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


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:
Obligatory slides - What is fuzzing?

A method for automatic software testing

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

Example of a basic fuzzer:

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

https://www.youtube.com/watch?v=4BkAxMfHSzI - @MShudrak
Structure of modern fuzzers

Modern fuzzers consist of

- Test case generator/mutator
- Bug oracle
- Feedback mechanism - usually code coverage

Examples: AFL, libfuzzer, honggfuzz...
Structure of modern fuzzers

Modern fuzzers consist of

- Test case generator/mutator
- Bug oracle
- Feedback mechanism - usually code coverage

Examples: AFL, libfuzzer, honggfuzz...
Structure of modern fuzzers

Modern fuzzers consist of

- Test case generator/mutator
- Bug oracle
- Feedback mechanism - usually code coverage

Examples: AFL, libfuzzer, honggfuzz...
Coverage Guided Fuzzing process

Initial corpus
Coverage Guided Fuzzing process

Initial corpus

queue
Coverage Guided Fuzzing process

Initial corpus → queue → generate / mutate
Coverage Guided Fuzzing process

Initial corpus

queue

generate / mutate

run target
Coverage Guided Fuzzing process

Initial corpus

queue

generate / mutate

run target

crash
Coverage Guided Fuzzing process

Initial corpus

queue → generate / mutate → run target → crash

triage & save
Coverage Guided Fuzzing process

- Initial corpus
  - queue
  - generate / mutate
  - run target
  - crash
    - triage & save
  - new coverage
Coverage Guided Fuzzing process

Initial corpus -> queue -> generate / mutate -> run target -> crash
triage & save -> minimize & save -> new coverage
Coverage Guided Fuzzing process

Initial corpus

queue

generate / mutate

run target

Discard

crash

triage & save

minimize & save

new coverage
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
kAFL Architecture

- Fuzzer
- QEMU
- KVM
- User
- Kernel
kAFL
Architecture

Fuzzer

QEMU

agent.exe

KVM

User

Kernel
kAFL Architecture

- Fuzzer
- Input
- agent.exe
- QEMU
- User
- Kernel
- KVM
**kAFL Architecture**

- **Fuzzer**
- **agent.exe**
- **test.sys**
- **QEMU**
- **User**
- **Kernel**
- **KVM**

Stop trace
kAFL Architecture

Fuzzer -> Input -> agent.exe

User -> Start trace -> Test input -> Stop trace

Kernel -> QEMU

QEMU -> test.sys

User -> Collect trace

Kernel

KVM
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?

Fuzzer

agent.exe

QEMU

User

Kernel

KVM
How to detect crashes?

The diagram illustrates a process involving several components:

- **Fuzzer**
- **agent.exe**
- **QEMU**
- **Hypercall**
- **KVM**
- **Kernel**
- **User**

The process involves interaction between these components, with `agent.exe` acting as a central agent that communicates via `Hypercall` with `QEMU`, which in turn interfaces with the **Kernel** and **User** environments through **KVM**.
kAFL

How to detect crashes?

User

Kernel

agent.exe

QEMU

ntoskrnl

Fuzzer

KVM
kAFL
How to detect crashes?

mov rax, rcx
mov rbx, rdx
mov rcx, r8
mov rdx, r9
vmcall
kAFL
How to detect crashes?
How to detect crashes?
**kAFL Dashboard**

<table>
<thead>
<tr>
<th>Runtime</th>
<th>000:00:26:23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Path</td>
<td>000:00:07:33</td>
</tr>
<tr>
<td>Bitmap</td>
<td>01.0b/ 00.0%</td>
</tr>
<tr>
<td>Blacklisted</td>
<td>0/ 0</td>
</tr>
<tr>
<td>Cycles</td>
<td>22</td>
</tr>
<tr>
<td>Level</td>
<td>7/ 8</td>
</tr>
<tr>
<td>Favs</td>
<td>8/ 11 (72.7%)</td>
</tr>
<tr>
<td>Pending</td>
<td>0/ 0</td>
</tr>
<tr>
<td>Skipped</td>
<td>0/ 0</td>
</tr>
<tr>
<td>Payload-Size</td>
<td>34B</td>
</tr>
<tr>
<td>Total</td>
<td>960K</td>
</tr>
</tbody>
</table>

