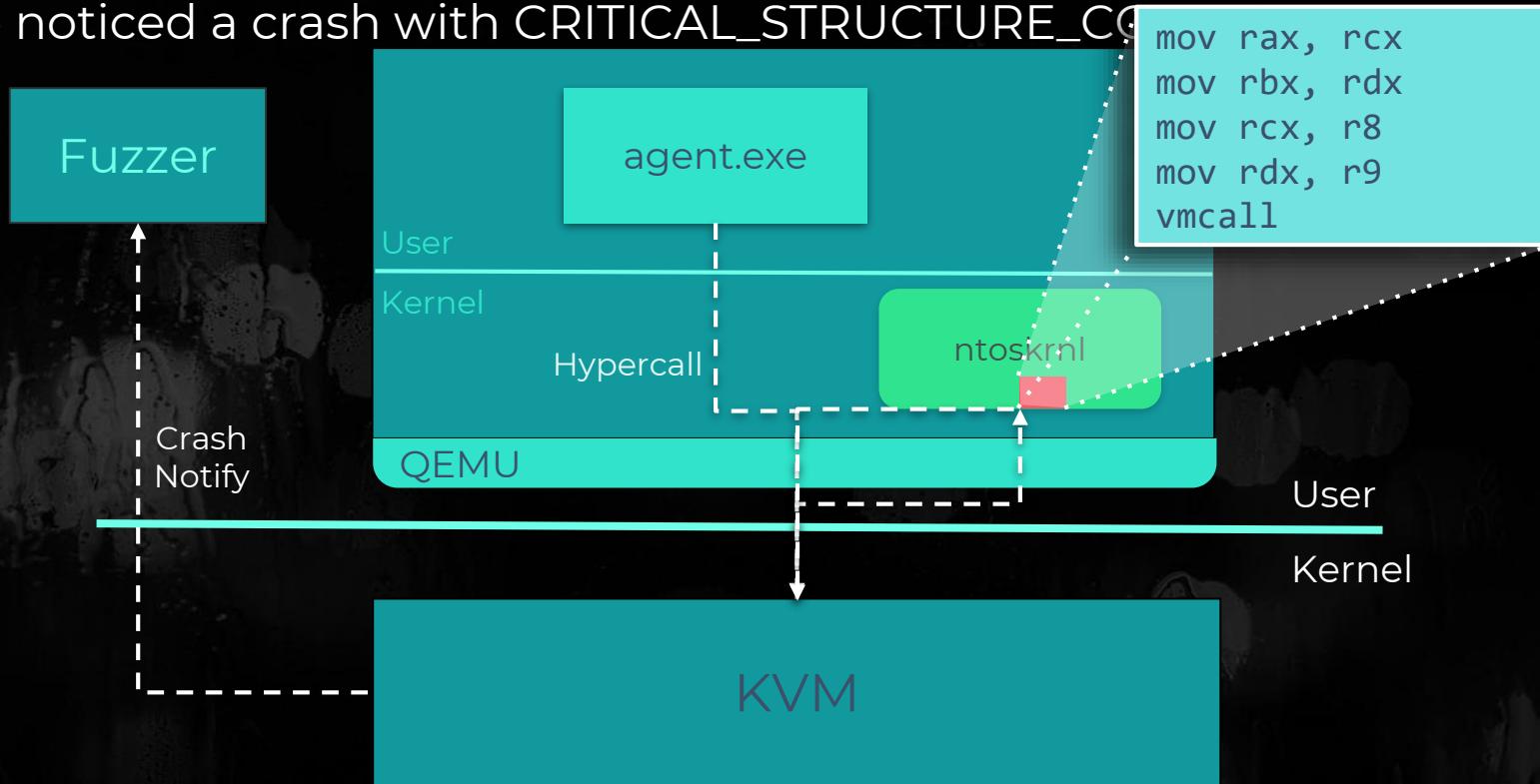
Not really

We noticed a crash with CRITICAL_STRUCTURE_CORRUPTION



Not really

We noticed a crash with CRITICAL_STRUCTURE_CO



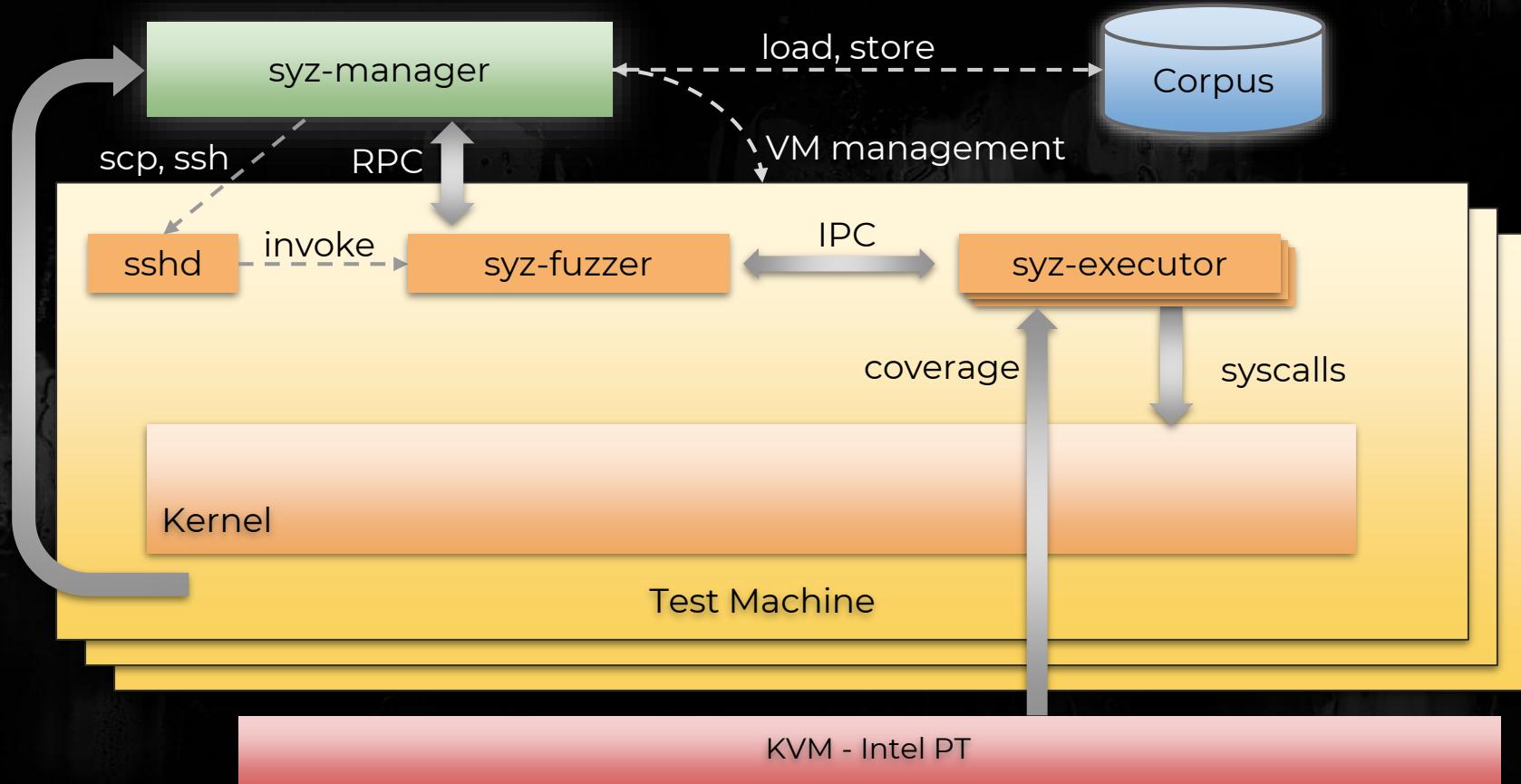
Not really

We noticed a crash with CRITICAL_STRUCTURE_CODE:

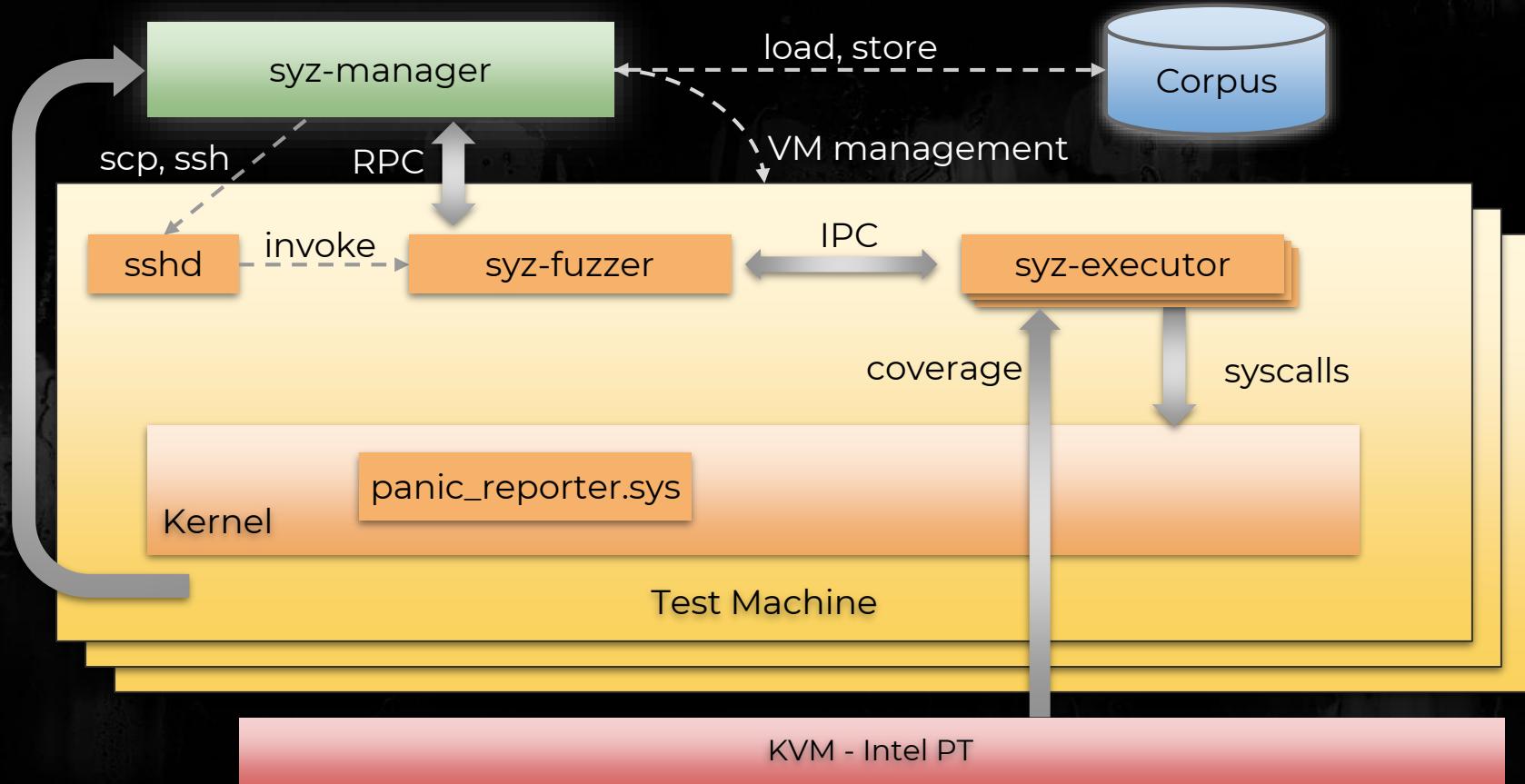
```
: mov rax, rcx
```



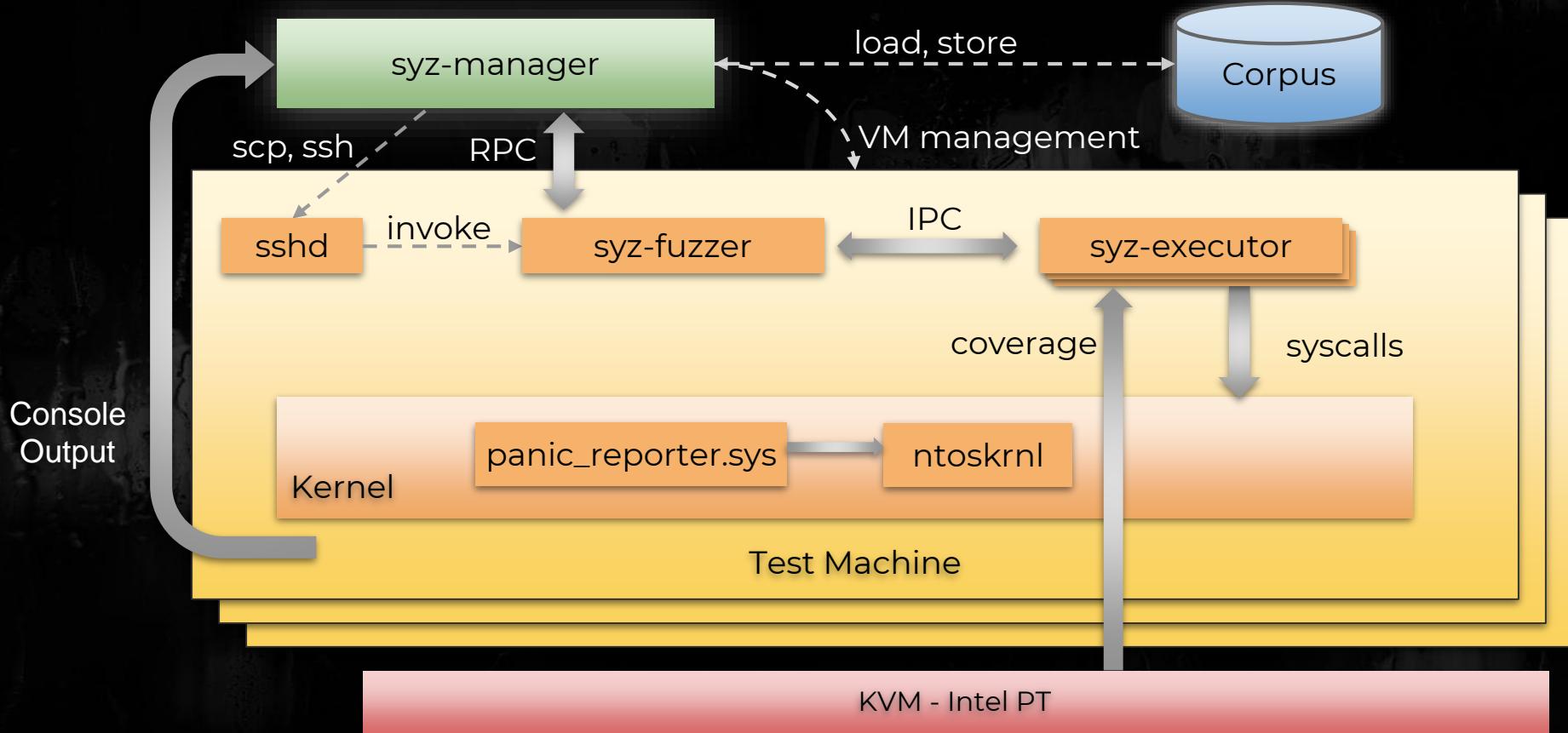
Crash detection



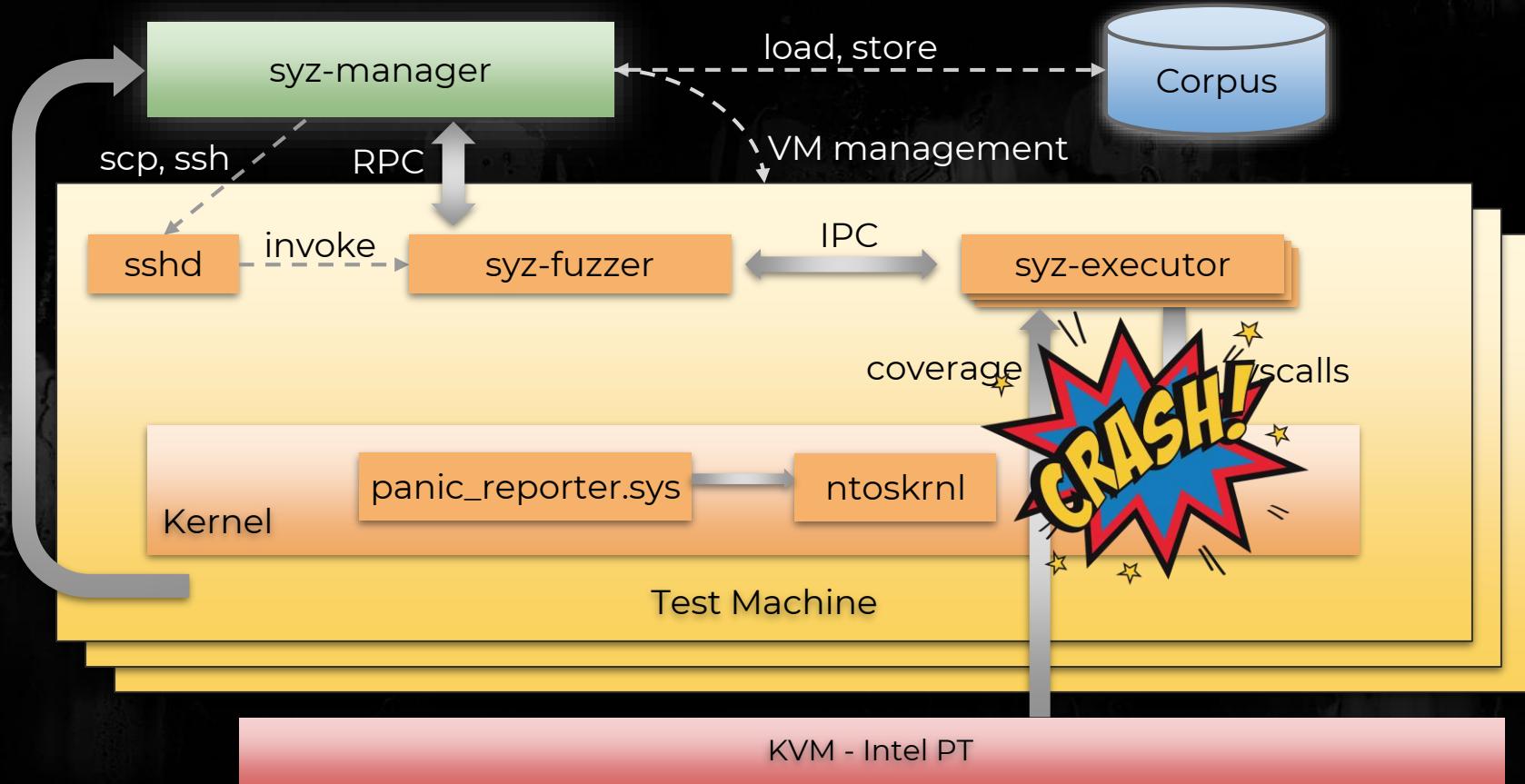
Crash detection



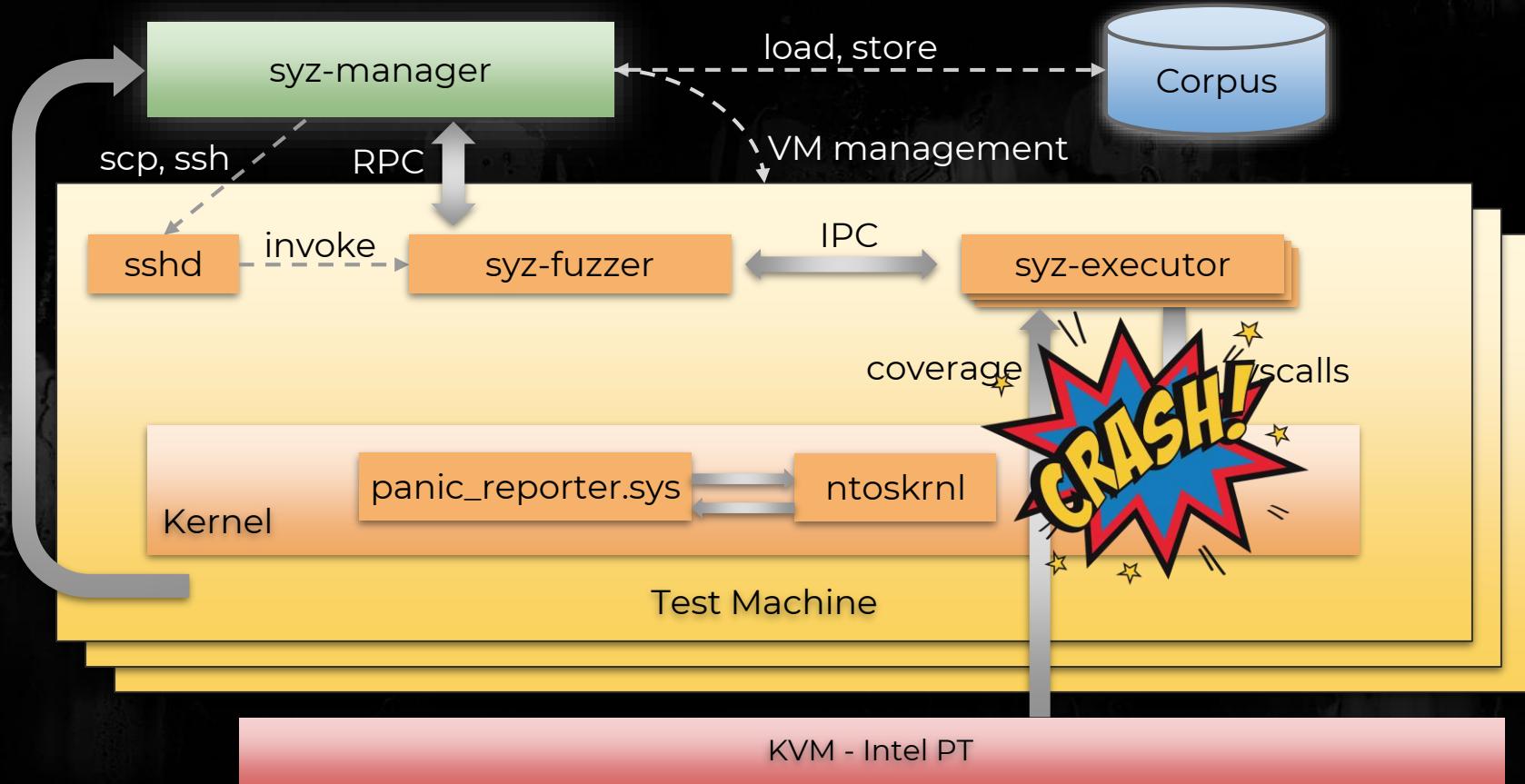
Crash detection



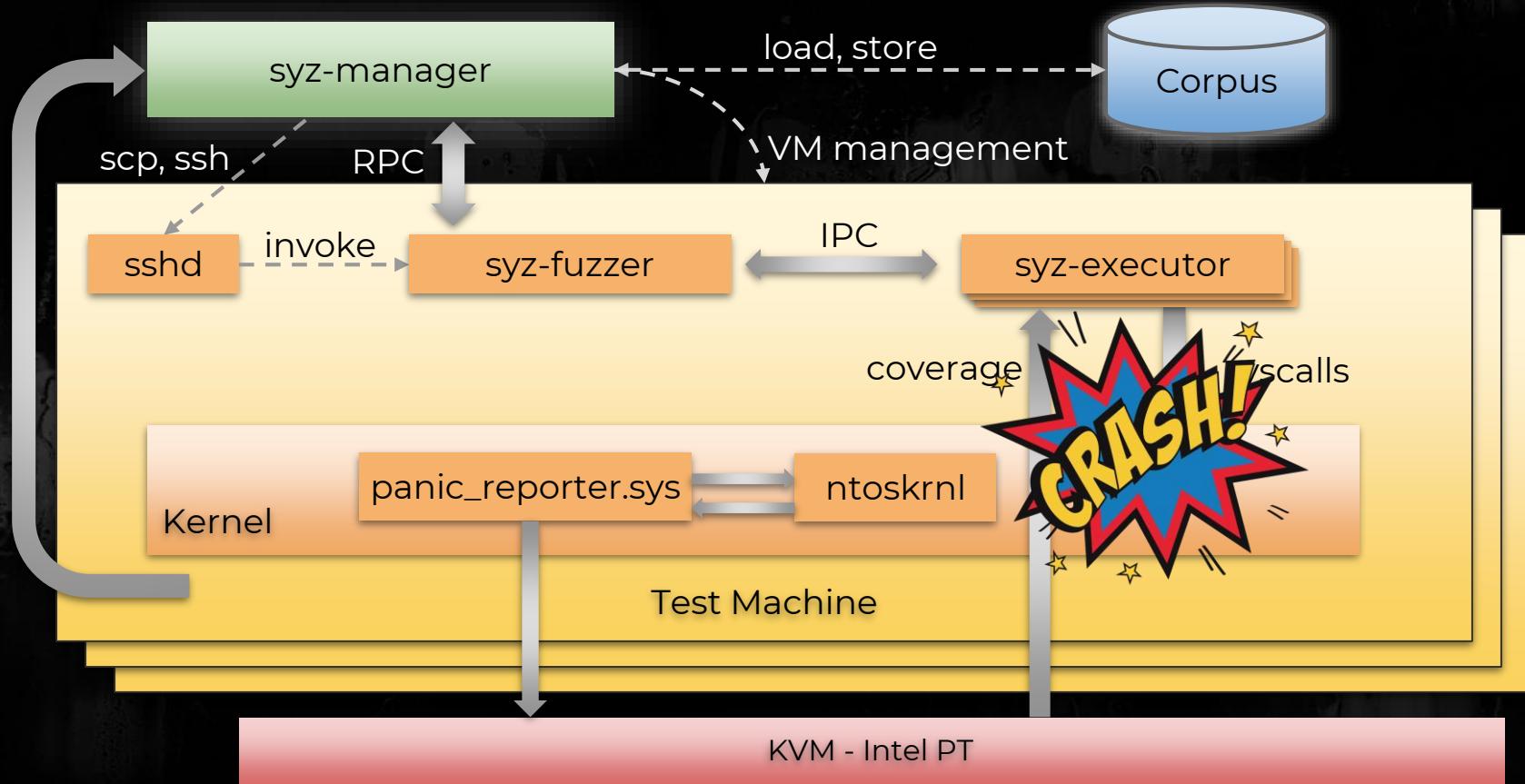
Crash detection



Crash detection



Crash detection



Crash De-duplication & symbolize #1

We run again and got new crash



Didn't work so well:

- There's a lot of garbage on the stack
- Lacks information to work on crashes that are hard to reproduce

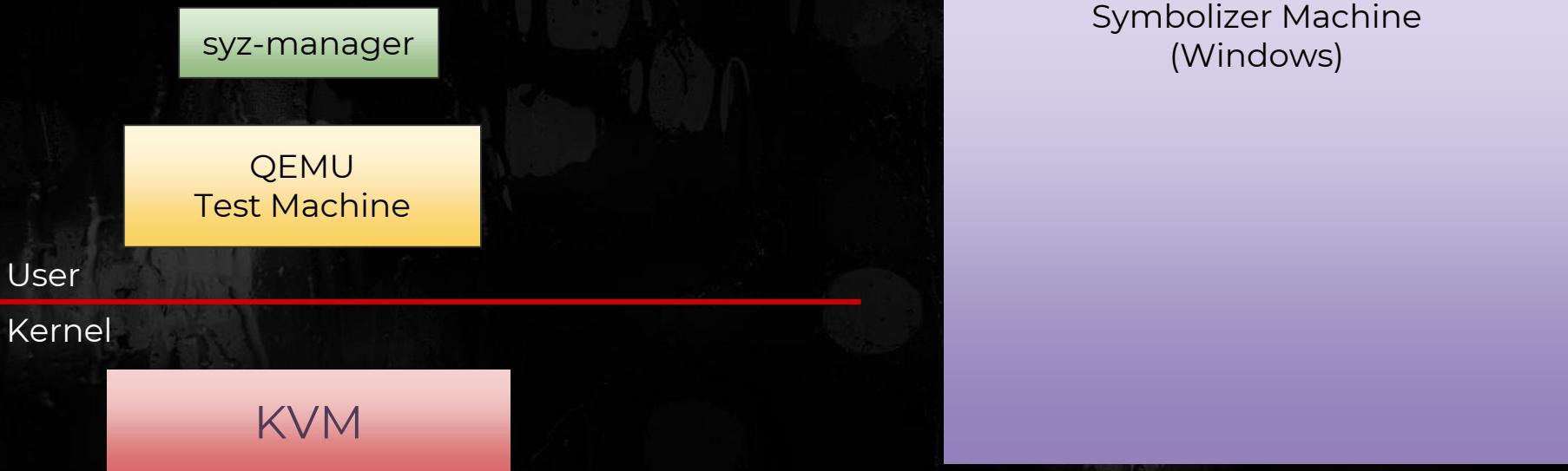
Crash De-duplication & symbolize #2

Using debugger!

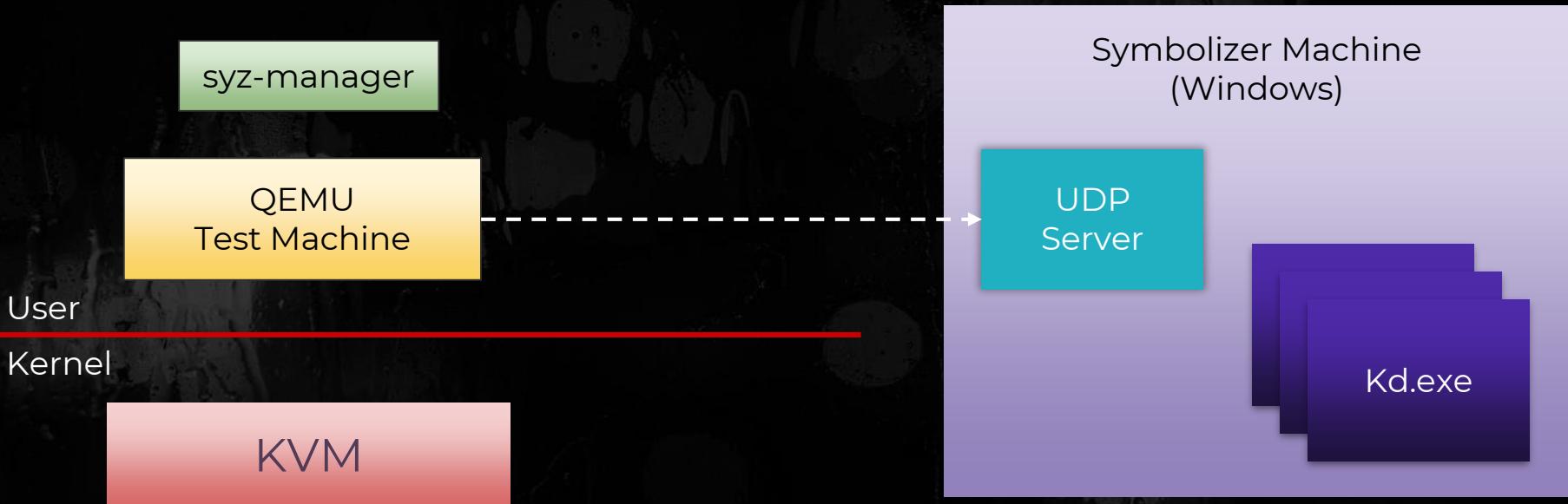
But WinDBG/Kd don't work very well on linux

- With wine kd would randomly crash
- Extracting crash information over wine took **5+ minutes**

Crash De-duplication & symbolize #2



Crash De-duplication & symbolize #2



Crash De-duplication & symbolize #3

How does kd/windbg is able to generate call stack? DbgHelp.dll

Crash De-duplication & symbolize #3

How does kd/windbg is able to generate call stack? DbgHelp.dll

```
BOOL IMAGEAPI StackWalk(
    DWORD                      MachineType,
    HANDLE                     hProcess,
    HANDLE                     hThread,
    LPSTACKFRAME              StackFrame,
    PVOID                      ContextRecord,
    PREAD_PROCESS_MEMORY_ROUTINE ReadMemoryRoutine,
    PFUNCTION_TABLE_ACCESS_ROUTINE FunctionTableAccessRoutine,
    PGET_MODULE_BASE_ROUTINE   GetModuleBaseRoutine,
    PTRANSLATE_ADDRESS_ROUTINE TranslateAddress
);
```

Crash De-duplication & symbolize #3

How does kd/windbg is able to generate call stack? DbgHelp.dll

```
BOOL IMAGEAPI StackWalk(
    DWORD
    HANDLE
    HANDLE
    LPSTACKFRAME
    PVOID
    PREAD_PROCESS_MEMORY_ROUTINE
    PFUNCTION_TABLE_ACCESS_ROUTINE
    PGET_MODULE_BASE_ROUTINE
    PTRANSLATE_ADDRESS_ROUTINE
);
{
    MachineType,
    hProcess,
    hThread,
    StackFrame,
    ContextRecord,
    ReadMemoryRoutine,
    FunctionTableAccessRoutine,
    GetModuleBaseRoutine,
    TranslateAddress
};
```

Crash De-duplication & symbolize #3

How does kd/windbg is able to generate call stack? DbgHelp.dll

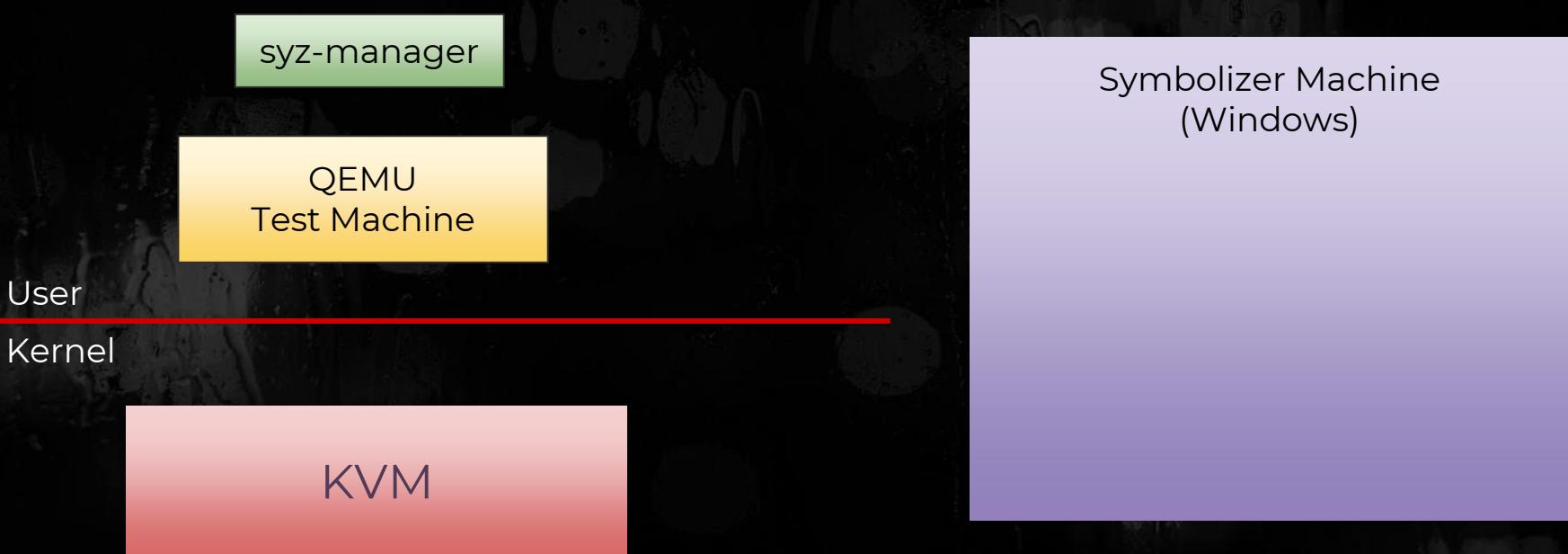
```
BOOL IMAGEAPI StackWalk(
    DWORD
    HANDLE
    HANDLE
    LPSTACKFRAME
    PVOID
    PREAD_PROCESS_MEMORY_ROUTINE
    PFUNCTION_TABLE_ACCESS_ROUTINE
    PGET_MODULE_BASE_ROUTINE
    PTRANSLATE_ADDRESS_ROUTINE
    );
    MachineType,
    hProcess,
    hThread,
    StackFrame,
    ContextRecord,
    ReadMemoryRoutine,
    FunctionTableAccessRoutine,
    GetModuleBaseRoutine,
    TranslateAddress
```

Crash De-duplication & symbolize #3

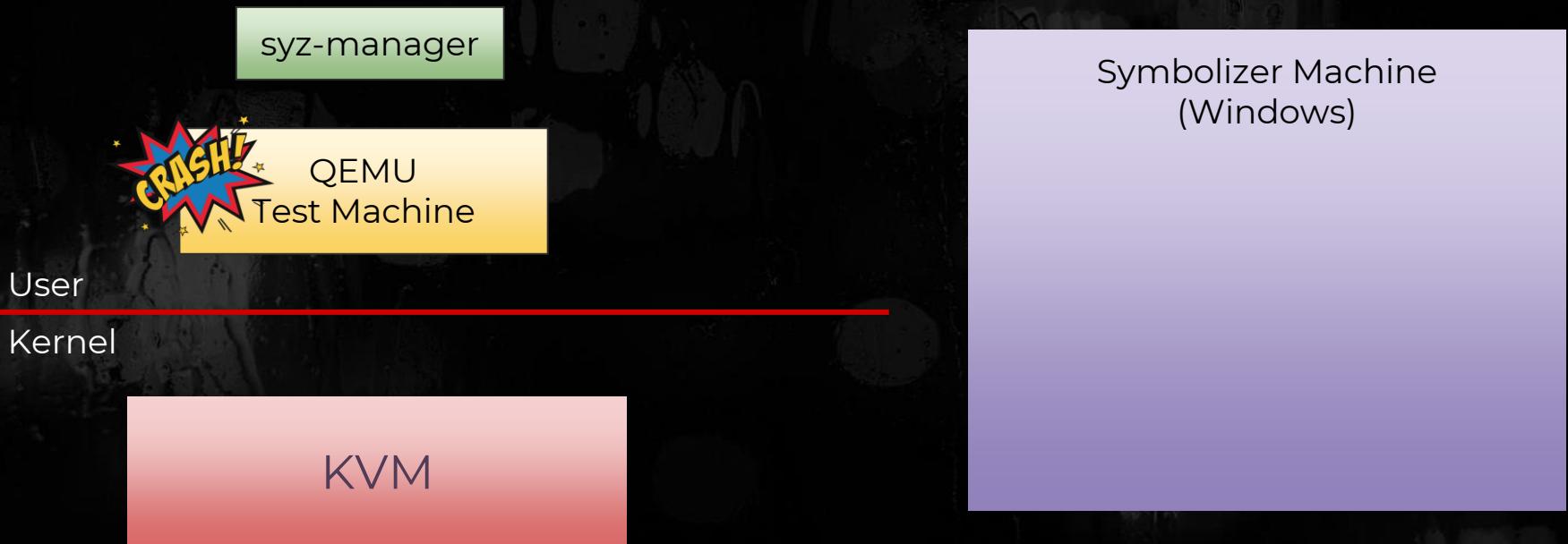
How does kd/windbg is able to generate call stack? DbgHelp.dll

```
BOOL IMAGEAPI StackWalk(  
    DWORD  
    HANDLE  
    HANDLE  
    LPSTACKFRAME  
    PVOID  
    PREAD_PROCESS_MEMORY_ROUTINE  
    PFUNCTION_TABLE_ACCESS_ROUTINE  
    PGET_MODULE_BASE_ROUTINE  
    PTRANSLATE_ADDRESS_ROUTINE  
);  
    MachineType,  
    hProcess,  
    hThread,  
    StackFrame,  
    ContextRecord,  
    ReadMemoryRoutine,  
    FunctionTableAccessRoutine,  
    GetModuleBaseRoutine,  
    TranslateAddress
```

Crash De-duplication & symbolize #3



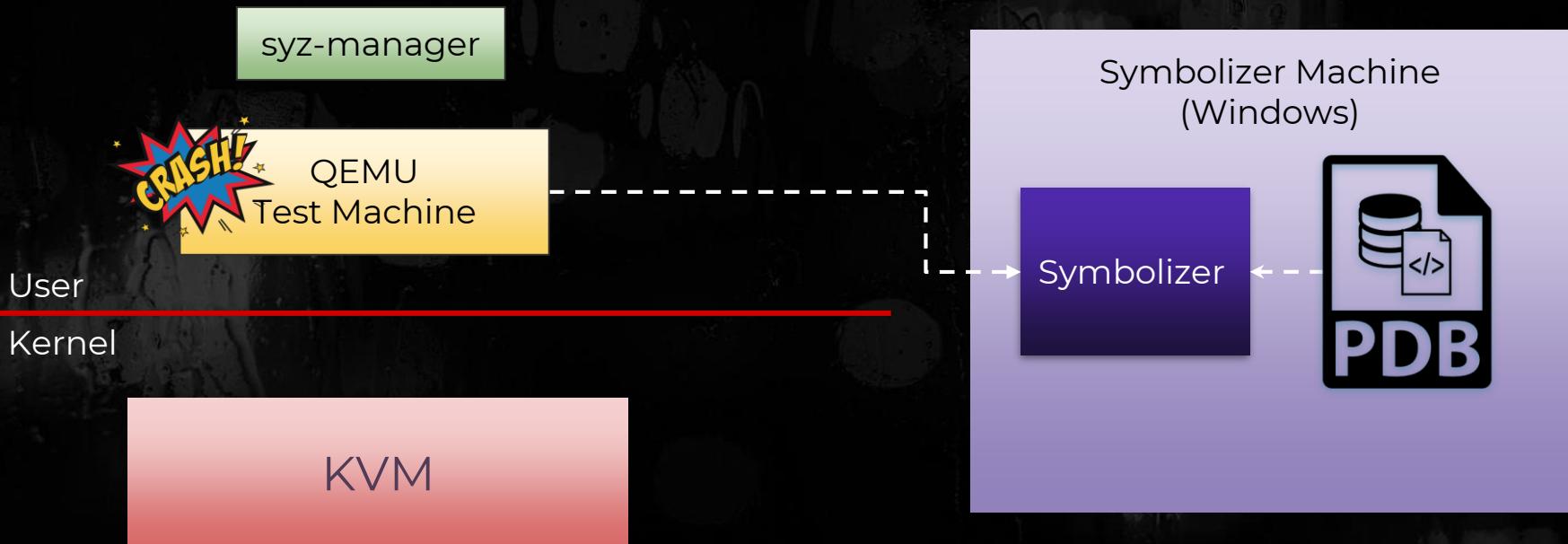
Crash De-duplication & symbolize #3



Crash De-duplication & symbolize #3



Crash De-duplication & symbolize #3



Syzkaller - WSL Symbolizer



```
0x98766 0x422f4 0x42024 0x41dc0 0x41d60 0x3f9fc
```

Syzkaller - WSL Symbolizer



0x98766 0x422f4 0x42024 0x41dc0 0x41d60 0x3f9fc



Callstack:

```
[0] - ntoskrnl!KiScanBugCheckCallbackList+0000009b
[1] - ntoskrnl!KeBugCheck2+00000d4e
[2] - ntoskrnl!KeBugCheckEx+00000107
[3] - ntoskrnl!KiBugCheckDispatch+00000069
[4] - ntoskrnl!KiSystemServiceHandler+0000007c
[5] - ntoskrnl!RtlpExecuteHandlerForException+0000000f
[6] - ntoskrnl!RtlDispatchException+00000430
[7] - ntoskrnl!KiDispatchException+00000144
[8] - ntoskrnl!KiExceptionDispatch+000000c2
[9] - ntoskrnl!KiPageFault+00000428
[10] - ntoskrnl!RtlpHpVsChunkSplit+000003c5
[11] - ntoskrnl!RtlpHpVsContextAllocateInternal+00000028c
[12] - ntoskrnl!RtlpHpVsContextAllocate+00000046
[13] - ntoskrnl!RtlpAllocateHeapInternal+00000055
[14] - ntoskrnl!RtlAllocateHeap+00000037
[15] - win32kfull!ClassAlloc+00000039
[16] - win32kfull!InternalRegisterClassEx+0000014e
[17] - win32kfull!_RegisterClassEx+00000085
[18] - win32kfull!NtUserRegisterClassExWOW+0000059f
[19] - ntoskrnl!KiSystemServiceCopyEnd+00000025
[20] - 7ff992c825a4
[21] - 7ff993d769c5
[22] - 10
[23] - 200000380
[24] - 45efc00
```