**Performance:**

- * x86-64 kernel AFL *
- ( 1 Processes)
- * Performance: [ ] 659 t/s *

**Fuzzing Technique Progress**

- Bitflipping: [ ] 778
- Arithmetic: [ ] 10K
- Interesting: [ ] 1.4K
- Havoc: [ ] 4.7K
- Splicing: [ ] 4.7K

**Other Details**

- Panic: 78 (2)
- KASan: 0 (0)
- Payload-Size: 34B
- Total: 960K
- CPU: 02.7%
- RAM: 02.7%
- Timeout: 0 (0)
- Splicing: 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
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**

```c
int main(int argc, char **argv) {
    WNDCLASSEX WindowClass = {0};
    HWND WindowA, WindowB, WindowC;
    ATOM Atom;
    WindowClass.cbSize = sizeof(WNDCLASSEX);
    WindowClass.lpfnWndProc = DefWindowProc;
    WindowClass.lpsz classNames = "Class";
    Atom = RegisterClassEx(&WindowClass);
    WindowA = CreateWindowEx(0, MAKEINTATOM(Atom), "One", 0, 0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, 0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, 0, 128, 128, NULL, NULL, NULL, NULL);
}
```
Typical kernel bug: **CVE-2018-0744**

```c
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, 0, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, 0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, 0, 128, 128, NULL, NULL, NULL, NULL);
}
```
Typical kernel bug: **CVE-2018-0744**

```c
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, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, 128, 128, NULL, NULL, NULL, NULL);
}
```
Typical kernel bug: **CVE-2018-0744**

```c
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, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, 128, 128, NULL, NULL, NULL, NULL);
}
```
Typical kernel bug: CVE-2018-0744

```c
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, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, 128, 128, NULL, NULL, NULL, NULL);
}
```
Typical kernel bug: **CVE-2018-0744**

```c
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, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, 128, 128, NULL, NULL, NULL, NULL);
}
```
Typical kernel bug: **CVE-2018-0744**

```c
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, 128, 128, NULL, NULL, NULL, NULL);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC);
    WindowB = CreateWindowEx(0, MAKEINTATOM(Atom), "Two", 0, 128, 128, NULL, NULL, NULL, NULL);
    GetDC(WindowA);
    SetClassLong(WindowA, GCL_STYLE, CS_CLASSDC | CS_OWNDC);
    WindowC = CreateWindowEx(0, MAKEINTATOM(Atom), "Three", 0, 128, 128, NULL, NULL, NULL, NULL);
}
```
Typical kernel bug: **CVE-2018-0744**

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

VS

syscall fuzzing
```c
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

<table>
<thead>
<tr>
<th>Title</th>
<th>Repro</th>
<th>Count</th>
<th>Last</th>
<th>Reported</th>
<th>Closed</th>
<th>Patch</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUG: bad unlock balance in jpmr_mfe_seq_stop</td>
<td>C</td>
<td>7493</td>
<td>692d</td>
<td>771d</td>
<td>692d</td>
<td>7d3d68ef <a href="https://github.com/intel/syzkaller/commit/7d3d68ef">jpmr: fix stale iterator</a></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in __lock_acquire</td>
<td>C</td>
<td>1161</td>
<td>700d</td>
<td>770d</td>
<td>692d</td>
<td>55b010d0 <a href="https://github.com/intel/syzkaller/commit/55b010d0">UPSTREAM: ANDROID: binder: remove waltqueue when thread exits</a></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in bio_copy_user_iow</td>
<td>syz</td>
<td>73</td>
<td>850d</td>
<td>890d</td>
<td>805d</td>
<td>4099ac93 <a href="https://github.com/intel/syzkaller/commit/4099ac93">scsi: sg: protect accesses to 'reserved' page array</a></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in fanout_demux_rollover</td>
<td>C</td>
<td>5</td>
<td>833d</td>
<td>854d</td>
<td>791d</td>
<td>6f7c74d4a <a href="https://github.com/intel/syzkaller/commit/6f7c74d4a">packet: hold bind lock when rebinding to fanout hook</a></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in parse_ipsecrequests</td>
<td>C</td>
<td>7</td>
<td>897d</td>
<td>897d</td>
<td>867d</td>
<td>3c17d418 <a href="https://github.com/intel/syzkaller/commit/3c17d418">UPSTREAM: af_key: Fix sadb_x_ipsecrquest parsing</a></td>
</tr>
</tbody>
</table>
Syzkaller

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

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

CVE-2019-2215
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

\[ r_0 = \text{open}((0x7f0000000000)="./file0", 0x3, 0x9) \]
Syzkaller Generated Program

\[ r_0 = \text{open}(&0x7f0000000000=\text{/file0}, 0x3, 0x9) \]
\[ \text{read}(r_0, &0x7f0000000010, 57) \]
\[ \text{close}(r_0) \]
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, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH
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, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH
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, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH

read(fd fd, buf buffer[out], count len[buf])
Generation
Generation syz-syssgen
Generation

syz-sysgen

Prog objects
Generation

syz-sysgen

Prog objects → syz-fuzzer

Generated Program

- open(...)  
- read(...)  
- close()
Mutations - insertCall