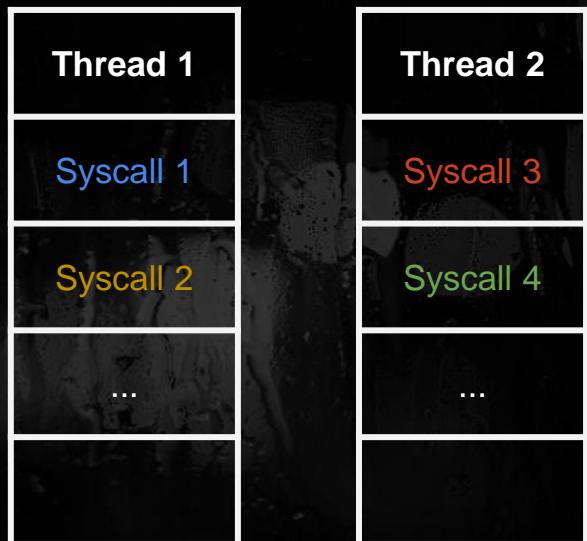
Crash De-duplication & symbolize #4 - WIP

We started working on a kd-go:

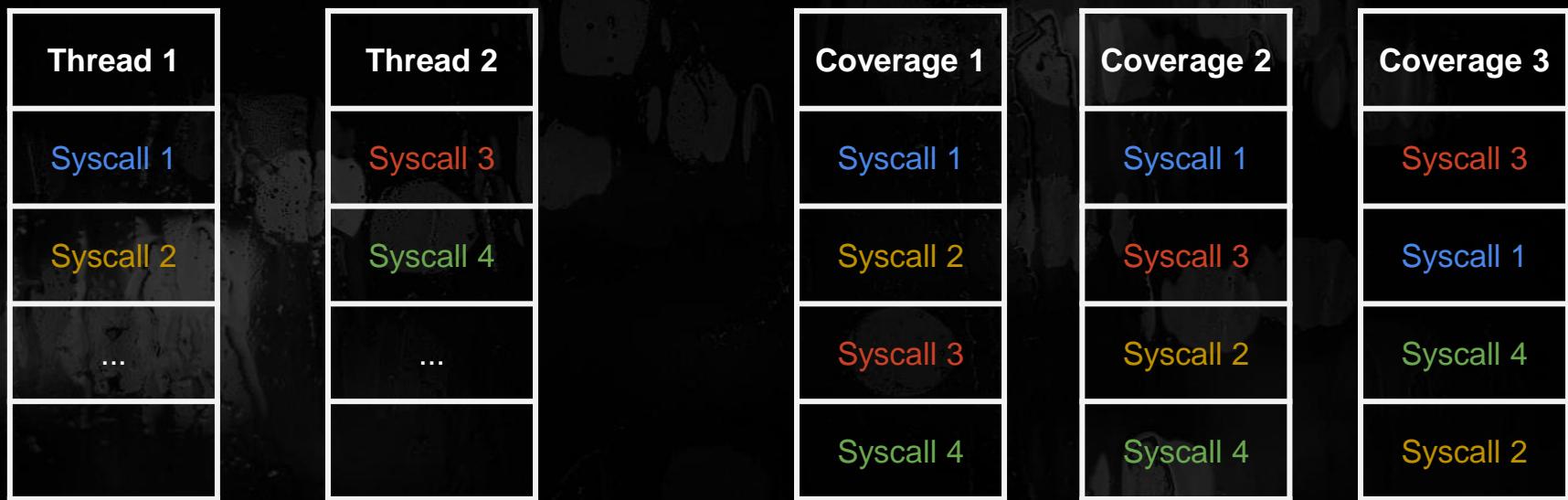
- PDB parser/fetcher
- Stack unwinder
- Kd serial client (Kdnet will be available soon)

Compiled into syzkaller to symbolize directly

Coverage stability



Coverage stability



Coverage stability

We need to separate coverage buffer for each thread like KCOV

For that we need:

- KVM Thread tracking (swapping buffers)
- Handle aware API in KVM

Thread tracking

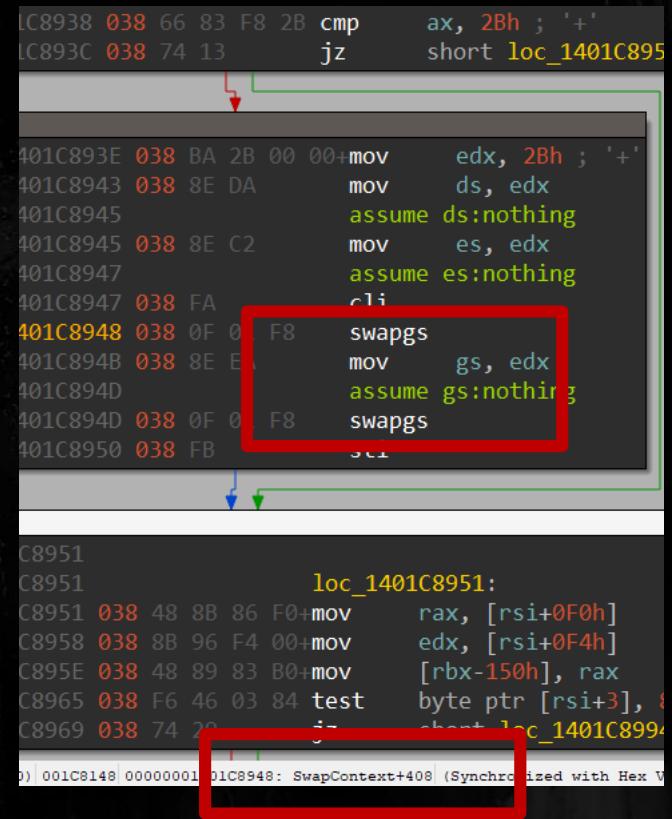
```
; Exported entry 1049. KeGetCurrentThread
; Exported entry 1679. PsGetCurrentThread

; PKTHREAD KeGetCurrentThread(void)
public KeGetCurrentThread
KeGetCurrentThread proc near
00 65 48 8B 04+mov    rax, gs:188h    ; KeGetCurrentThread
00 C3                 retn
KeGetCurrentThread endp

0 0000000140031E80: KeGetCurrentThread (Synchronized with Hex View-1)
```

Thread tracking

```
; Exported entry 1049. KeGetCurrentThread  
; Exported entry 1679. PsGetCurrentThread  
  
; PKTHREAD KeGetCurrentThread(void)  
public KeGetCurrentThread  
KeGetCurrentThread proc near  
    KeGetCurrentThread  
    proc near  
        mov     rax, gs:188h      ; KeGetCurrentThread  
        retn  
    KeGetCurrentThread endp  
  
0| 0000000140031E80: KeGetCurrentThread (Synchronized with Hex View-1)
```



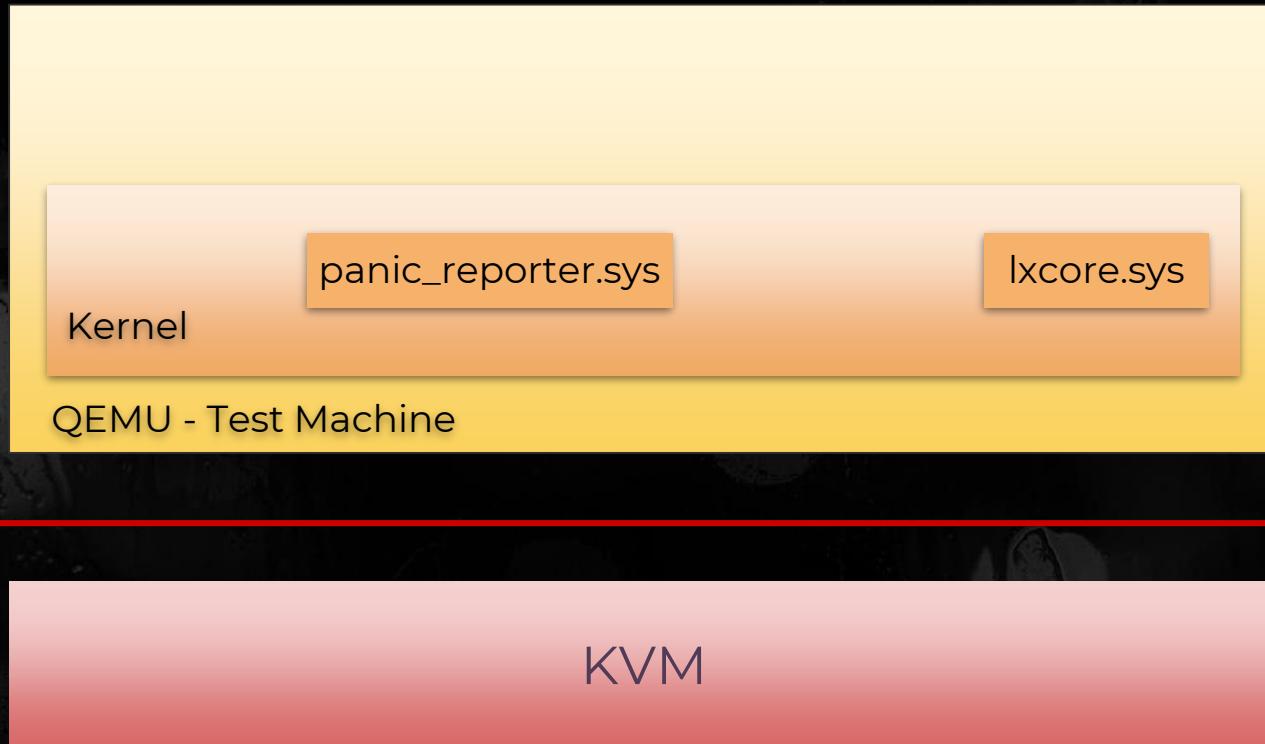
```
LC8938 038 66 83 F8 2B cmp     ax, 2Bh ; '+'  
LC893C 038 74 13     jz      short loc_1401C895  
  
401C893E 038 BA 2B 00 00+mov     edx, 2Bh ; '+'  
401C8943 038 8E DA      mov     ds, edx  
401C8945          assume ds:nothing  
401C8945 038 8E C2      mov     es, edx  
401C8947          assume es:nothing  
401C8947 038 FA      cli  
401C8948 038 0F 01 F8 swapgs  
401C894B 038 8E EA      mov     gs, edx  
401C894D          assume gs:nothing  
401C894D 038 0F 01 F8 swapgs  
401C8950 038 FB      set  
  
C8951  
C8951          loc_1401C8951:  
C8951 038 48 8B 86 F0+mov     rax, [rsi+0F0h]  
C8958 038 8B 96 F4 00+mov     edx, [rsi+0F4h]  
C895E 038 48 89 83 B0+mov     [rbx-150h], rax  
C8965 038 F6 46 03 84 test    byte ptr [rsi+3],  
C8969 038 74 20     jne     short loc_1401C8994  
  
0| 001C8148 00000001 01C8948: SwapContext+408 (Synchronized with Hex View-1)
```

Thread tracking

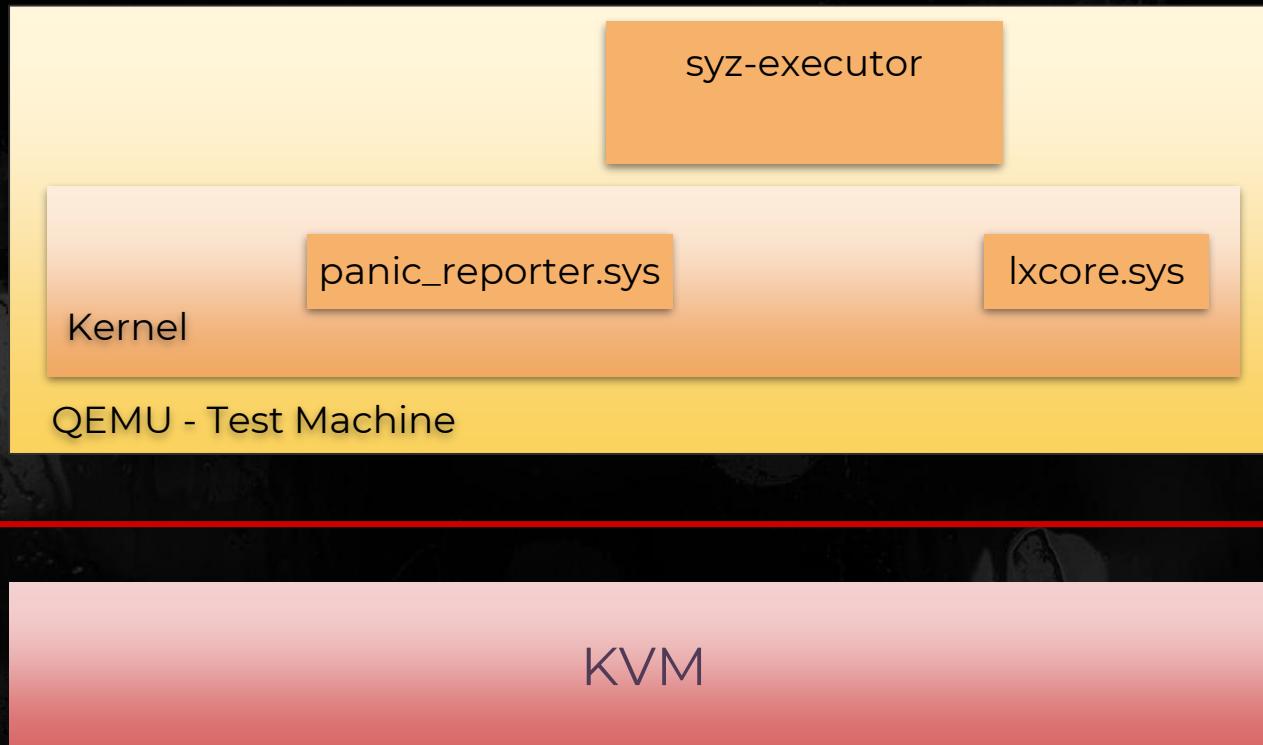
Guest driver would issue a hypercall on thread termination

```
msr_bitmap = vmx->vmcs01.msr_bitmap;
vmx_disable_intercept_for_msр(msr_bitmap, MSR_IA32_TSC, MSR_TYPE_R);
vmx_disable_intercept_for_msр(msr_bitmap, MSR_FS_BASE, MSR_TYPE_RW);
vmx_disable_intercept_for_msр(msr_bitmap, MSR_GS_BASE, MSR_TYPE_RW);
vmx_disable_intercept_for_msр(msr_bitmap, MSR_KERNEL_GS_BASE, MSR_TYPE_RW);
vmx_disable_intercept_for_msр(msr_bitmap, MSR_IA32_SYSENTER_CS, MSR_TYPE_RW);
vmx_disable_intercept_for_msр(msr_bitmap, MSR_IA32_SYSENTER_ESP, MSR_TYPE_RW);
vmx_disable_intercept_for_msр(msr_bitmap, MSR_IA32_SYSENTER_EIP, MSR_TYPE_RW);
if (kvm_cstate_in_guest(vcpu->kvm)) {
    vmx_disable_intercept_for_msр(msr_bitmap, MSR_CORE_C1_RES, MSR_TYPE_R);
    vmx_disable_intercept_for_msр(msr_bitmap, MSR_CORE_C3_RESIDENCY, MSR_TYPE_R)
```