```c
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

```c
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

```c
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

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

syz-manager

load, store

Corpus
Syzkaller - Architecture (Linux)

syz-manager

load, store

VM management

Corpus

Kernel

Test Machine
Syzkaller - Architecture (Linux)

- syz-manager
- scp, ssh
- sshd
- Corpus
- load, store
- VM management
- Kernel
- Test Machine
Syzkaller - Architecture (Linux)

- syz-manager
- scp, ssh
- sshd invokes syz-fuzzer
- load, store
- VM management
- Corpus

Test Machine

Kernel
Syzkaller - Architecture (Linux)

- syz-manager
- Corpus
- load, store
- scp, ssh
- RPC
- VM management
- sshd
- invoke
- syz-fuzzer
- Kernel
- Test Machine
Syzkaller - Architecture (Linux)
Syzkaller - Architecture (Linux)

- **syz-manager**
  - scp, ssh
  - RPC

- **Corpus**
  - load, store
  - VM management

- **sshd**
  - invoke

- **syz-fuzzer**
  - IPC

- **syz-executor**
  - syscall

- **Kernel**

- **Test Machine**
Syzkaller - Architecture (Linux)

- syz-manager
- Corpus
  - load, store
  - VM management
- scp, ssh
- RPC
- sshd
  - invoke
  - syz-fuzzer
  - IPC
  - syz-executor
    - coverage
    - syscalls
    - KCOV
    - Kernel
    - Test Machine
Syzkaller - Architecture (Linux)

- syz-manager
  - scp, ssh
  - RPC
  - load, store
  - VM management

- sshd
  - invoke

- syz-fuzzer
  - IPC
  - coverage

- syz-executor
  - coverage
  - syscalls
  - KCOV

- Kernel

- Test Machine
Test case generator/mutator
Feedback mechanism
Bug oracle
Test case generator/mutator
Feedback mechanism
Bug oracle
Syzkaller - Architecture (Linux)

- **syz-manager**
- **Corpus**
- **load, store**
- **VM management**
- **scp, ssh**
- **RPC**
- **sshd**
- **invoke**
- **syz-fuzzer**
- **IPC**
- **coverage**
- **syz-executor**
- **syscalls**
- **coverage**
- **KCOV**
- **Kernel**
- **Test Machine**
- **Console Output**
Syzkaller - Architecture (Linux)

```
r0 = syz_open_procfs(0x0, &0x7f0000000000='setgroups\x00')
write$P9_RAUTH(r0, 0x0, 0x6)
```

Console Output
Test case generator/mutator
Feedback mechanism
Bug oracle
Syzkaller - Unsupervised

syz-manager
Syzkaller - Unsupervised

- syz-manager
- syz-fuzzer
- syz-executor
- sshd

RPC:
- scp, ssh
- invoke

Coverage:
- IPC
- coverage

Kernel:
- KCOV
- syscalls

Reproducer Machine

SYZ
Syzkaller - Unsupervised

- syz-manager
- syz-fuzzer
- syz-executor

Console Output

Kernel

KCOV

RPC

scp, ssh

invoke

IPC

coverage

syscalls

Reproducer Machine

SYZ

C
Syzkaller is awesome!
But...
But...

Syzkaller is for Linux
Syzkaller is for Linux

But...

Dmitry Vyukov (syzkaller’s father)

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

It supports 6 full-fledged OSes, not counting Windows, Trusty and gVisor.
Dmitry Vyukov (syzkaller’s father)

But... Syzkaller is for **Linux** also

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

[google/syzkaller](https://github.com)

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

Require less resources than running a **Virtual Machine**.

Aimed for running **bash and core Linux commands**.
WSL - Bird’s eye view

NT Process
- PEB
- TEB
- Dlls

Minimal Process

NT Kernel
WSL - Bird’s eye view

- NT Process:
  - PEB
  - TEB
  - Dlls

- Minimal Process:

- Pico Process:
  - ELF

NT Kernel
WSL - Bird’s eye view

NT Process
- PEB
- TEB
- Dlls

Minimal Process

Pico Process
- ELF

NT Kernel

Pico Provider Lxss/Lxcore
Why WSL?

Similar to fuzzing Linux

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

First steps with syzkaller
What do we need to get started

- Test Machine
- syz-manager
- syscalls coverage
- scp, ssh
- RPC
- Corpus
- load, store
- VM management
- syz-executor
- IPC
- coverage
- syscall
- syz-fuzzer
- stdout
- Console Output
What do we need to get started

- syz-manager
- scp, ssh
- RPC
- load, store
- VM management
- Corpus
- syz-manager
- syz-fuzzer
- IPC
- coverage
- syscall
- vm-executor
- sshd
- KCOV
- Console Output
- Test Machine
- Kernel
What do we need to get started

- syz-manager
- scp, ssh, RPC
- Corpus
- sshd
- syz-fuzzer
- syz-executor
- coverage
- syscalls
- KCOV
- IPC
- VM management
- load, store
- Console Output
- Test Machine
- Kernel
- Output
What do we need to get started

- syz-manager
- scp, ssh
- RPC
- Corpus
- load, store
- VM management
- sshd
- invoke
- syz-fuzzer
- IPC
- syz-executor
- coverage
- syscalls
- KCOV
- Kernel
- Crash detection & symbolize
- Console Output
- Test Machine

- syz-executor
- syscalls
- IPC
- coverage
- KCOV
- Kernel
- Test Machine
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

- **syz-manager**
- **Corpus**
- **load, store**
- **VM management**
- **scp, ssh**
- **RPC**
- **sshd**
- **invoke**
- **syz-fuzzer**
- **syz-executor**
- **IPC**
- **coverage**
- **syscalls**
- **KCOV**
- **Kernel**
- **Test Machine**
- **Console Output**
Coverage - Implementation

- syz-manager
- scp, ssh
- RPC
- load, store
- VM management
- Corpus

- sshd
- invoke
- syz-fuzzer
- IPC
- coverage
- syscalls

- Kernel
- Test Machine
- KVM - Intel PT

Console Output
Syzkaller - **WSL** coverage

<table>
<thead>
<tr>
<th>Function</th>
<th>Start Address</th>
<th>Coverage</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>LxpEpollContextAddFileDescriptor</td>
<td>0x1C004D5C</td>
<td>37 / 79</td>
<td>218 / 369</td>
</tr>
<tr>
<td>LxpMiFreeOverlappingRange</td>
<td>0x1C0086200</td>
<td>43 / 68</td>
<td>209 / 297</td>
</tr>
<tr>
<td>LxpDevTerminalDefaultDeviceControl</td>
<td>0x1C00F68B</td>
<td>37 / 89</td>
<td>208 / 447</td>
</tr>
<tr>
<td>LxpMmMremap</td>
<td>0x1C0088AEC</td>
<td>54 / 100</td>
<td>325 / 499</td>
</tr>
<tr>
<td>LxpEpollContextWait</td>
<td>0x1C004E768</td>
<td>53 / 108</td>
<td>256 / 533</td>
</tr>
<tr>
<td>LxpDevTtyInputWorker</td>
<td>0x1C0101070</td>
<td>0 / 80</td>
<td>0 / 359</td>
</tr>
<tr>
<td>LxpProcFsRootCpuInfoGenerateStringCommon</td>
<td>0x1C00AEADC</td>
<td>0 / 65</td>
<td>0 / 520</td>
</tr>
<tr>
<td>LxpProcFsPidMapsGenerateBufferCommon</td>
<td>0x1C00AA51C</td>
<td>0 / 97</td>
<td>0 / 544</td>
</tr>
<tr>
<td>LxpSemPerformOperations</td>
<td>0x1C00F33F0</td>
<td>19 / 114</td>
<td>142 / 473</td>
</tr>
<tr>
<td>VfsRename</td>
<td>0x1C00929F8</td>
<td>67 / 95</td>
<td>376 / 462</td>
</tr>
<tr>
<td>LxUtilNtStatusToLxError</td>
<td>0x1C00FA894</td>
<td>84 / 94</td>
<td>171 / 191</td>
</tr>
<tr>
<td>LxpElfInfoParse</td>
<td>0x1C004CABC</td>
<td>0 / 127</td>
<td>0 / 541</td>
</tr>
<tr>
<td>LxpDevLogTraceLoggingWrite</td>
<td>0x1C00B5024</td>
<td>0 / 105</td>
<td>0 / 431</td>
</tr>
<tr>
<td>LxpMmAllocateMapVm</td>
<td>0x1C0086860</td>
<td>84 / 150</td>
<td>424 / 615</td>
</tr>
<tr>
<td>LxpProcFsPidStatusGenerateBuffer</td>
<td>0x1C00ABC50</td>
<td>0 / 110</td>
<td>0 / 664</td>
</tr>
<tr>
<td>LxpDevTerminalProcessInput</td>
<td>0x1C00FE72C</td>
<td>110 / 149</td>
<td>411 / 510</td>
</tr>
<tr>
<td>_output_1</td>
<td>0x1C0009560</td>
<td>67 / 182</td>
<td>252 / 573</td>
</tr>
</tbody>
</table>