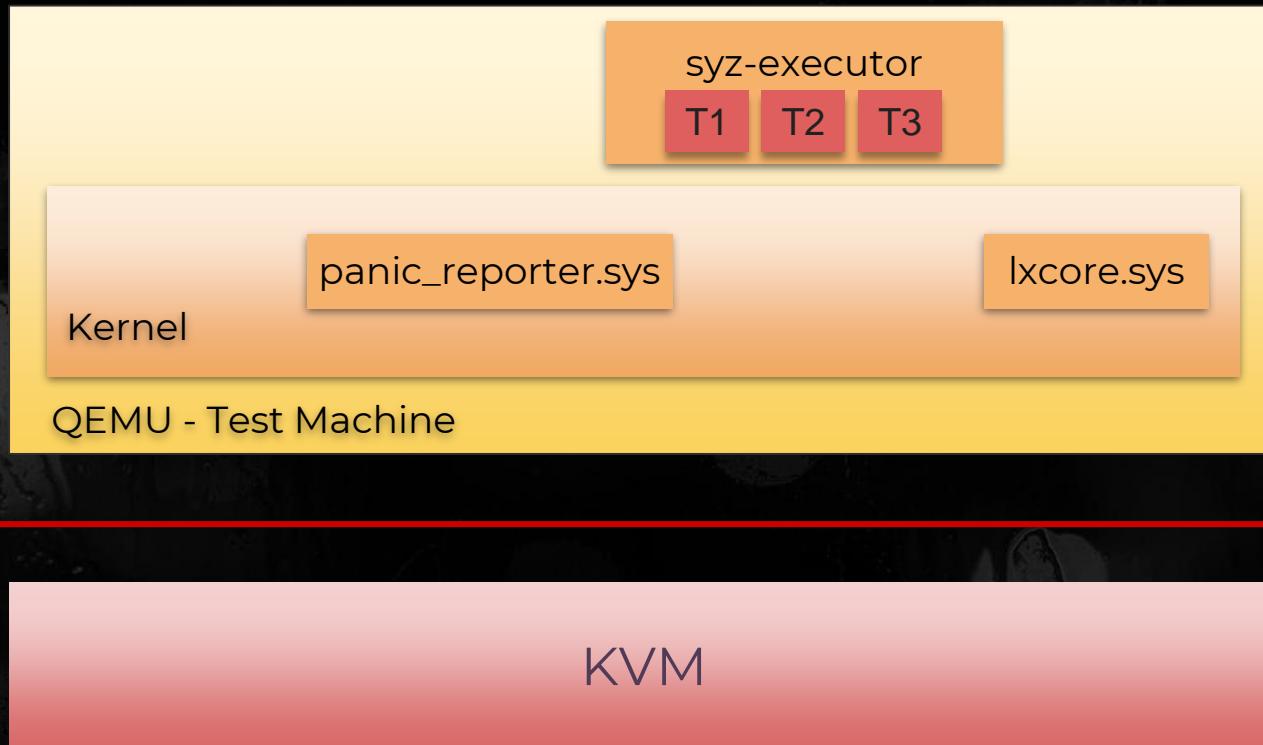
Thread tracking – putting all together



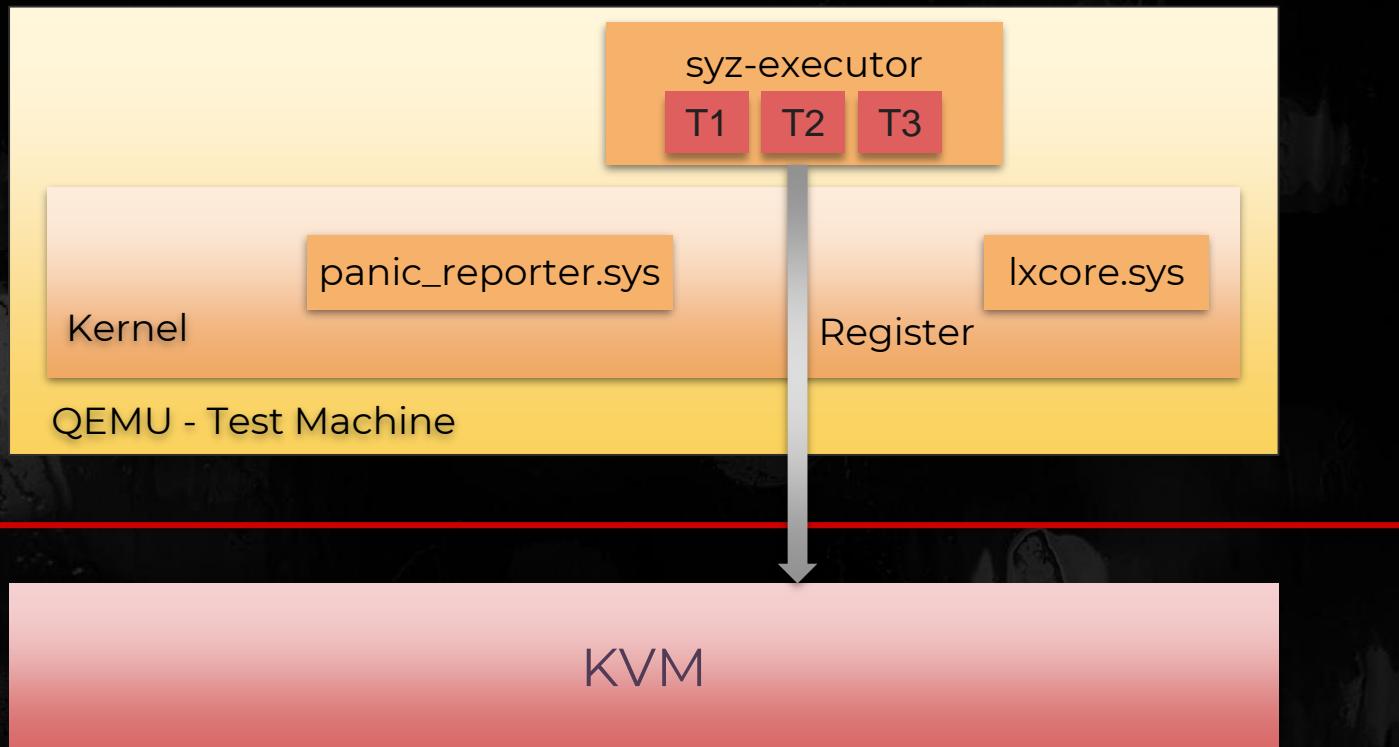
Thread tracking – putting all together



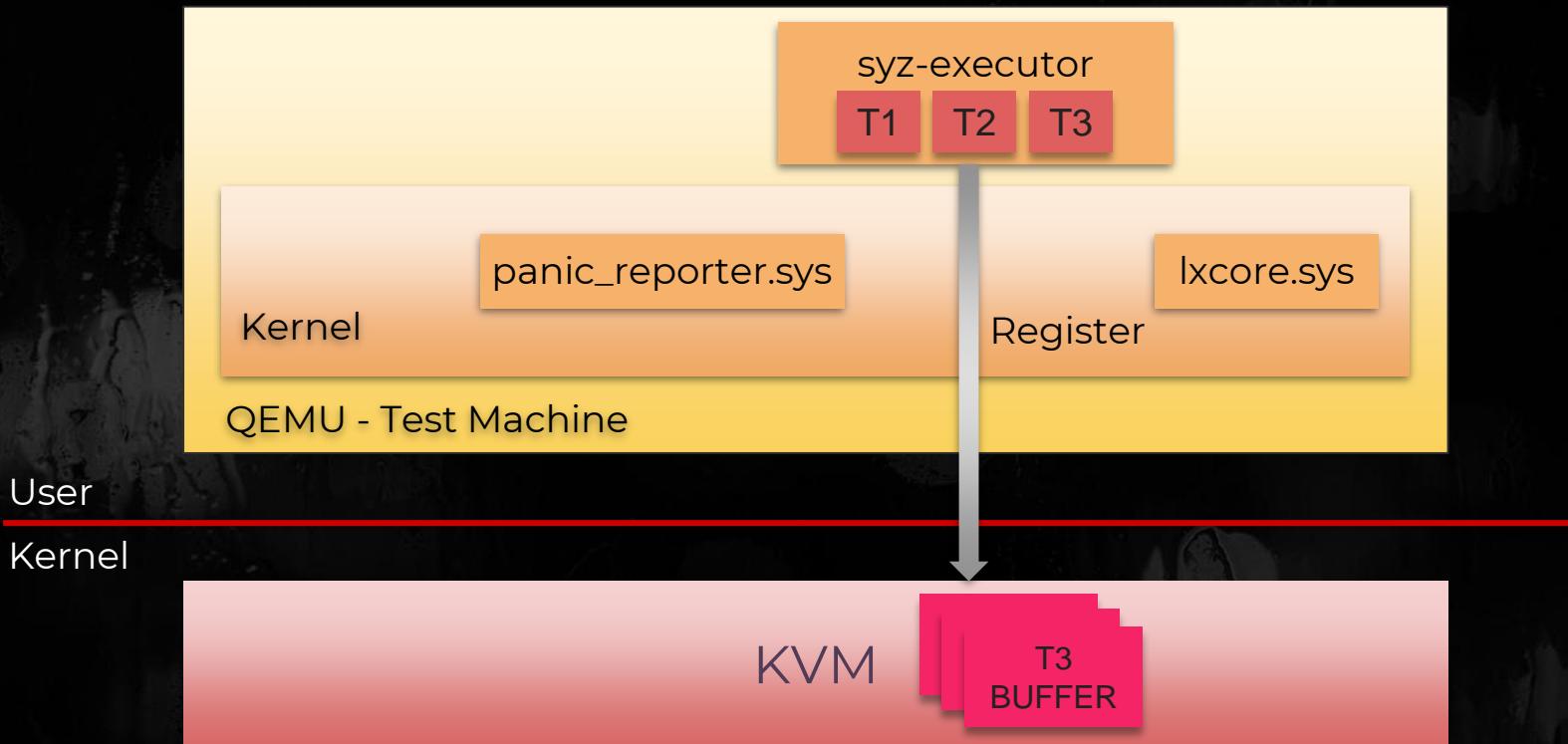
Thread tracking – putting all together



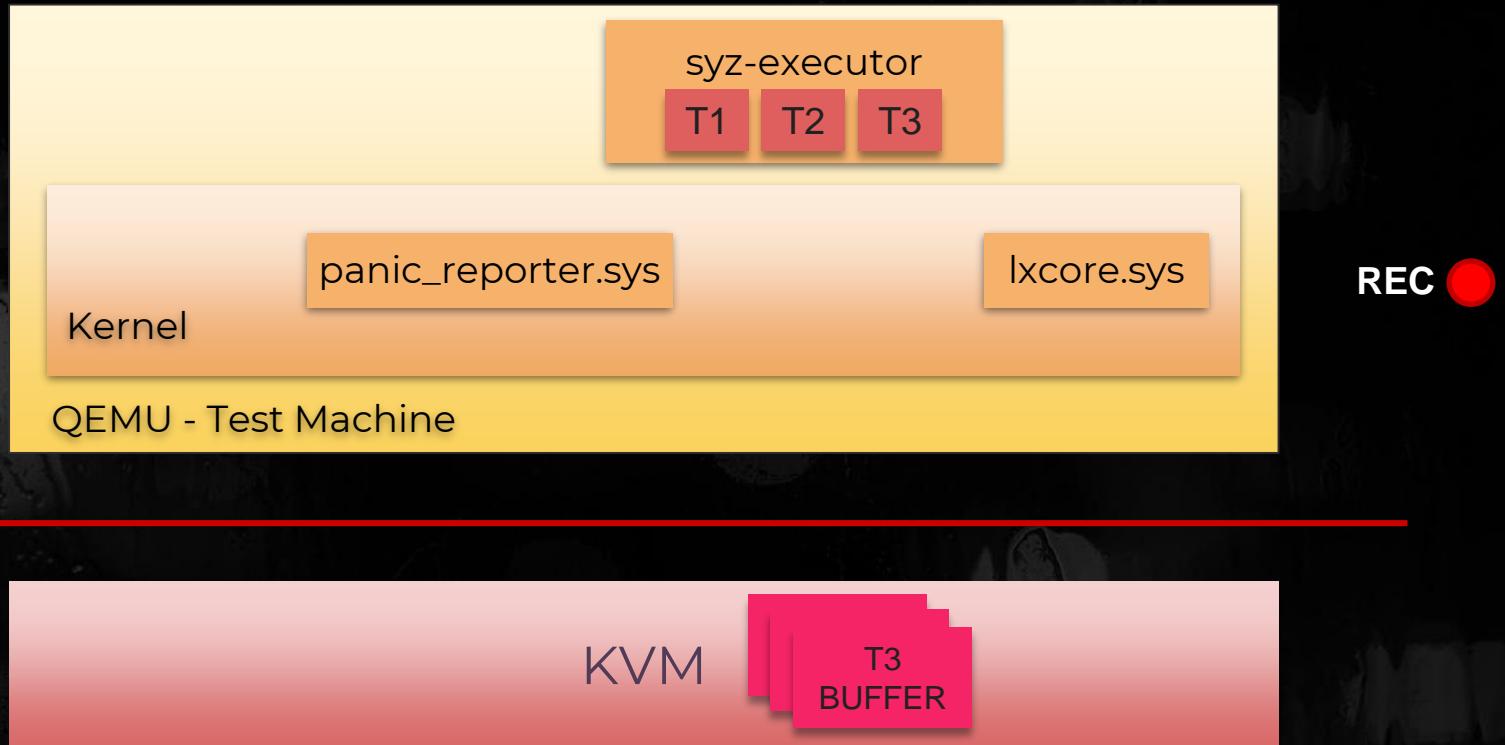
Thread tracking – putting all together



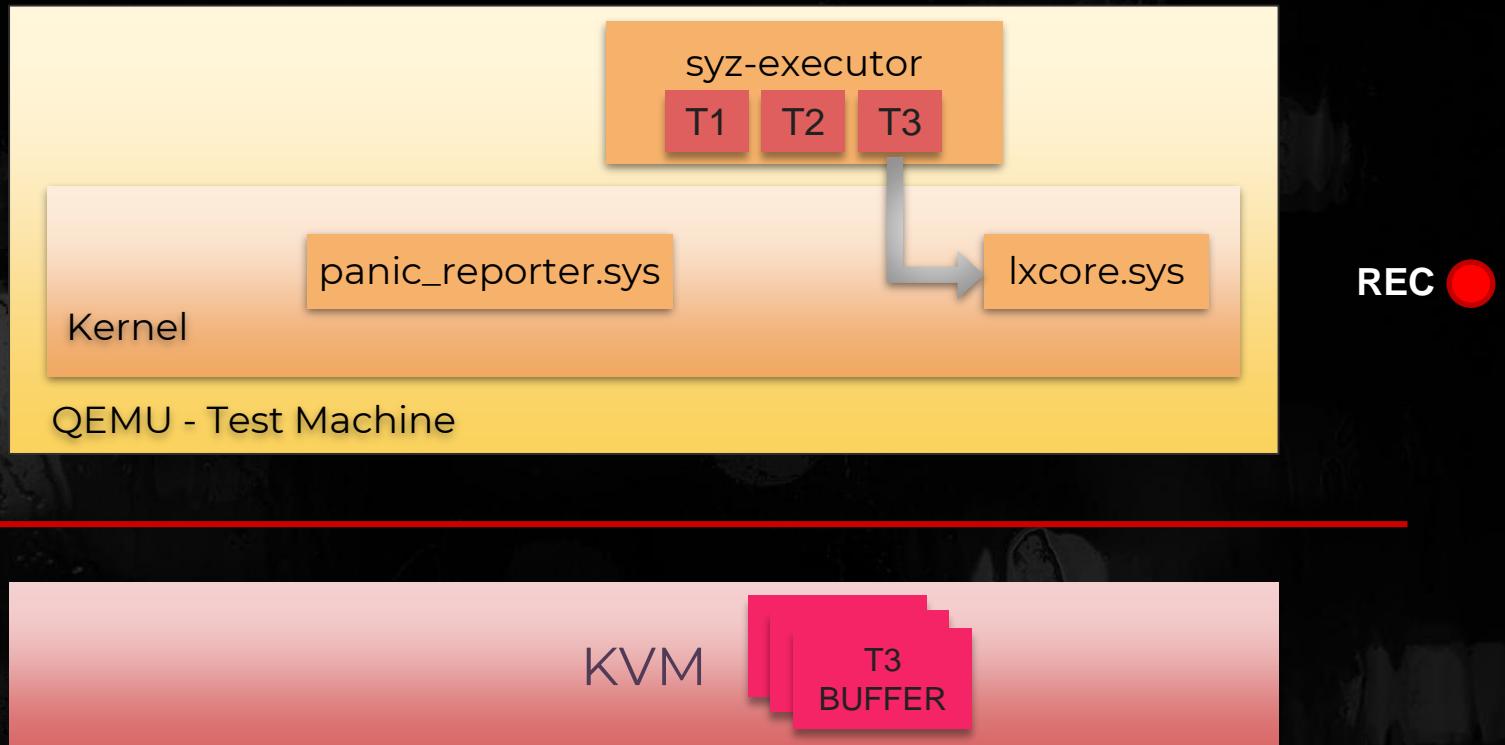
Thread tracking – putting all together



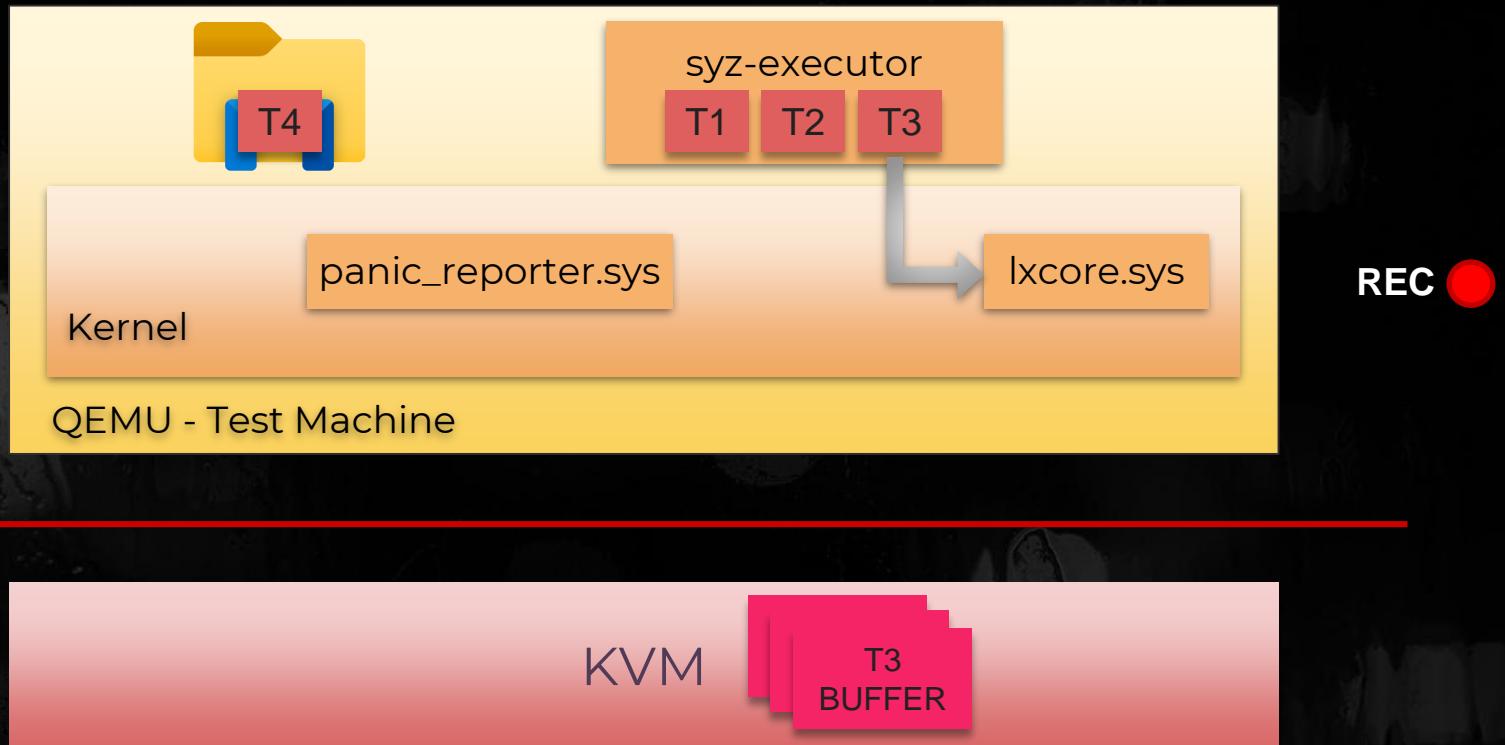
Thread tracking – putting all together



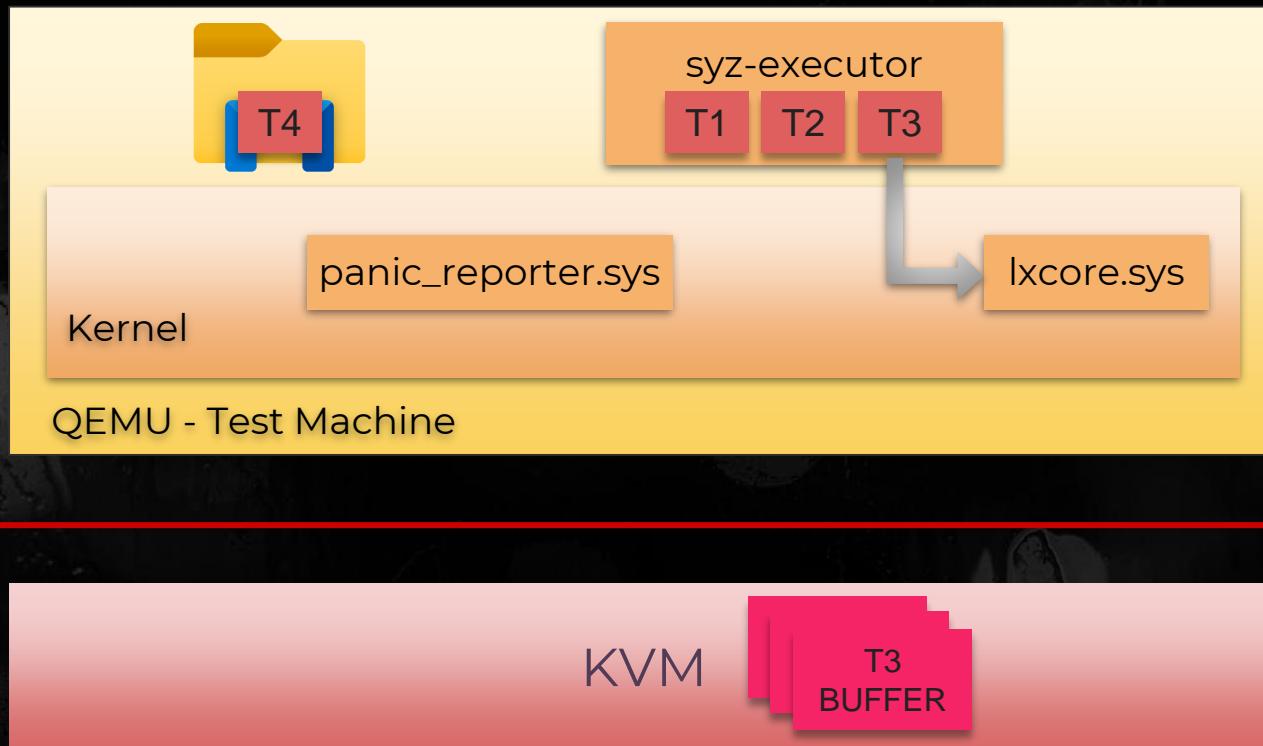
Thread tracking – putting all together



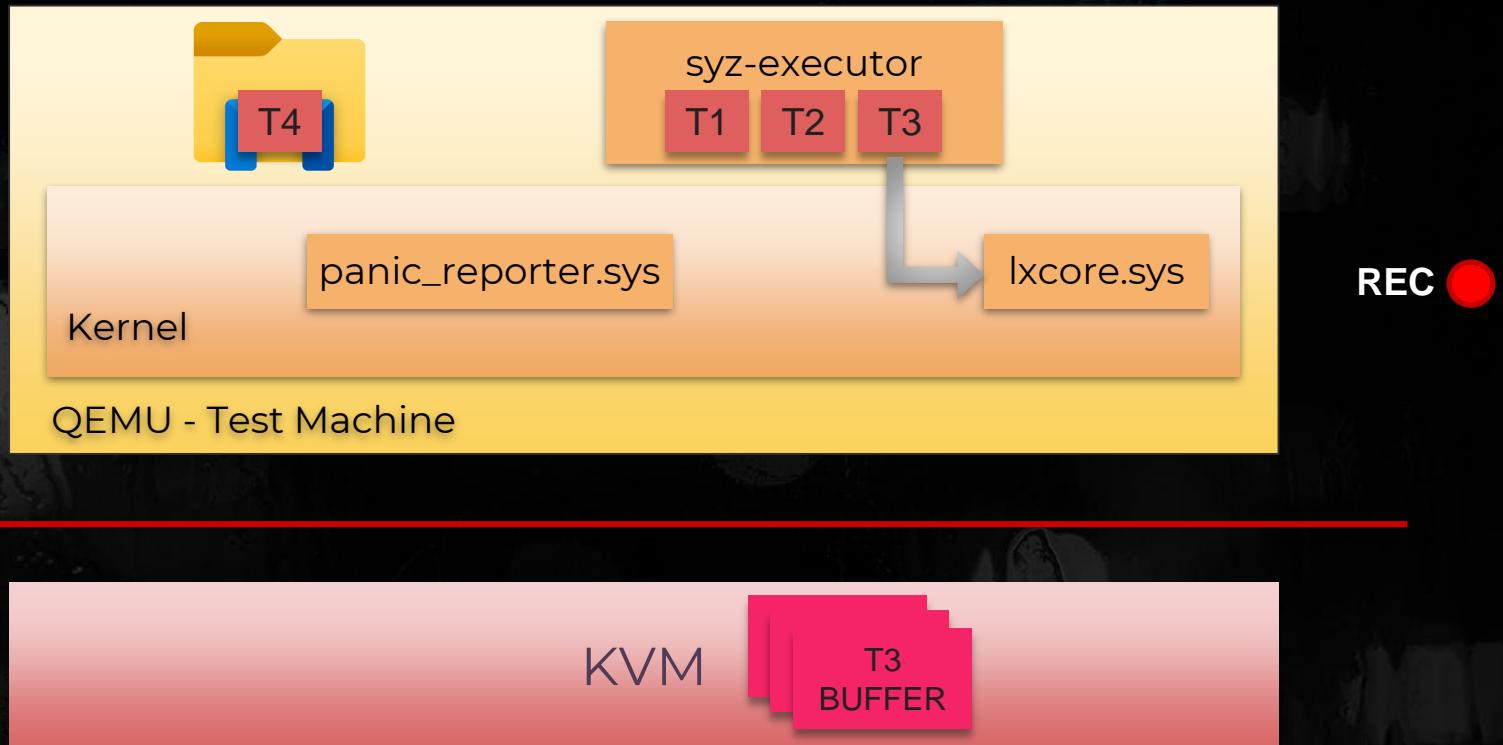
Thread tracking – putting all together



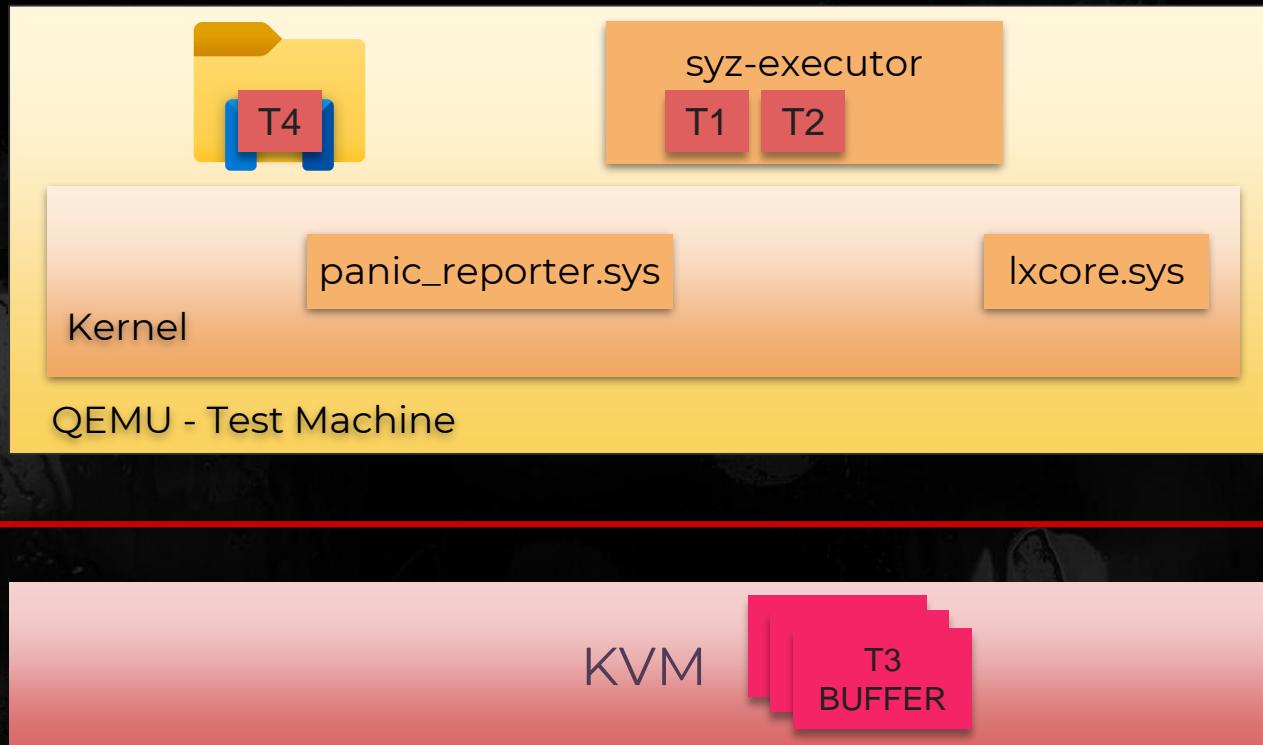
Thread tracking – putting all together



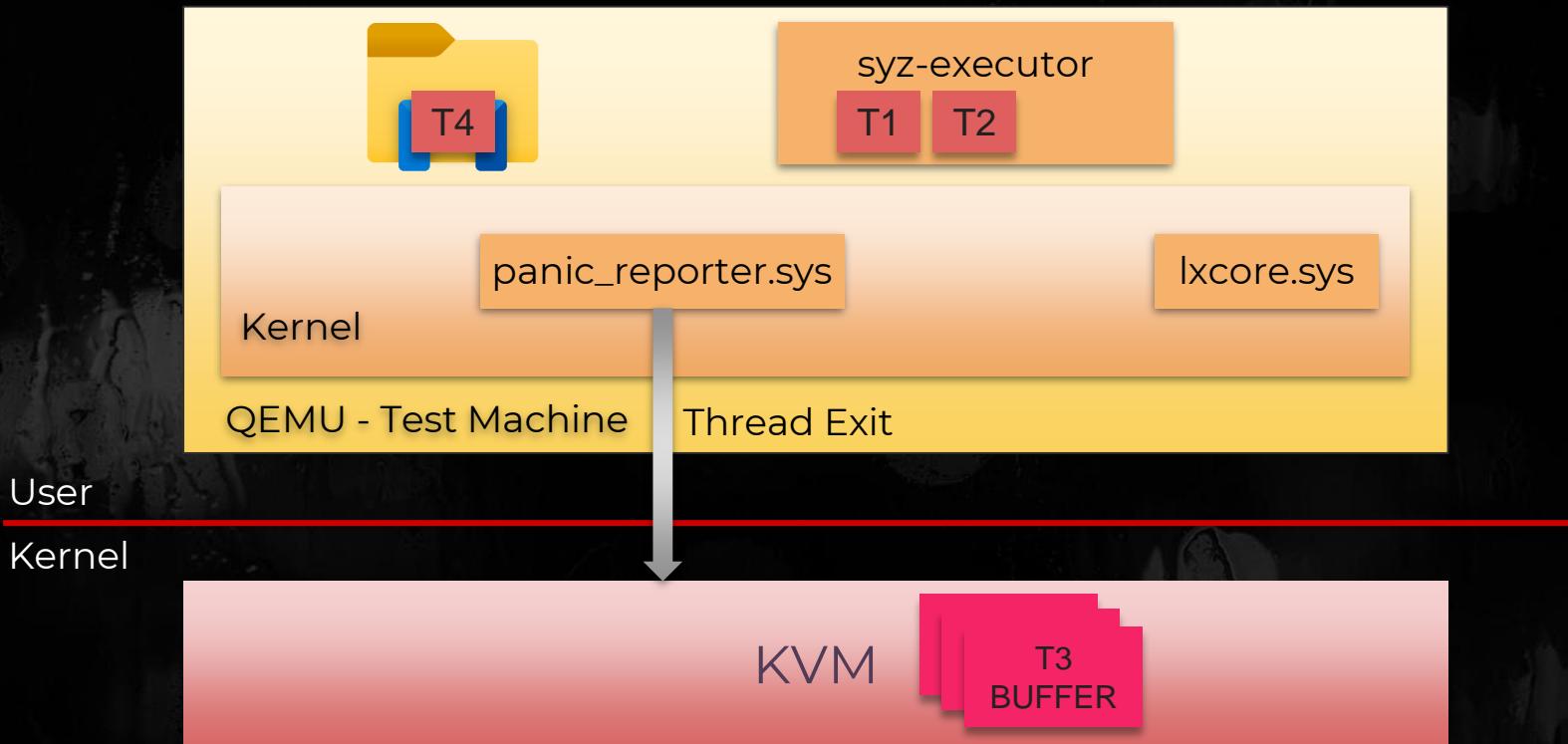
Thread tracking – putting all together



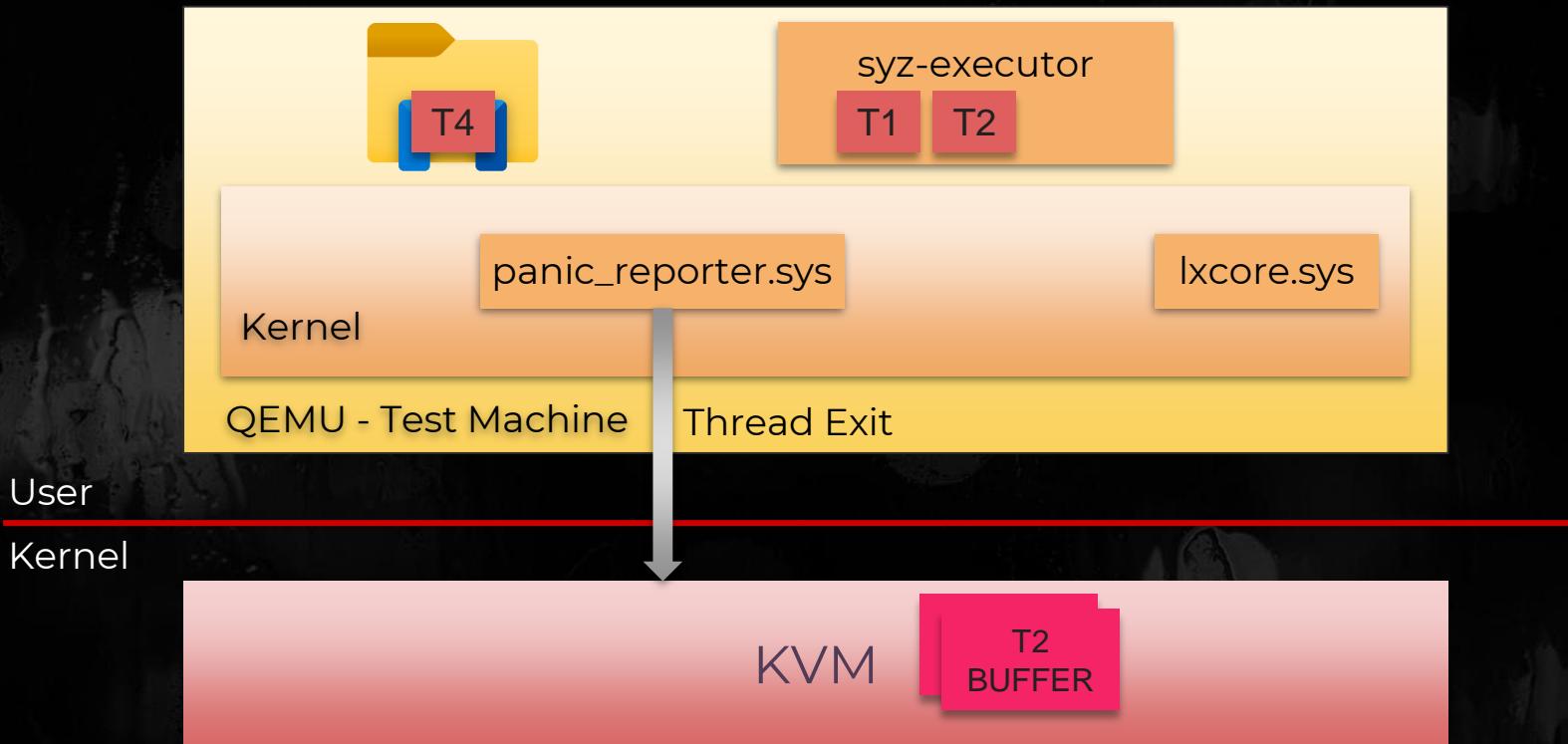
Thread tracking – putting all together



Thread tracking – putting all together



Thread tracking – putting all together

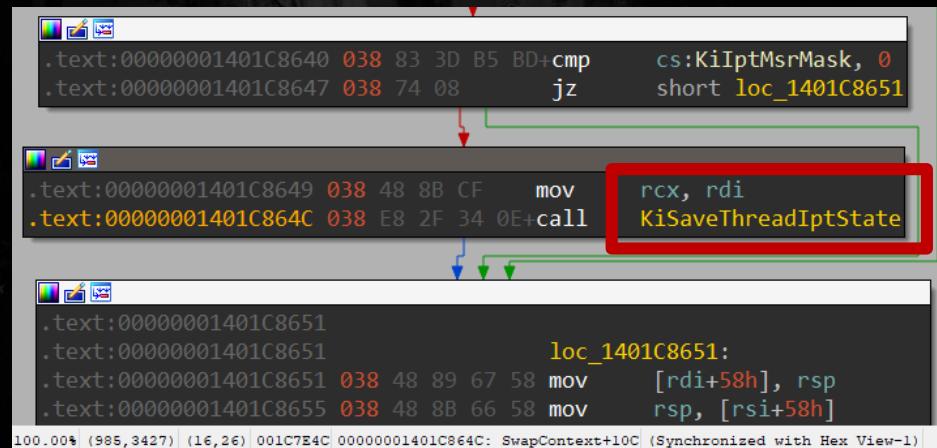


Coverage stability - in hindsight

SwapContext references ipt.sys

<https://github.com/ionescu007/winipt>

- Supports per thread tracking
- Doesn't support kernel tracing



We could have:

- Purchased an Ice Lake server (vmx support for intel-pt)
- Wrote small patch to IPT.sys (to allow kernel tracing)

Glitches be crazy

KnownDLLs	Winlogon	Winsock Providers	Print Monitors	LSA Providers	Network Providers	WMI		
Everything	Logon	Explorer	Internet Explorer	Scheduled Tasks	Services	Drivers	Codecs	Boot Execute
Autorun Entry	Description	Publisher	Image Path	Timestamp				
 HKCU\Software\Microsoft\Windows\CurrentVersion\Run				21/12/2018 3:32				
<input checked="" type="checkbox"/>  OneDrive	Microsoft OneDrive	Microsoft Corporation	c:\users\user\appdata\local\microso...	07/08/1973 7:28				
 HKLM\Software\Microsoft\Active Setup\Installed Components				12/04/2018 1:16				

Glitches be crazy

KnownDLLs	Winlogon	Winsock Providers	Print Monitors	LSA Providers	Network Providers	WMI				
Everything	Logon	Explorer	Internet Explorer	Scheduled Tasks	Services	Drivers	Codecs	Boot Execute		
Autorun Entry		Description			Publisher		Image Path			
HKCU\Software\Microsoft\Windows\CurrentVersion\Run						21/12/2018 3:32				
<input checked="" type="checkbox"/> OneDrive			Microsoft OneDrive			Microsoft Corporation				
HKLM\Software\Microsoft\Active Setup\Installed Components						c:\users\user\appdata\local\microso... 07/08/1973 7:28				
Windows Time			Maintains d...	Manual (Trigger Start)	Local Service					
Windows Update			Enables the ...	Manual (Trigger Start)	Local System					
Windows Update Medic Service			Enables rem...	Manual	Local System					

Glitches be crazy

KnownDLLs	Winlogon	Winsock Providers	Print Monitors	LSA Providers	Network Providers	WMI				
Everything	Logon	Explorer	Internet Explorer	Scheduled Tasks	Services	Drivers	Codecs	Boot Execute		
Autorun Entry		Description			Publisher		Image Path			
HKCU\Software\Microsoft\Windows\CurrentVersion\Run										
OneDrive			Microsoft OneDrive			Microsoft Corporation				
HKLM\Software\Microsoft\Active Setup\Installed Components						c:\users\user\appdata\local\microso...				
Windows Time			Maintains d...	Manual (Trigger Start)	Local Service					
Windows Update			Enables the ...	Manual (Trigger Start)	Local System					
Windows Update Medic Service			Enables rem...	Manual	Local System					
Allow antimalware service to startup with normal priority				Not configured	No					
Turn off Windows Defender Antivirus				Enabled	No					
Configure local administrator merge behavior for lists				Not configured	No					

Glitches be crazy

A screenshot of a Windows system monitor or task manager interface. The top navigation bar includes tabs for KnownDLLs, Winlogon, Winsock Providers, Print Monitors, LSA Providers, Network Providers, WMI, Everything, Logon, Explorer, Internet Explorer, Scheduled Tasks, Services, Drivers, Codecs, and Boot Execute. Below this is a table with columns for Autorun Entry, Description, Publisher, Image Path, and Timestamp.

Autorun Entry	Description	Publisher	Image Path	Timestamp
HKCU\Software\Microsoft\Windows\CurrentVersion\Run				21/12/2018 3:32
OneDrive	Microsoft OneDrive	Microsoft Corporation	c:\users\user\appdata\local\microso...	07/08/1973 7:28
HKLM\Software\Microsoft\Active Setup\Installed Components				12/04/2018 1:16

Below the table, a list of services is shown:

Windows Time	Maintains d...	Manual (Trigger Start)	Local Service
Windows Update	Enables the ...	Manual (Trigger Start)	Local System
Windows Update Medic Service	Enables rem...	Manual	Local System

A context menu is open over the Windows Update service, listing options:

- Allow antimalware service to startup with normal priority
- Turn off Windows Defender Antivirus
- Configure local administrator merge behavior for lists

At the bottom, a "Performance Options" dialog box is displayed, showing the "Visual Effects" tab selected. It contains the following text and radio button options:

Select the settings you want to use for the appearance and performance of Windows on this computer.

Let Windows choose what's best for my computer
 Adjust for best appearance
 Adjust for best performance

WSL - results



38 vCPUs for 2 weeks

WSL - results

- 38 vCPUs for 2 weeks
- A working prototype

WSL - results

- 38 vCPUs for 2 weeks
- A working prototype
- 4 DoS

WSL - results

- 38 vCPUs for 2 weeks
- A working prototype
- 4 DoS

```
void main() {
    int fd = open("/proc/self/setgroups", O_RDWR);
    write(fd, 0xdeadbabe, 6)
}
```

```
void main() {
    unshare(CLONE_NEWNS);
    open("/proc/self/ns/mnt", O_RDONLY);
}
```



WSL - results

- 38 vCPUs for 2 weeks
- A working prototype
- 4 DoS
- 2 Deadlocks (still not resolved)

```
void main() {
    int fd = open("/proc/self/setgroups", O_RDWR);
    write(fd, 0xdeadbabe, 6)
}
```

```
void main() {
    unshare(CLONE_NEWNS);
    open("/proc/self/ns/mnt", O_RDONLY);
}
```



WSL - results

- 38 vCPUs for 2 weeks
- A working prototype
- 4 DoS
- 2 Deadlocks (still not resolved)
- 0 Vulnerabilities 😕

```
void main() {
    int fd = open("/proc/self/setgroups", O_RDWR);
    write(fd, 0xdeadbabe, 6)
}
```

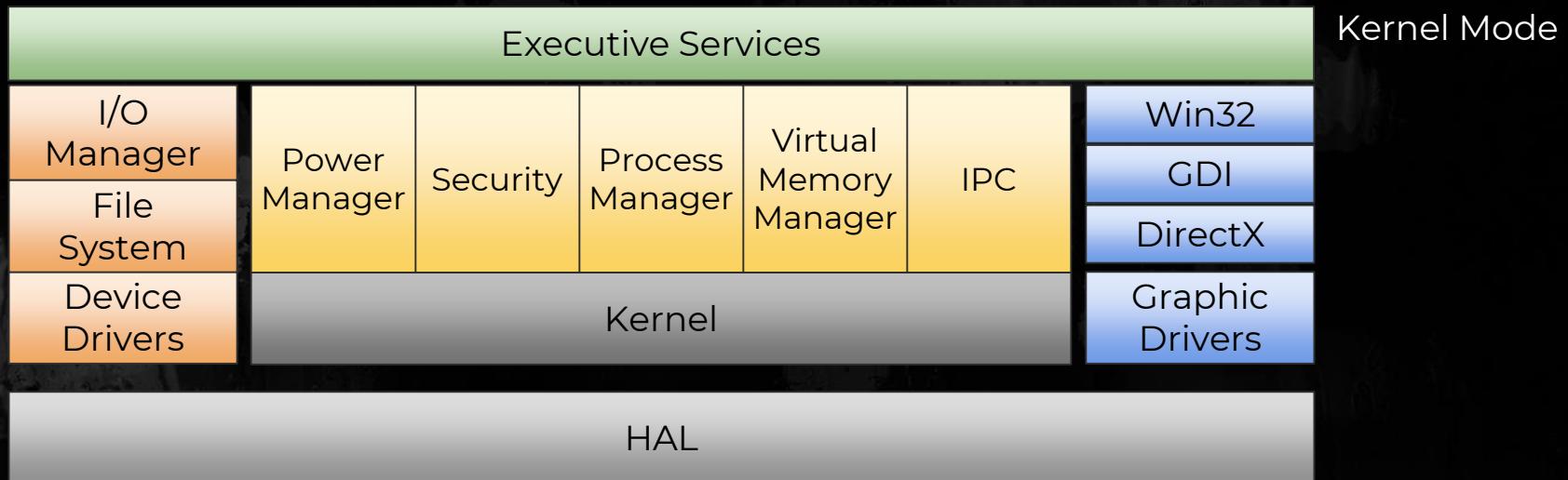
```
void main() {
    unshare(CLONE_NEWNS);
    open("/proc/self/ns/mnt", O_RDONLY);
}
```



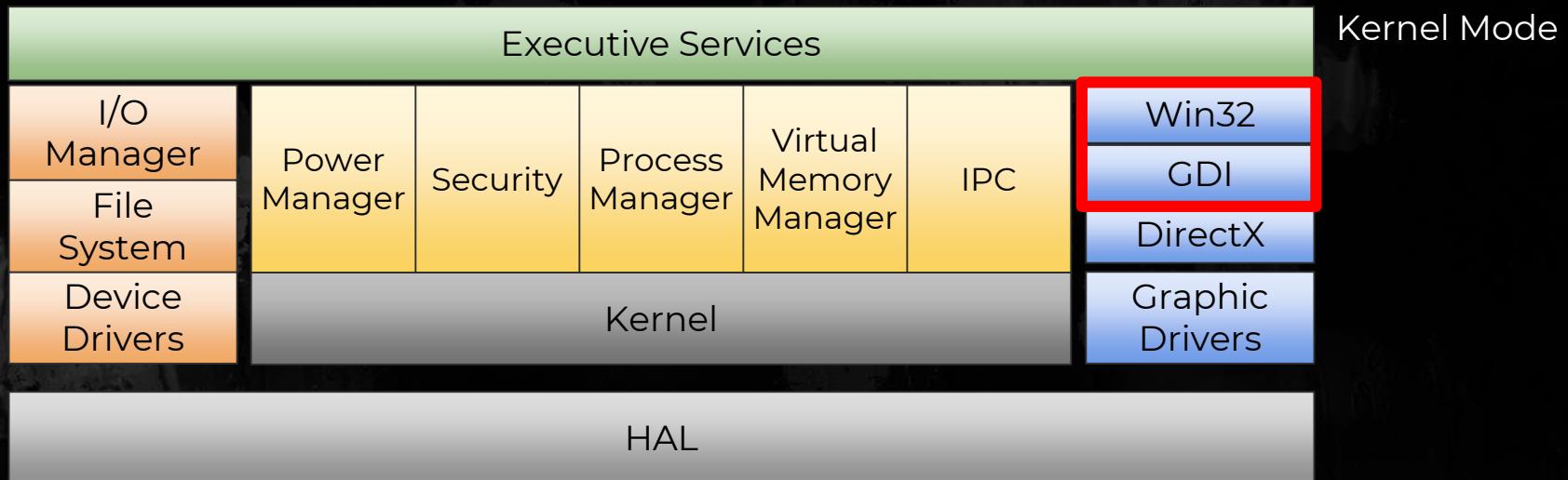


Let's move to a real target

Windows Kernel



Windows Kernel



Why Win32k?

Popular target for LPEs

Why Win32k?