Composer
Syzkaller - **WSL** coverage
Bug Oracle

We used the same technique as kAFL
Bug Oracle

We used the same technique as kAFL

Enabled driver verifier for lxcore & lxss
Symbolizing crashes
Symbolizing crashes
Symbolizing

crashes
Symbolizing
crashes
Syzkaller - **WSL Symbolizer**

- QEMU - Test Machine
- Symbolizer Machine
- syz-manager
- KVM

User

Kernel

Registers, Stack, Modules addresses

symbolizer

PDB
Syzkaller - **WSL Symbolizer**

```
0x98766 0x422f4 0x42024 0x41dc0 0x41d60 0x3f9fc
```
Syzkaller - **WSL Symbolizer**

Callstack:

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

syz-manager

Corpus

load, store

VM management

syz-fuzzer

IPC

syz-executor

coverage

syscalls

Kernel

Test Machine

KVM - Intel PT

Console Output

scp, ssh

RPC

invoke

sshd
Recap

- syz-manager
- Corpus
- scp, ssh
- RPC
- VM management
- load, store
- sshd
- invoke
- syz-fuzzer
- IPC
- syz-executor
- coverage
- syscall
- Kernel
- Test Machine
- Console Output
- KVM - Intel PT
Recap

- syz-manager
  - load, store
  - VM management

- Corpus

- scp, ssh, RPC

- svz
  - syscalls coverage

- svz-executor
  - IPC
  - coverage
  - syscalls

- svz-fuzzer
  - invoke

- sshd

- Kernel

- Test Machine

- KVM - Intel PT

Crash detection & symbolize

Console Output
And then it rained bugs!
And then it rained bugs!

Not really
Not really

We noticed a crash with CRITICAL_STRUCTURE_CORRUPTION
We noticed a crash with CRITICAL_STRUCTURE_CORRUPTION

Not really
We noticed a crash with CRITICAL_STRUCTURE_CORRUPTION

```
mov rax, rcx
mov rbx, rdx
mov rcx, r8
mov rdx, r9
vmcall
```
We noticed a crash with CRITICAL_STRUCTURE_CORRUPTION

Not really

We noticed a crash with CRITICAL_STRUCTURE_CORRUPTION

Good News Everyone

Patch Guard Is Working
Crash detection

syz-manager

Corpus

load, store

VM management

scp, ssh

RPC

syz-fuzzer

syz-executor

IPC

coverage

syz-executor

Console Output

Kernel

Test Machine

KVM - Intel PT
Crash detection

- syz-manager
- Corpus
- load, store
- scp, ssh
- RPC
- sshd
- invoke
- syz-fuzzer
- IPC
- syz-executor
- coverage
- syscalls
- Kernel
- panic_reporter.sys
- Console Output
- Test Machine
- KVM - Intel PT
Crash detection

- syz-manager
- Corpus
- scp, ssh
- RPC
- VM management
- load, store
- syz-executor
- syscalls
- coverage
- IPC
- invoke
- syz-fuzzer
- sshd
- Console Output
- Test Machine
- Kernel
- panic_reporter.sys
- ntoskrl
- Intel PT
- KVM
Crash detection

syz-manager

Corpus

load, store

RPC

VM management

scp, ssh

syz-executor

syz-fuzzer

IPC

coverage

syz-fuzzer

invoke

sshd

KVM - Intel PT

Test Machine

Console

Output

Kernel

panic_reporter.sys

ntoskrnl

CRASH!
Crash detection

- **syz-manager**
- **Corpus**
- **syz-fuzzer**
- **syz-executor**
- **panic_reporter.sys**
- **ntoskrnl**
- **sshd**
- **scp, ssh**
- **RPC**
- **VM management**
- **IPC**
- **coverage**
- **load, store**
- **KVM - Intel PT**
- **Test Machine**
- **Kernel**
- **Console Output**

Diagram showing the flow of processes and components involved in crash detection.
Coverage stability

Thread 1
- Syscall 1
- Syscall 2
- ...

Thread 2
- Syscall 3
- Syscall 4
- ...

...
# Coverage stability

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Coverage 1</th>
<th>Coverage 2</th>
<th>Coverage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syscall 1</td>
<td>Syscall 3</td>
<td>Syscall 1</td>
<td>Syscall 1</td>
<td>Syscall 3</td>
</tr>
<tr>
<td>Syscall 2</td>
<td>Syscall 4</td>
<td>Syscall 2</td>
<td>Syscall 3</td>
<td>Syscall 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Syscall 3</td>
<td>Syscall 2</td>
<td>Syscall 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Syscall 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Coverage stability

**TL; DR**
- We added OS thread tracking to KVM
- Allocating a buffer for each thread
<table>
<thead>
<tr>
<th>Autorun Entry</th>
<th>Description</th>
<th>Publisher</th>
<th>Image Path</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run</td>
<td>OneDrive</td>
<td>Microsoft OneDrive</td>
<td>c: \users\user\appdata\local\microsoft\onedrive.exe</td>
<td>21/12/2018 3:32</td>
</tr>
<tr>
<td>HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components</td>
<td></td>
<td></td>
<td></td>
<td>07/08/1973 7:28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12/04/2018 1:16</td>
</tr>
</tbody>
</table>
### Glitches be crazy

<table>
<thead>
<tr>
<th>Autorun Entry</th>
<th>Description</th>
<th>Publisher</th>
<th>Image Path</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run</td>
<td></td>
<td>Microsoft OneDrive</td>
<td>c:\users\user\appdata\local\microso...</td>
<td>21/12/2018 3:32</td>
</tr>
<tr>
<td>HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components</td>
<td></td>
<td></td>
<td></td>
<td>07/08/1973 7:28</td>
</tr>
<tr>
<td>HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run</td>
<td></td>
<td></td>
<td></td>
<td>12/04/2018 1:16</td>
</tr>
</tbody>
</table>

- **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
Glitches be crazy

<table>
<thead>
<tr>
<th>Autorun Entry</th>
<th>Description</th>
<th>Publisher</th>
<th>Image Path</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run</td>
<td>OneDrive</td>
<td>Microsoft OneDrive</td>
<td>Microsoft Corporation</td>
<td>c:\users\user\appdata\local\microsoft</td>
</tr>
<tr>
<td>HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components</td>
<td>Windows Time</td>
<td></td>
<td></td>
<td>12/04/2018 1:16</td>
</tr>
<tr>
<td></td>
<td>Windows Update</td>
<td>Enables the...</td>
<td>Manual (Trigger Start)</td>
<td>Local System</td>
</tr>
<tr>
<td></td>
<td>Windows Update</td>
<td>Enables rem...</td>
<td>Manual</td>
<td>Local System</td>
</tr>
<tr>
<td></td>
<td>Allow antimalware service to startup with normal priority</td>
<td>Not configured</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Turn off Windows Defender Antivirus</td>
<td>Enabled</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configure local administrator merge behavior for lists</td>
<td>Not configured</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
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

```c
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);
}
```