Popular target for LPEs

Huge attack surface >1500 syscalls

Windows x86-64 WIN32K.SYS System Call Table (XP/2003/Vista/2008/7/2012/8/10)

Author: Mateusz "j00ru" Jurczyk (j00ru.vx tech blog)

See also: Windows System Call Tables in CSV/JSON formats on [GitHub](#)

Special thanks to: Woodmann, Deus, Gynvael Coldwind, MeMek, Alex, Omega Red, Wandering Glitch

Layout by Metasploit Team

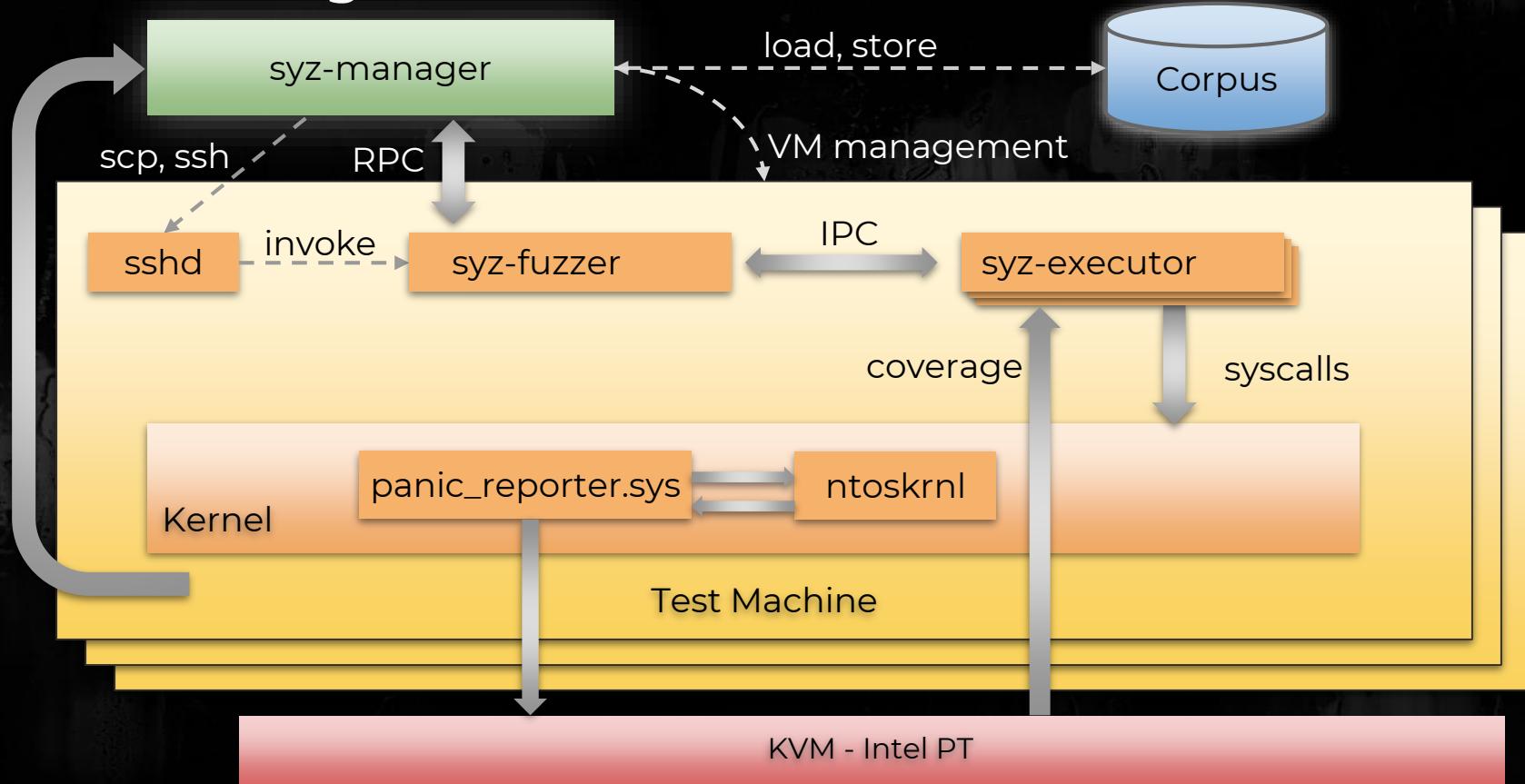
Enter the Syscall ID to highlight (hex):

Highlight

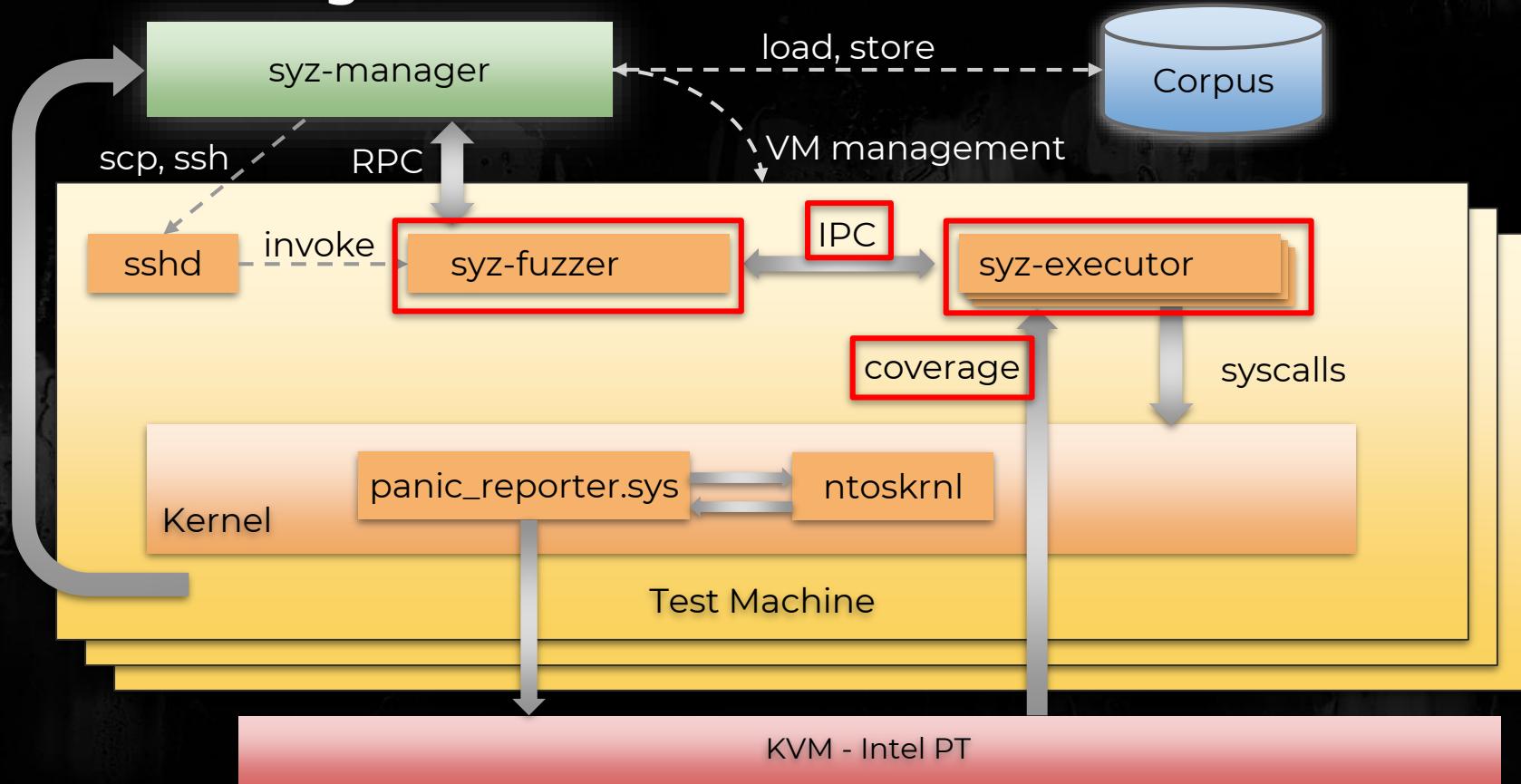
Show all Hide all

System Call Symbol	Windows XP (show)	Windows Server 2003 (show)	Windows Vista (show)	Windows Server 2008 (show)	Windows 7 (show)	Windows Server 2012 (show)	Windows 8 (show)	Windows 10 (show)
--------------------	---------------------------------------	--	--	--	--------------------------------------	--	--------------------------------------	---------------------------------------

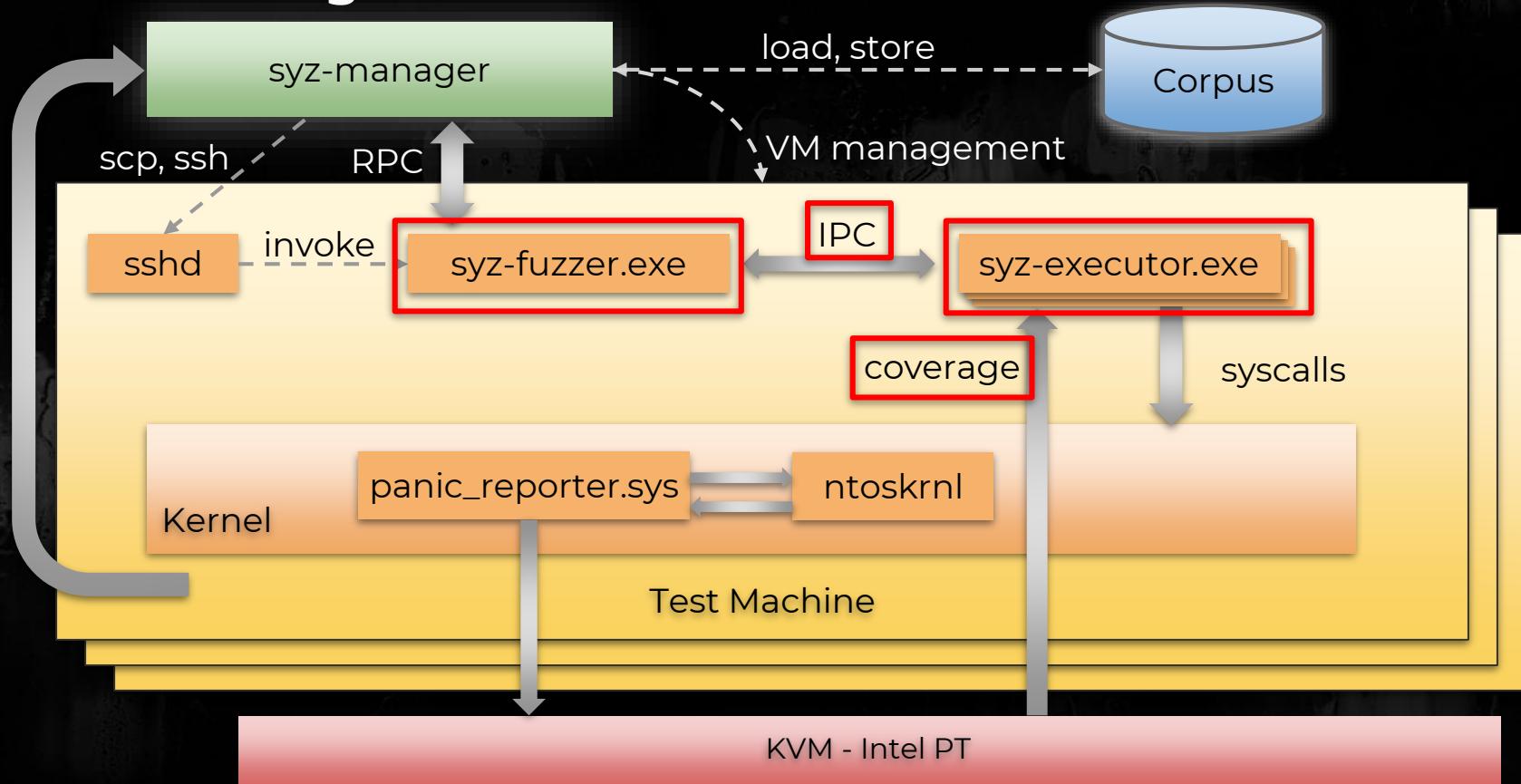
Win32k - changes



Win32k - changes



Win32k - changes



syz-executor changes

- Support for up to 12 parameters for syscalls 
- OS related changes (threads, shared memory, pipes...)
- Expose windows syscalls
- Cross compiled with mingw++

syz-executor changes

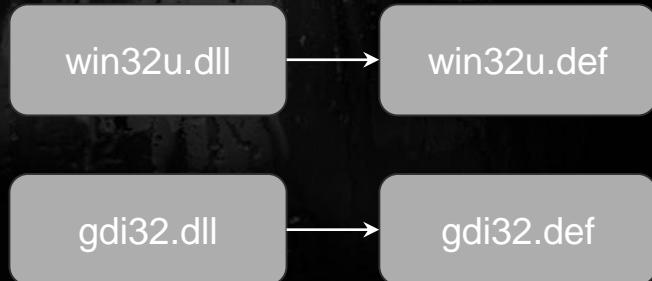
- Support for up to 12 parameters for syscalls 🤯
- OS related changes (threads, shared memory, pipes...)
- Expose windows syscalls
- Cross compiled with mingw++

win32u.dll

gdi32.dll

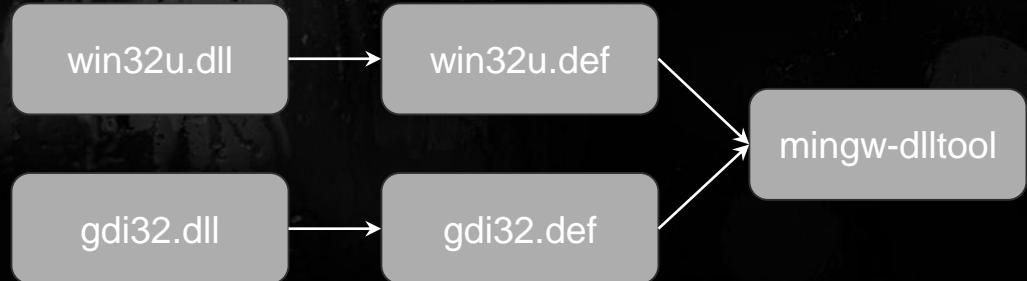
syz-executor changes

- Support for up to 12 parameters for syscalls 
- OS related changes (threads, shared memory, pipes...)
- Expose windows syscalls
- Cross compiled with mingw++



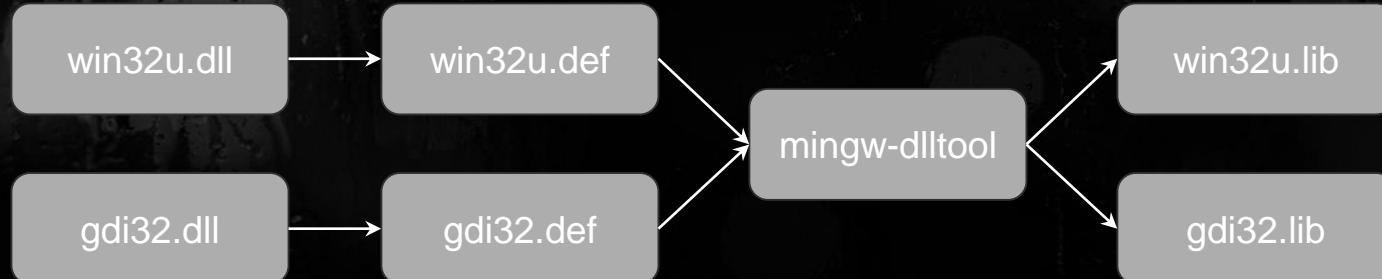
syz-executor changes

- Support for up to 12 parameters for syscalls 🤯
- OS related changes (threads, shared memory, pipes...)
- Expose windows syscalls
- Cross compiled with mingw++



syz-executor changes

- Support for up to 12 parameters for syscalls 🤯
- OS related changes (threads, shared memory, pipes...)
- Expose windows syscalls
- Cross compiled with mingw++



syz-executor changes

- Support for up to 12 parameters for syscalls 🤯
- OS related changes (threads, shared memory, pipes...)
- Expose windows syscalls
- Cross compiled with mingw++



syz-fuzzer

- OS related changes (sharing memory, handles ...)
- Win32k Grammar

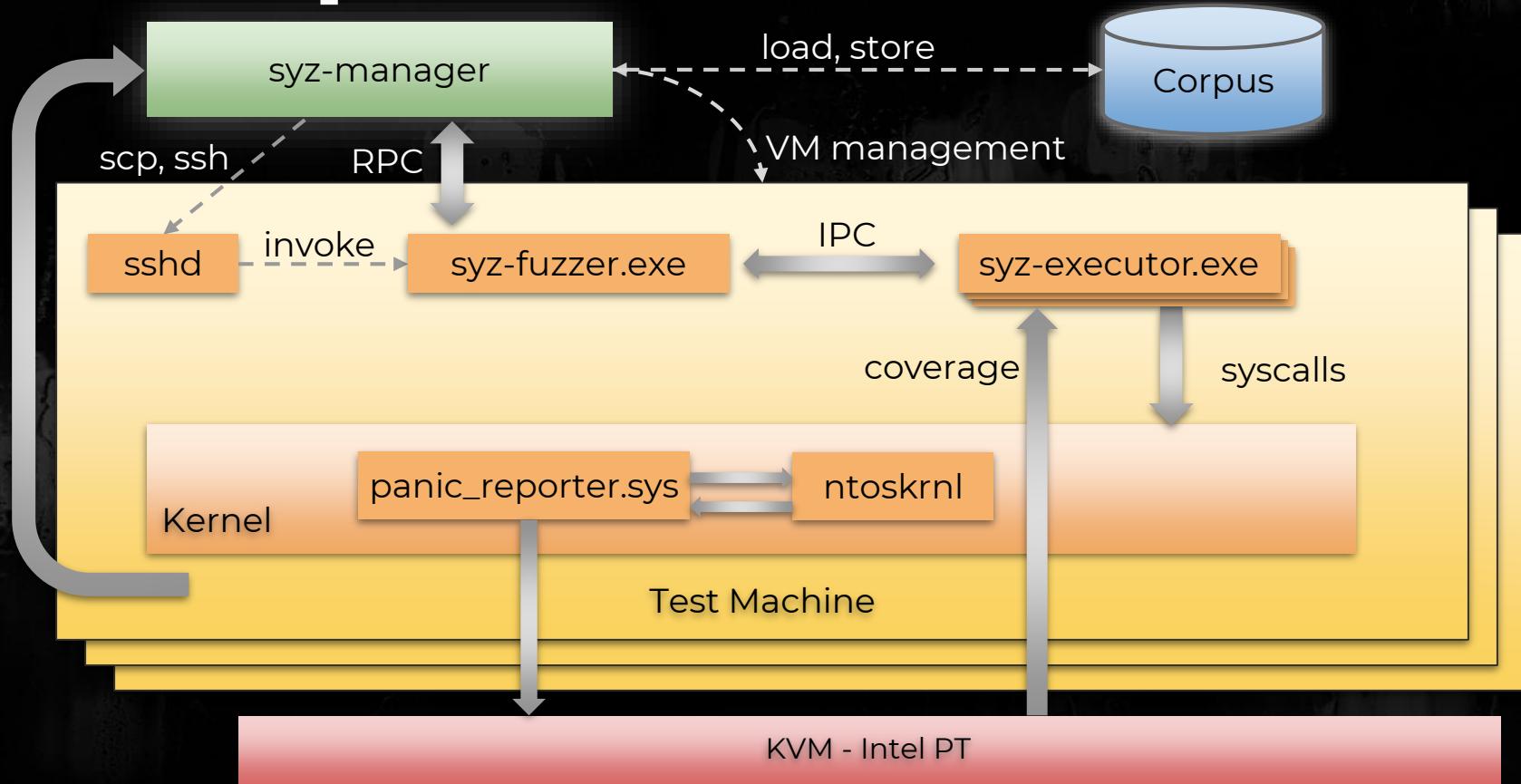
Win32k - coverage

Win32k = win32k.sys + win32kbase.sys + win32kfull.sys

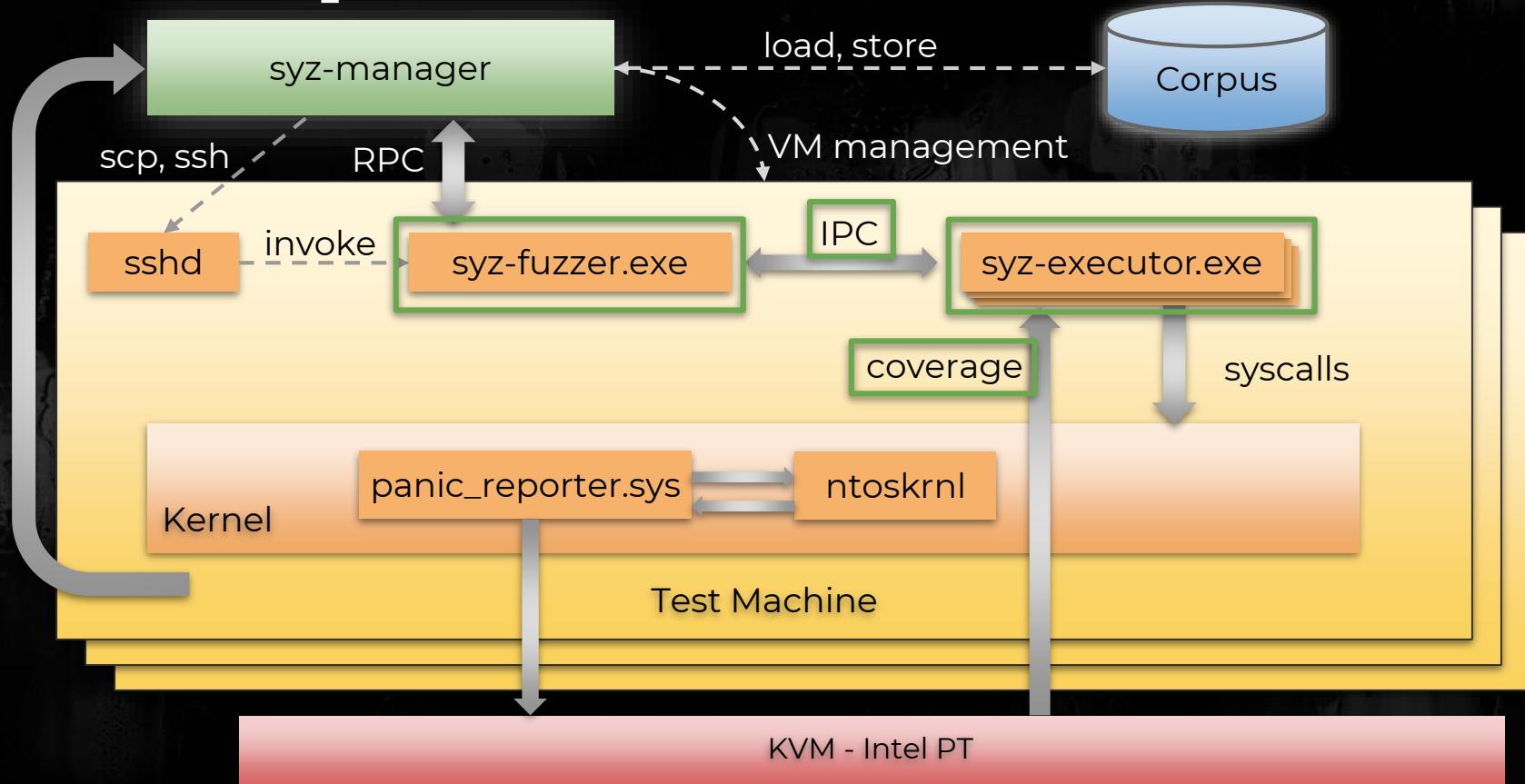
We added coverage support for multiple modules