< 140
WSL - results

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

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

```c
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 😞
Let’s move to a real target
Windows Kernel

Executive Services

I/O Manager
- Power Manager
- Security
- Process Manager
- Virtual Memory Manager
- IPC

Kernel

Kernel Mode

Win32
- GDI

DirectX

Graphic Drivers

HAL
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.tech blog](https://j00ru.tech/blog))

See also: Windows System Call Tables in CSV/JSON formats on [GitHub](https://github.com/MOnsic/WinSysCallTables)

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

Layout by Metasploit Team

|--------------------|------------------|---------------------------|---------------------|---------------------------|-----------------|----------------------------|----------------|-----------------|
Win32k - changes

syz-manager

Corpus

load, store

VM management

scp, ssh

RPC

syz-fuzzer

IPC

coverage

syscalls

syz-executor

Kernel

panic_reporter.sys

ntoskrnl

Test Machine

KVM - Intel PT

Console Output

Invoke

Test Machine

syz-fuzzer

invoke

syz-executor
Win32k - changes

- syz-manager
- scp, ssh
- RPC
- VM management
- load, store
- Corpus
- sshd
- syz-fuzzer
- IPC
- syz-executor
- coverage
- syscalls
- kernel
- panic_reporter.sys
- ntoskrnl
- Kernel
- Test Machine
- Console Output
- KVM - Intel PT
Win32k - changes

- syz-manager
  - load, store
  - scp, ssh
  - RPC

- Corpus
- syscalls
- coverage
- load, store
- VM management

- sshd
  - invoke
- syz-fuzzer.exe
  - IPC
- syz-executor.exe
- syscall

- Kernel
  - panic_reporter.sys
  - ntoskrnl

- Test Machine
  - KVM - Intel PT

Console Output
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++

```
win32u.dll  win32u.def
```

```
gdi32.dll   gdi32.def
```
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++

```
win32u.dll -> win32u.def

gdi32.dll -> gdi32.def

mingw-dlltool

win32u.lib

gdi32.lib

syz-executor.exe
```
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
Win32k - Recap

Start the test machine

- syz-manager
  - load, store
  - Corpus

- scp, ssh
  - RPC

- VM management

- syz-fuzzer.exe
  - invoke
  - syz-executor.exe
  - IPC
  - coverage
  - syscalls

- sshd

- Kernel
  - panic_reporter.sys
  - ntoskrnl

- Test Machine

- Console Output

- KVM - Intel PT
Win32k - Recap

- syz-manager
- Corpus
- scp, ssh
- RPC
- load, store
- VM management
- sshd
- invoke
- syz-fuzzer.exe
- IPC
- syz-executor.exe
- coverage
- syscalls

Test Machine
- panic_reporter.sys
- ntoskrnl
- Kernel
- Console Output
- KVM - Intel PT
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
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

Reproduces on some machines

What the **fuzz**?
First Bug  UAF

Reproduces on some machines

What the fuzz?

void crash() {
    OpenClipboard(0);
    HWINSTA wnd1 = OpenClipboard(0, 0, WINSTA_READSCREEN | WINSTA_ENUMDESKTOPS, 0);
    HWINSTA wnd2 = GetDesktopWindow();
    OpenClipboard(wnd2);
    SetProcessWindowStation(wnd1);
}
First Bug  UAF

```c
UserSetLastError(5i64);

// ;
// ; Check if ETW 487d6e37-1b9d-46d3-a8fd-54ce8bdf8a53 has specific flag
// ;
if ((unsigned int)Win32kTraceLoggingLevel > 5
    && TlgKeywordOn((__int64)&Win32kTraceLoggingLevel, 0x400000000000164))
{
    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

```c
UserSetLastError(5164);
    //
    // ; Check if ETW 487d6e37-1b9d-46d3-a8fd-54ce8bdf8a53 has specific flag
    //
    if ((unsigned int)Win32kTraceLoggingLevel > 5
        && TLgKeywordOn((__int64)&Win32kTraceLoggingLevel, 0x4000000000000000)
    )
    {
        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