86.88	EngStretchBltOld(_SURFOBJ *,__...)	0x1C0272B80	147 / 207	1033 / 1189	6088	129	
44.42	GreSetDIBitsToDeviceInternal	0x1C0085700	93 / 299	605 / 1362	6278	167	
24.13	BltLnkRect(_BLTLNKINFO *,_REC...	0x1C006DF00	43 / 261	324 / 1343	6335	112	
41.30	xxxMenuWindowProc	0x1C02100F0	76 / 329	650 / 1574	6552	209	
43.76	xxxInterSendMsgEx	0x1C0040BA0	122 / 385	673 / 1538	6769	275	
18.67	EngStretchBltNew(_SURFOBJ *,__...)	0x1C002D7A8	24 / 229	252 / 1350	7247	142	
57.74	GreExtTextOutWLocked(XDCOBJ &...)	0x1C007B690	147 / 369	1097 / 1900	8366	222	
43.02	xxxRealDefWindowProc	0x1C0057BDC	161 / 476	850 / 1976	8389	283	
49.78	NtGdiBitBltInternal	0x1C00818A0	153 / 429	1030 / 2069	9268	252	
42.63	NtGdiAlphaBlend	0x1C007F1A0	131 / 503	951 / 2231	10005	318	
64.16	xxxCreateWindowEx	0x1C0050290	283 / 568	1806 / 2815	13518	322	
43.37	xxxSystemParametersInfoWorker	0x1C00A1DB4	309 / 998	1649 / 3802	16767	619	
33.28	xxxScanSysQueue(tagTHREADINFO...	0x1C00470D0	165 / 771	1091 / 3278	16824	480	

Win32k - Recap



Win32k - Recap



Sanity Check

We used the fuzzer to reproduce [CVE-2018-0744](#)
(showed earlier)

Sanity Check

We used the fuzzer to reproduce CVE-2018-0744
(showed earlier)

That didn't work

Sanity Check

We used the fuzzer to reproduce CVE-2018-0744
(showed earlier)

That didn't work

Session 0 → Session 1

Sanity Check

We used the fuzzer to reproduce CVE-2018-0744
(showed earlier)

That didn't work

Session 0 → Session 1

Conclusion → Reproduce old bugs

Stability check

Added ~15 APIs and we let it run for the night

:(
BSOD

:(
BSOD



:(
BSOD

First Bug UAF

```
nt!DbgBreakPointWithStatus  
nt!KiBugCheckDebugBreak+0x12  
nt!KeBugCheck2+0x957  
nt!KeBugCheckEx+0x107  
nt!MiSystemFault+0x1ac22a  
nt!MmAccessFault+0x327  
nt!KiPageFault+0x343  
win32kfull!_OpenClipboard+0xd7439  
win32kfull!NtUserOpenClipboard+0x14a  
nt!KiSystemServiceCopyEnd+0x25  
win32u!NtUserOpenClipboard+0x14  
USER32!OpenClipboard+0x11
```

First Bug UAF

```
nt!DbgBreakPointWithStatus  
nt!KiBugCheckDebugBreak+0x12  
nt!KeBugCheck2+0x957  
nt!KeBugCheckEx+0x107  
nt!MiSystemFault+0x1ac22a  
nt!MmAccessFault+0x327  
nt!KiPageFault+0x343  
win32kfull!_OpenClipboard+0xd7439  
win32kfull!NtUserOpenClipboard+0x14a  
nt!KiSystemServiceCopyEnd+0x25  
win32u!NtUserOpenClipboard+0x14  
USER32!OpenClipboard+0x11  
  
void crash() {  
    OpenClipboard(0);  
    HWINSTA wnd1 = OpenClipboard(0, 0, WINSTA_READSCREEN | WINSTA_ENUMDESKTOPS, 0);  
    HWINSTA wnd2 = GetDesktopWindow();  
    OpenClipboard(wnd2);  
    SetProcessWindowStation(wnd1);  
}
```

Reproduces on some machines

What the **fuzz**?

First Bug UAF

```
}

UserSetLastError(5i64);
// ;
// ; Check if ETW 487d6e37-1b9d-46d3-a8fd-54ce8bdf8a53 has specific flag
// ;
if ( (unsigned int)Win32kTraceLoggingLevel > 5
    && TlgKeywordOn((__int64)&Win32kTraceLoggingLevel, 0x400000000000i64) )
{
    thread_info = *(_QWORD *) (winsta_tag + 48);
    // ;
    // ; // pti = tagTHREADINFO
    // ; // ppi = PPROCESSINFO
    // ;
    // ; pti->ppi->W32Pid
    // ;
    v14 = *(_DWORD *) (*(_QWORD *) (thread_info + 416) + 56i64);
    v17 = &v14;
    v18 = 4i64;
    TlgCreateWsz(&v19, (unsigned __int16 *) (*(_QWORD *) (thread_info + 416) + 960i64));
    // ;
    // ; call EtwWriteTransfer
    // ;
    TlgWrite((__int64)&Win32kTraceLoggingLevel, (unsigned __int8 *) &unk_1C02D87BC, 0i64, 0i64, v13, (__int64)&v16);
}
```

First Bug UAF

```
}

UserSetLastError(5i64);
// ;
// ; Check if ETW 487d6e37-1b9d-46d3-a8fd-54ce8bdf8a53 has specific flag
// ;
if ( (unsigned int)Win32kTraceLoggingLevel > 5
    && TlgKeywordOn((__int64)&Win32kTraceLoggingLevel, 0x400000000000i64) )
{
    thread_info = *(_QWORD *) (winsta_tag + 48);
    // ;
    // ; // pti = tagTHREADINFO
    // ; // ppi = PPROCESSINFO
    // ;
    // ; pti->ppi->W32Pid
    // ;
    v14 = *(_DWORD *) (*(_QWORD *) (thread_info + 416) + 56i64);
    v17 = &v14;
    v18 = 4i64;
    TlgCreateWsz(&v19, (unsigned __int16 *) (*(_QWORD *) (thread_info + 416) + 960i64));
    // ;
    // ; call EtwWriteTransfer
    // ;
    TlgWrite((__int64)&Win32kTraceLoggingLevel, (unsigned __int8 *) &unk_1C02D87BC, 0i64, 0i64, v13, (__int64)&v16);
}
```

First Bug UAF

```
}

UserSetLastError(5i64);
// ;
// ; Check if ETW 487d6e37-1b9d-46d3-a8fd-54ce8bdf8a53 has specific flag
// ;
if ( (unsigned int)Win32kTraceLoggingLevel > 5
    && TlgKeywordOn((__int64)&Win32kTraceLoggingLevel, 0x400000000000i64) )
{
    thread_info = *(_QWORD *) (winsta_tag + 48);
    // ;
    // ; // pti = tagTHREADINFO
    // ; // ppi = PPROCESSINFO
    // ;
    // ; pti->ppi->W32Pid
    // ;
    v14 = *(_DWORD *) (*(_QWORD *) (thread_info + 416) + 56i64);
    v17 = &v14;
    v18 = 4i64;
    TlgCreateWsz(&v19, (unsigned __int16 *) (*(_QWORD *) (thread_info + 416) + 960i64));
    // ;
    // ; call EtwWriteTransfer
    // ;
    TlgWrite((__int64)&Win32kTraceLoggingLevel, (unsigned __int8 *) &unk_1C02D87BC, 0i64, 0i64, v13, (__int64)&v16);
}
```

First Bug UAF

```
}

UserSetLastError(5i64);
// ;
// ; Check if ETW 487d6e37-1b9d-46d3-a8fd-54ce8bdf8a53 has specific flag
// ;
if ( (unsigned int)Win32kTraceLoggingLevel > 5
    && TlgKeywordOn((__int64)&Win32kTraceLoggingLevel, 0x400000000000i64) )
{
    thread_info = *(_QWORD *) (winsta_tag + 48);
    // ;
    // ; // pti = tagTHREADINFO
    // ; // ppi = PPROCESSINFO
    // ;
    // ; pti->ppi->W32Pid
    // ;
    v14 = *(_DWORD *) (*(_QWORD *) (thread_info + 416) + 56i64);
    v17 = &v14;
    v18 = 4i64;
    TlgCreateWsz(&v19, (unsigned __int16 *) (*(_QWORD *) (thread_info + 416) + 960i64));
    // ;
    // ; call EtwWriteTransfer
    // ;
    TlgWrite((__int64)&Win32kTraceLoggingLevel, (unsigned __int8 *)unk_1C02D87BC, 0i64, 0i64, v13, (__int64)&v16);
}
```

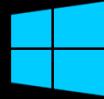
I DONT ALWAYS GET
AN A/B TESTED MACHINE



BUT WHEN I DO, I
FIND A VULNERABILITY IN IT

Back to checking stability

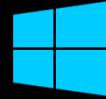
We re-installed windows, let it run again for the night



Back to checking stability

We re-installed windows, let it run again for the night

We got another bug



2nd Bug

```
nt!RtlpHpVsChunkSplit+0x3c5
nt!RtlpHpVsContextAllocateInternal+0x28c
nt!RtlpHpVsContextAllocate+0x46
nt!RtlpAllocateHeapInternal+0x55
nt!RtlAllocateHeap+0x37
win32kfull!ClassAlloc+0x39
win32kfull!InternalRegisterClassEx+0x14e
win32kfull!_RegisterClassEx+0x85
win32kfull!NtUserRegisterClassExWOW+0x59f
nt!KiSystemServiceCopyEnd+0x25
win32u!NtUserRegisterClassExWOW+0x14
USER32!RegisterClassExWOWA+0x205
USER32!RegisterClassExA+0x30
poc!main+0x67 [r:\poc\poc\main.cpp @ 15]
```

DoS in RegisterClassExA

```
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = { 0 };
    WindowClass.cbSize = sizeof(WNDCLASSEX);
    WindowClass.cbClsExtra = 0x2771;
    WindowClass.lpfnWndProc = DefWindowProc;
    WindowClass.lpszClassName = "Class";
    RegisterClassExA(&WindowClass);
    return 0;
}
```

Motivation++

15 syscalls → 2 Bugs

~1500 syscalls → ~200 Bugs



Win32k - grammar

Create syscall grammar from scratch

Automation?

Win32k - grammar

Create syscall grammar from scratch

Automation?

```
HWND  
WINAPI  
CreateWindowExA(  
    _In_     DWORD dwExStyle,  
    _In_opt_  LPCSTR lpClassName,  
    _In_opt_  LPCSTR lpWindowName,  
    _In_     DWORD dwStyle,  
    _In_     int   X,  
    _In_     int   Y,  
    _In_     int   nWidth,  
    _In_     int   nHeight,  
    _In_opt_  HWND  hWndParent,  
    _In_opt_  HMENU hMenu,  
    _In_opt_  HINSTANCE hInstance,  
    _In_opt_  LPVOID lpParam);
```

Win32k - grammar

Create syscall grammar from scratch

Automation?

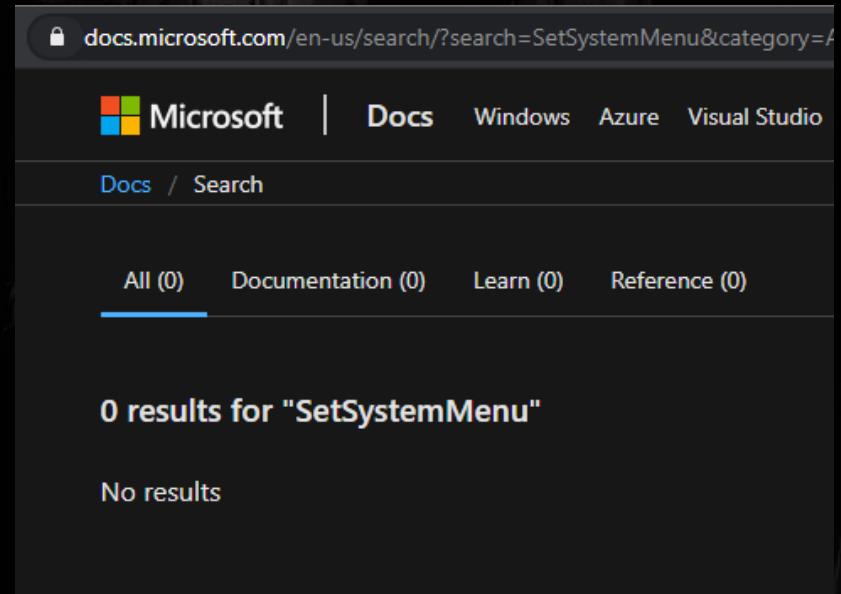
```
HWND  
WINAPI  
CreateWindowExA(  
    _In_ DWORD dwExStyle,  
    _In_opt_ LPCSTR lpClassName,  
    _In_opt_ LPCSTR lpWindowName,  
    _In_ DWORD dwStyle,  
    _In_ int X,  
    _In_ int Y,  
    _In_ int nWidth,  
    _In_ int nHeight,  
    _In_opt_ HWND hWndParent,  
    _In_opt_ HMENU hMenu,  
    _In_opt_ HINSTANCE hInstance,  
    _In_opt_ LPVOID lpParam);
```

Win32k - grammar

Create syscall grammar from scratch

Automation?

```
HWND  
WINAPI  
CreateWindowExA(  
    _In_ DWORD dwExStyle,  
    _In_opt_ LPCSTR lpClassName,  
    _In_opt_ LPCSTR lpWindowName,  
    _In_ DWORD dwStyle,  
    _In_ int X,  
    _In_ int Y,  
    _In_ int nWidth,  
    _In_ int nHeight,  
    _In_opt_ HWND hWndParent,  
    _In_opt_ HMENU hMenu,  
    _In_opt_ HINSTANCE hInstance,  
    _In_opt_ LPVOID lpParam);
```



The screenshot shows a Microsoft Docs search results page. The URL in the address bar is docs.microsoft.com/en-us/search/?search=SetSystemMenu&category=/. The page features the Microsoft logo and navigation links for Docs, Windows, Azure, and Visual Studio. Below the navigation is a search bar with the text "Docs / Search". Underneath are four buttons: All (0), Documentation (0), Learn (0), and Reference (0). The "All (0)" button is highlighted with a blue underline. The main content area displays the message "0 results for 'SetSystemMenu'" and "No results".

Win32k - grammar

- Technically windows is open source
 - Windows NT Leaked sources - <https://github.com/ZoloZiak/WinNT4>
 - Windows 2000 Leaked sources - <https://github.com/pustladi/Windows-2000>
 - ReactOS (Leaked w2k3 sources?) - <https://github.com/reactos/reactos>
 - Windows Research Kit - <https://github.com/Zer0Mem0ry/ntoskrnl>
- For each syscall
 - We looked at the sources + MSDN
 - Verified definition with IDA / WinDbg
- Most APIs are pretty simple
- But others are a nightmare



```
SetBkColor(hDc HDC, color int32)
EndPage(hDc HDC)
EndPath(hDc HDC)
CloseFigure(hDc HDC)
BeginPath(hDc HDC)
SetMetaRgn(hDc HDC)
StrokePath(hDc HDC)
GetWindowTextA(hWnd HWND, lpString buffer[out], nMaxCount len[lpString])
SaveDC(hDc HDC)
StrokeAndFillPath(hDc HDC)
StartPage(hDc HDC)
CancelDC(hDc HDC)
GetSystemPaletteUse(hDc HDC)
ChildWindowFromPointEx(hWnd HWND, pt ptr[in, POINT], flags int32[0:4]) HWND
WidenPath(hDc HDC)
SwapBuffers(hDc HDC)
AbortDoc(hDc HDC)
FlattenPath(hDc HDC)
UpdateColors(hDc HDC)
CreateHalftonePalette(hDc HDC) HPALETTE
SelectPalette(hDc HDC, hPal HPALETTE, bForceBkgd int32[0:1]) HPALETTE
FillPath(hDc HDC)
EndDoc(hDc HDC)
PathToRegion(hDc HDC) HRGN
AbortPath(hDc HDC)
SetLayout(hDc HDC, l int32[0x0:0x10])
SelectClipPath(hDc HDC, mode int32[0x0:0x10])
PtVisible(hDc HDC, x int32, y int32)
LineTo(hDc HDC, x int32, y int32)
GetNearestColor(hDc HDC, cr int32)
RestoreDC(hDc HDC)
WindowFromPoint(point ptr[in, POINT]) HWND
DrawCaption(hWnd HWND, hDc HDC, point ptr[in, POINT], uflags flags[dc_flags]) HWND
DrawAnimatedRects(hWnd HWND, idAni int32[0:3], lprcFrom ptr[in, RECT], lprcTo ptr[in, R
RegisterHotKey(hWnd HWND[opt], id int32[0:0xff], fsModifiers flags[mod_flags], vk int32
UnregisterHotKey(hWnd HWND[opt], id int32[0x0:0xff])
ShowWindow(hWnd HWND, nCmdShow int32[0x0:0x10])
ShowWindowAsync(hWnd HWND, nCmdShow int32[0x0:0x10])
UpdateLayeredWindow(hWnd HWND, hdcDst HDC[opt], pptDst ptr[in, POINT, opt], ppSize ptr[
FlashWindow(hWnd HWND, bInvert int32[0:1])
FlashWindowEx(pFwi ptr[in, FLASHINFO])
```

```
91 ▼ GUITHREADINFO {
92     cbSize      const[72, int32]
93     flags       int32[0:16]
94     hwndActive  HWND
95     hwndFocus   HWND
96     hwndCapture HWND
97     hwndMenuOwner  HWND
98     hwndMoveSize  HWND
99     hwndCaret   HWND
00     rcRect      RECT
01 }
02
03 ▼ MNDDRAG {
04     a          int32
05     hMenu     HMENU
06     b          int32
07     hWnd       HWND
08 }
09
10 ▼ TRIVERTEX {
11     x          int32
12     y          int32
13     Red        int16
14     Green      int16
15     Blue       int16
16     Alpha      int16
17 }
```

3 new vulnerabilities in GDI

3 new vulnerabilities in GDI

But we want more!

Going deeper

Fuzzers are not **magic**

- We need to **teach** them tricks & help them reach difficult attack surfaces

Our process

- We learn as much we can about the attack surfaces (bug classes, prior work, ..)
- We try to **reproduce** old bugs
- Take time to look at coverage results

We try to **inject insights** back into the fuzzing process

Win32k “trick” example

Gdi Shared Handle Table, pointed from the PEB

Win32k “trick” example

Gdi Shared Handle Table, pointed from the PEB

```
typedef struct {
    PVOID64 pKernelAddress;
    USHORT wProcessId;
    USHORT wCount;
    USHORT wUpper;
    USHORT wType;
    PVOID64 pUserAddress;
} GDICELL;
```

Win32k “trick” example

Gdi Shared Handle Table, pointed from the PEB

Includes **global** handles (created by win32k)

Let's use them - “GetGdiHandle(type, index)”

```
typedef struct {
    PVOID64 pKernelAddress;
    USHORT wProcessId;
    USHORT wCount;
    USHORT wUpper;
    USHORT wType;
    PVOID64 pUserAddress;
} GDICELL;
```

Win32k “trick” example

Gdi Shared Handle Table, pointed from the PEB

Includes **global** handles (created by win32k)

Let's use them - “GetGdiHandle(type, index)”

```
typedef struct {
    PVOID64 pKernelAddress;
    USHORT wProcessId;
    USHORT wCount;
    USHORT wUpper;
    USHORT wType;
    PVOID64 pUserAddress;
} GDICELL;
```

Using this trick **CVE-2019-1159 UAF** triggered by one syscall



```
void crash() {
    GetDCEx(0, hGlobalHrgn, DCX_EXCLUDERGN);
}
```

Results - Win32k



60 vCPUs & 1.5 months of fuzzing part time

Results - Win32k



60 vCPUs & 1.5 months of fuzzing part time



8 vulns, 6 CVEs (1 duplicate, 1 pending)

Results - Win32k



60 vCPUs & 1.5 months of fuzzing part time



8 vulns, 6 CVEs (1 duplicate, 1 pending)

- CVE-2019-1014, CVE-2019-1096, CVE-2019-1159,
CVE-2019-1164, CVE-2019-1256, CVE-2019-1286

Results - Win32k



60 vCPUs & 1.5 months of fuzzing part time

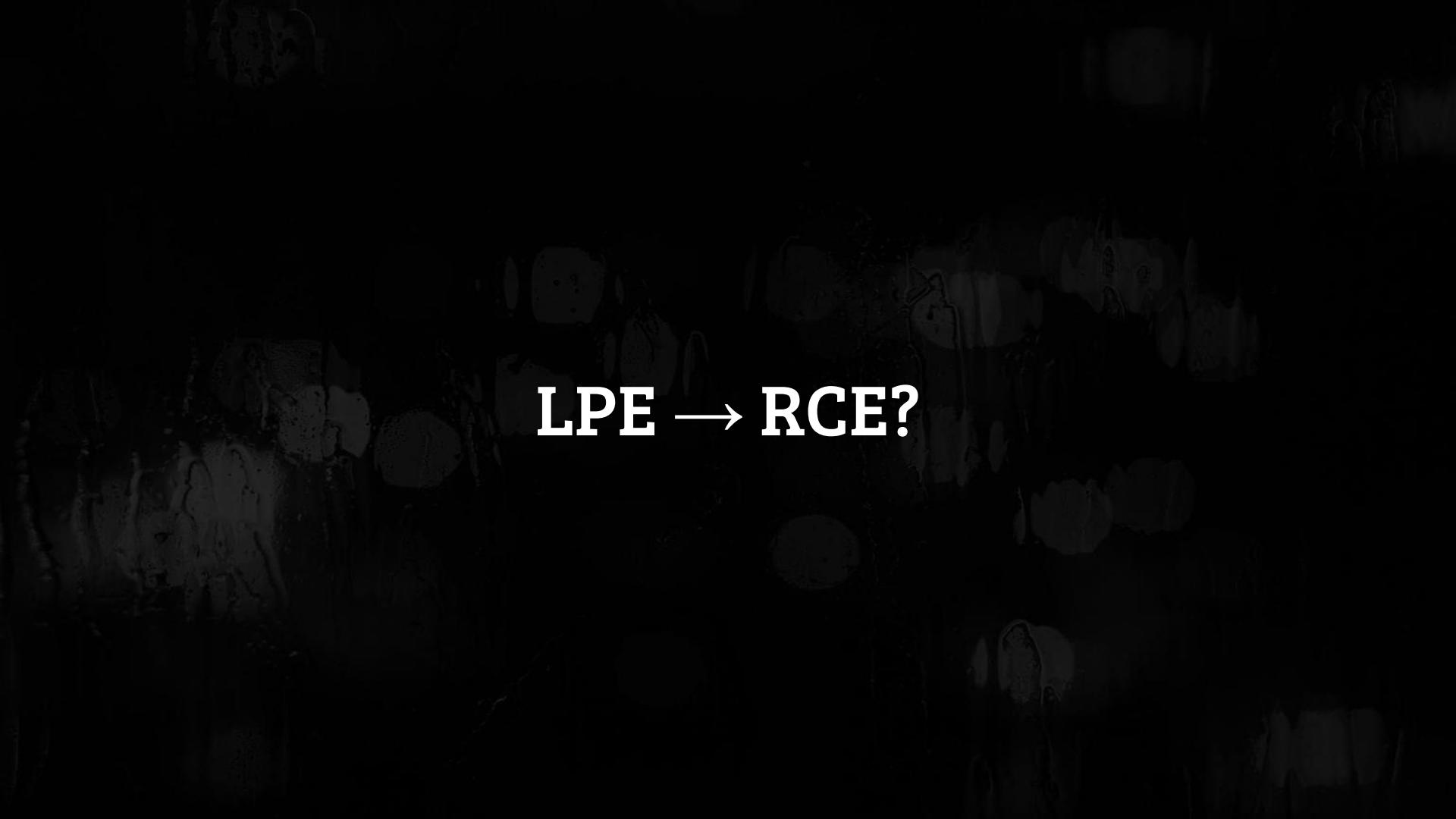


8 vulns, 6 CVEs (1 duplicate, 1 pending)

- CVE-2019-1014, CVE-2019-1096, CVE-2019-1159,
CVE-2019-1164, CVE-2019-1256, CVE-2019-1286



3 DOS, 1 crash in WinLogon and few deadlocks



LPE → RCE?

WMF - Windows Metafile Format

Designed back in 1990s

Supports both **vector graphics** and **bitmaps**

The image is basically a lists of records describing GDI calls

Microsoft Extended WMF:

- EMF
- EMF+
- EMFSPOOL

<https://j00ru.vexillium.org/slides/2016/pacsec.pdf>

EMF - Enhanced Metafile Format

2.3	EMF Records.....	73
2.3.1	Bitmap Record Types	74
2.3.1.1	EMR_ALPHABLEND Record	76
2.3.1.2	EMR_BITBLT Record	80
2.3.1.3	EMR_MASKBLT Record.....	83
2.3.1.4	EMR_PLGBLT Record	87
2.3.1.5	EMR_SETDIBITSTODEVICE Record	90
2.3.1.6	EMR_STRETCHBLT Record.....	93
2.3.1.7	EMR_STRETCHDIBITS Record	95
2.3.1.8	EMR_TRANSPARENTBLT Record	98
2.3.2	Clipping Record Types	101

EMF - Enhanced Metafile Format

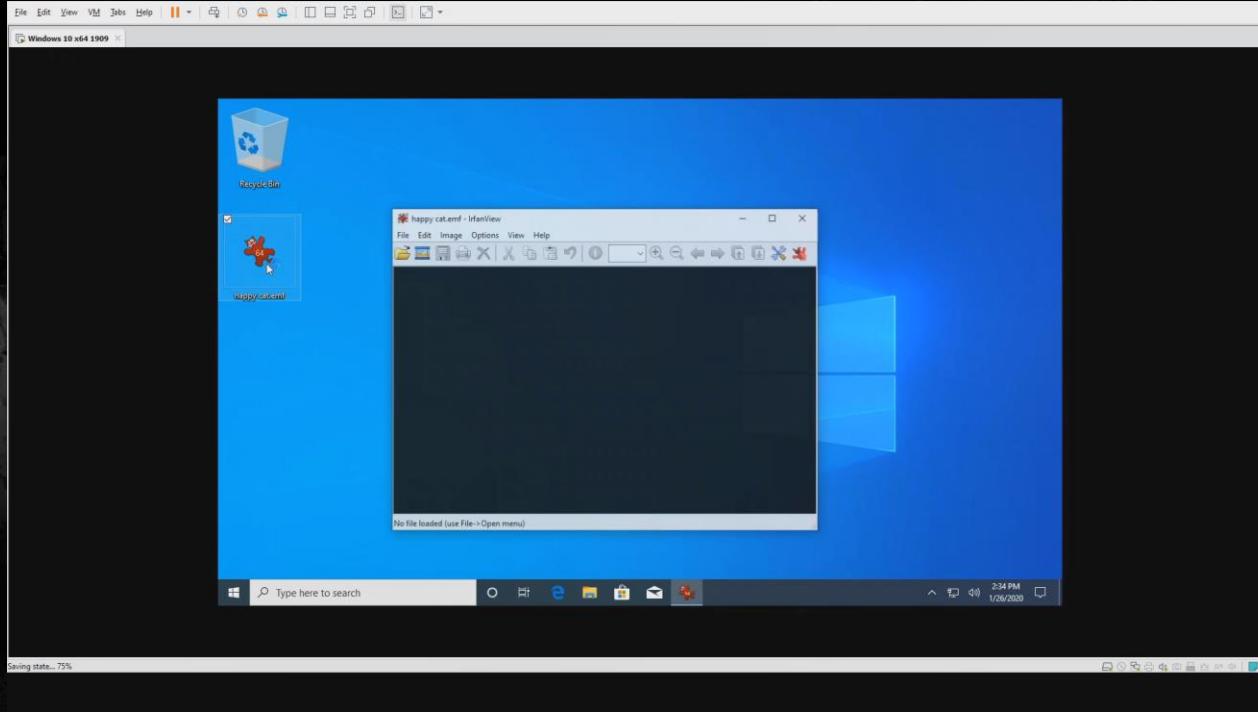
2.3	EMF Records.....	73
2.3.1	Bitmap Record Types	74
2.3.1.1	EMR_ALPHABLEND Record	76
2.3.1.2	EMR_BITBLT Record	80
2.3.1.3	EMR_MASKBLT Record.....	83
2.3.1.4	EMR_PLGBLT Record	87
2.3.1.5	EMR_SETDIBITSTODEVICE Record	90
2.3.1.6	EMR_STRETCHBLT Record.....	93
2.3.1.7	EMR_STRETCHDIBITS Record	95
2.3.1.8	EMR_TRANSPARENTBLT Record	98
2.3.2	Clipping Record Types	101

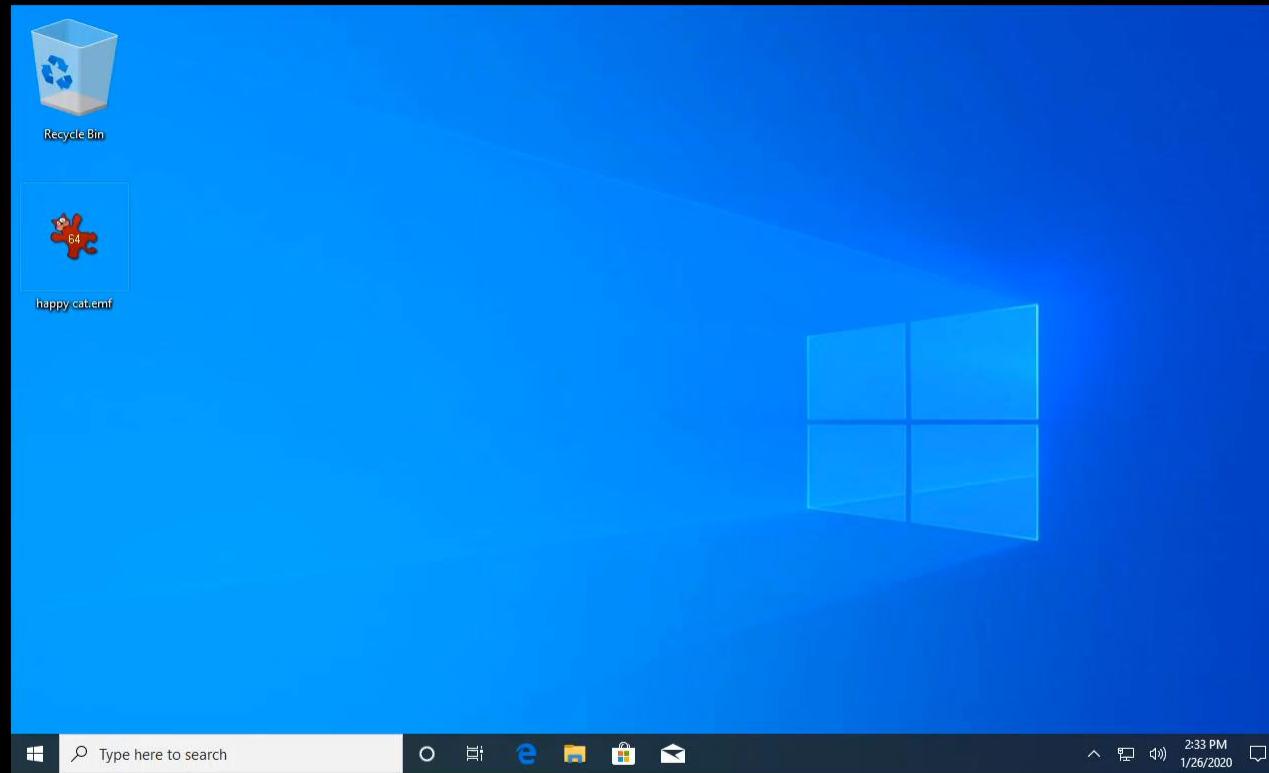
EMF - Enhanced Metafile Format

2.3	EMF Records.....	73
2.3.1	Bitmap Record Types	74
2.3.1.1	EMR_ALPHABLEND Record	76
2.3.1.2	EMR_BITBLT Record	80
2.3.1.3	EMR_MASKBLT Record.....	83
2.3.1.4	EMR_PLGBLT Record	87
2.3.1.5	EMR_SETDIBITSTODEVICE Record	90
2.3.1.6	EMR_STRETCHBLT Record.....	93
2.3.1.7	EMR_STRETCHDIBITS Record	95
2.3.1.8	EMR_TRANSPARENTBLT Record	98
2.3.2	Clipping Record Types	101

We happen to have a
vulnerability in **StretchBlt**

GDI -> EMF -> RCE: DEMO





Future work

- Win32k
 - DirectX drivers
 - Win32 callbacks (callbacks in general)
- Generated corpus (450k) + New bug oracles (kcsan / bochspwn)
- Open source
- The rest of the kernel

Summary

50 CVEs

Summary

50 CVEs

kAFL

Wanted Kernel

Summary

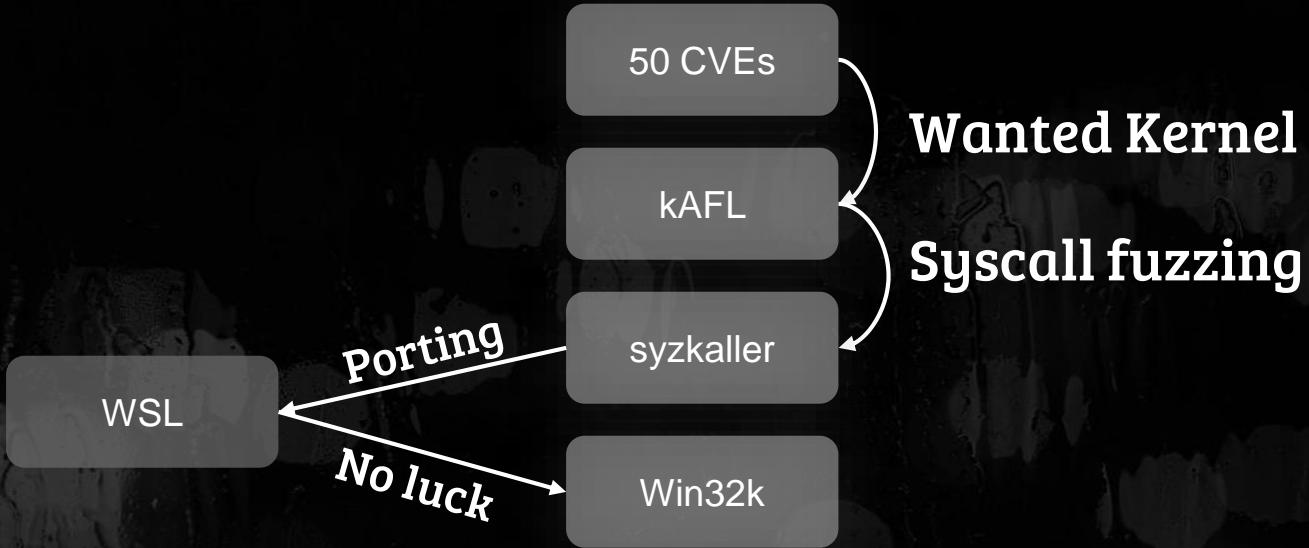


Wanted Kernel
Syscall fuzzing

Summary



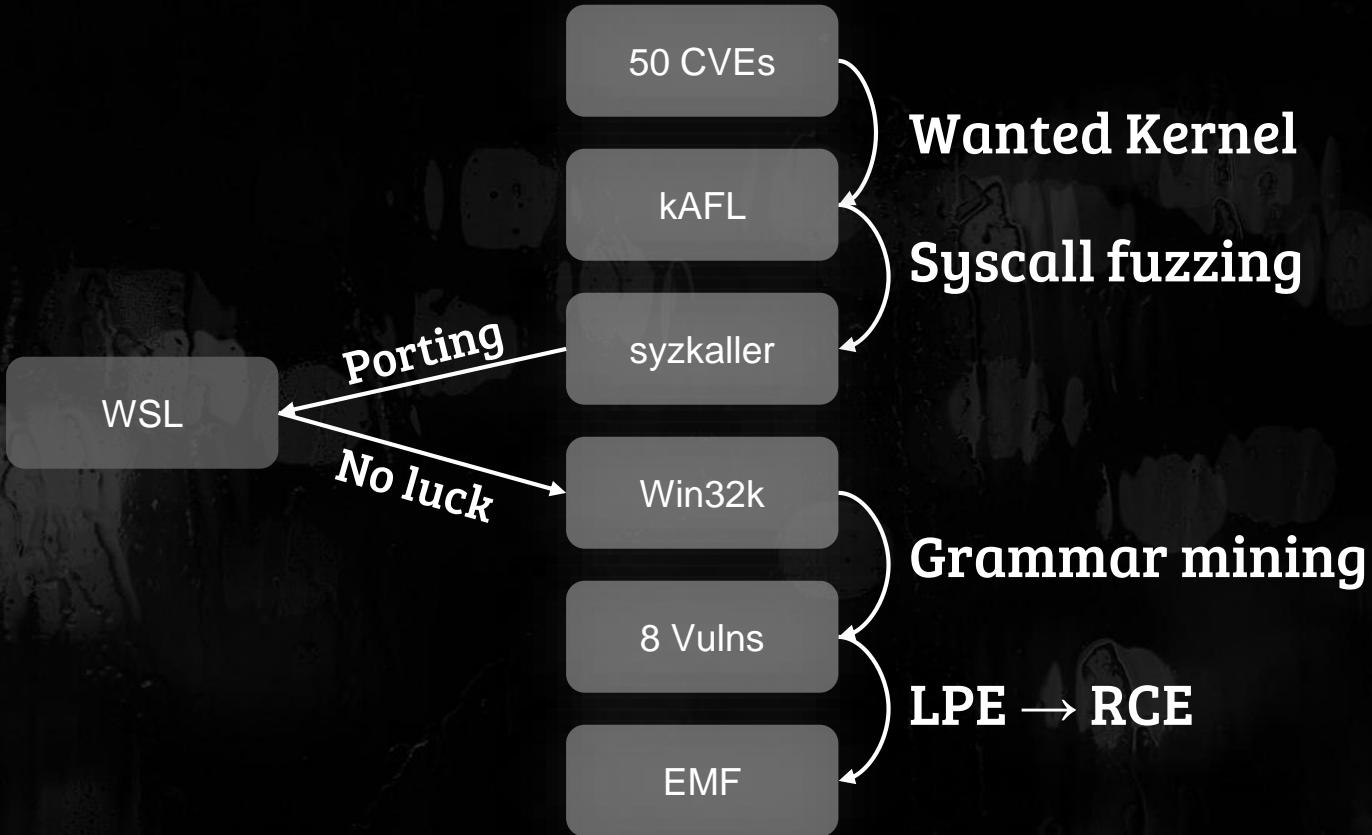
Summary



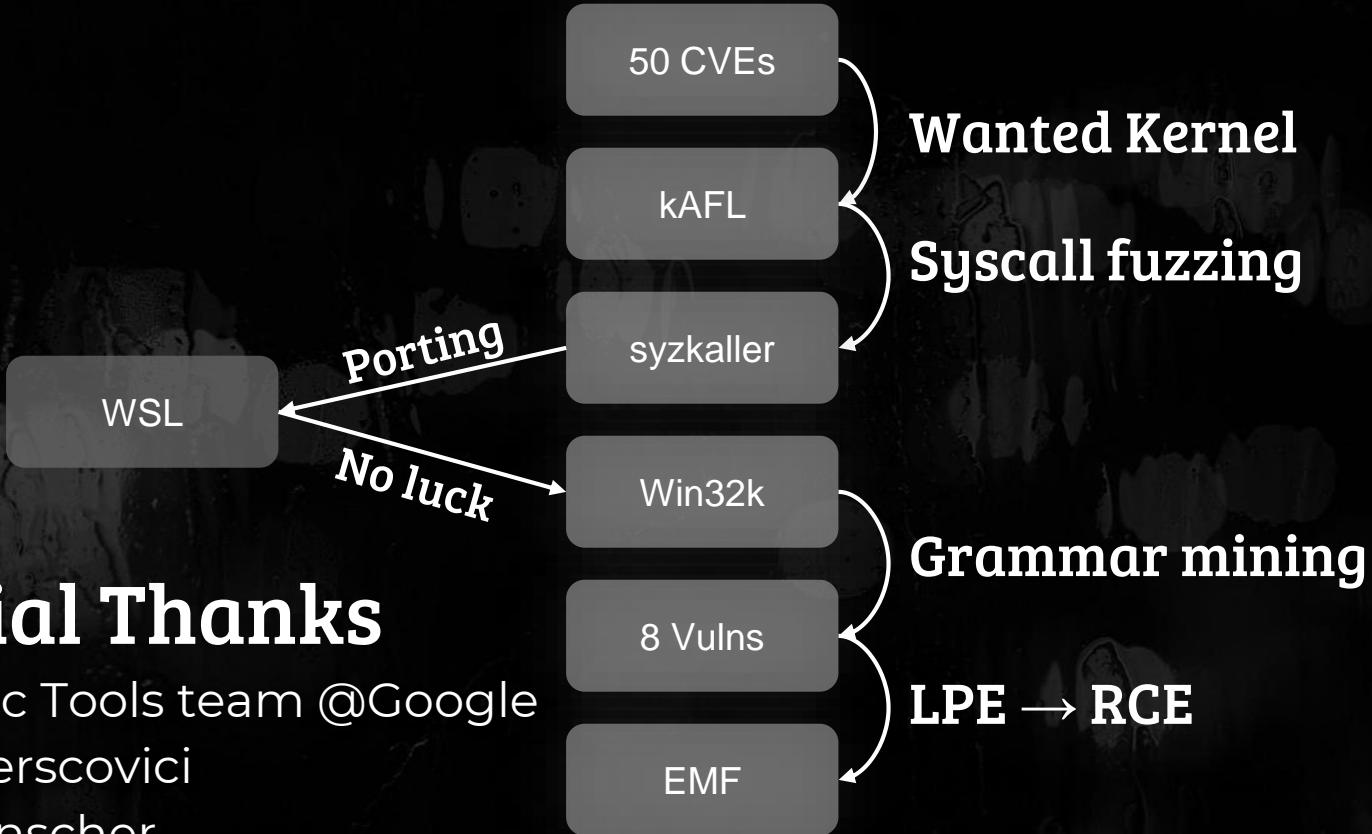
Summary



Summary



Summary





Questions?



@YoavAlon



@NetanelBenSimon