```c
UserSetLastError(5i64);
// ;
// ; Check if ETW 487d6e37-1b9d-46d3-a8fd-54ce8bdf8a53 has specific flag
// ;
if ( (unsigned int)Win32kTraceLoggingLevel > 5
    && TlgKeywordOn((__int64)&Win32kTraceLoggingLevel, 0x4000000000000164) )
{
    thread_info = *(_QWORD *)(winsta_tag + 48);
    // ;
    // ; pti = tagTHREADINFO
    // ; ppi = PPROCESSINFO
    // ;
    pti->ppi->W32Pid
    // ;
    v14 = *(_DWORD *)(*(QWORD *)(thread_info + 416) + 56i64);
    v17 = &v14;
    v18 = 4i64;
    TlgCreateWs((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

```c
UserSetLastError(5i64);
// ;
// ; Check if ETW 487d6e37-1b9d-46d3-a8fd-54ce8bdf8a53 has specific flag
// ;
if ( (unsigned int)Win32kTraceLoggingLevel > 5
    && TlgKeywordOn(__int64)&Win32kTraceLoggingLevel, 0x4000000000000000i64 )
{
    thread_info = *(QWORD *)(winsta_tag + 48);
    // ;
    // ; // pti = tagTHREADINFO
    // ; // ppi = PPROCESSINFO
    // ;
    // ; pti->ippi->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 DON'T 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

We got another bug
2nd Bug

DoS in RegisterClassExA

```c
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?
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?
Win32k - grammar

- Technically windows is open source
  - Windows NT Leaked sources - https://github.com/ZoloZiak/WinNT4
  - 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
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

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)”

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

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

```c
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)

Results - Win32k

- 60 vCPUs & 1.5 months of fuzzing part time
- 8 vulns, 6 CVEs (1 duplicate, 1 pending)
- 3 DOS, 1 crash in WinLogon and few deadlocks
LPE $\rightarrow$ RCE?
**WMF - Windows Metafile Format**

Designed back in 1990s

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

The image is basically a list of records describing GDI calls

Microsoft Extended WMF:

- EMF
- EMF+
- EMFSPOOL

https://j00ru.vexillium.org/slides/2016/pacsec.pdf
## EMF - Enhanced Metafile Format

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3</td>
<td>EMF Records</td>
<td>73</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Bitmap Record Types</td>
<td>74</td>
</tr>
<tr>
<td>2.3.1.1</td>
<td>EMR_ALPHABLEND Record</td>
<td>76</td>
</tr>
<tr>
<td>2.3.1.2</td>
<td>EMR_BITBLT Record</td>
<td>80</td>
</tr>
<tr>
<td>2.3.1.3</td>
<td>EMR_MASKBLT Record</td>
<td>83</td>
</tr>
<tr>
<td>2.3.1.4</td>
<td>EMR_PLGBLT Record</td>
<td>87</td>
</tr>
<tr>
<td>2.3.1.5</td>
<td>EMR_SETDIBITSTODEVICE Record</td>
<td>90</td>
</tr>
<tr>
<td>2.3.1.6</td>
<td>EMR.StretchBLT Record</td>
<td>93</td>
</tr>
<tr>
<td>2.3.1.7</td>
<td>EMR.StretchDIBITS Record</td>
<td>95</td>
</tr>
<tr>
<td>2.3.1.8</td>
<td>EMR_TRANSPARENTBLT Record</td>
<td>98</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Clipping Record Types</td>
<td>101</td>
</tr>
</tbody>
</table>
# EMF - Enhanced Metafile Format

## 2.3 EMF Records

### 2.3.1 Bitmap Record Types

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMR_ALPHABLEND Record</td>
<td>76</td>
</tr>
<tr>
<td>EMR_BITBLT Record</td>
<td>80</td>
</tr>
<tr>
<td>EMR_MASKBLT Record</td>
<td>83</td>
</tr>
<tr>
<td>EMR_PLGBLT Record</td>
<td>87</td>
</tr>
<tr>
<td>EMR_SETDIBITSTODEVICE Record</td>
<td>90</td>
</tr>
<tr>
<td>EMR_STRETCHBLT Record</td>
<td>93</td>
</tr>
<tr>
<td>EMR_STRETCHDIBITS Record</td>
<td>95</td>
</tr>
<tr>
<td>EMR_TRANSPARENTBLT Record</td>
<td>98</td>
</tr>
</tbody>
</table>

### 2.3.2 Clipping Record Types

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Page</th>
</tr>
</thead>
</table>

---
# EMF - Enhanced Metafile Format

## 2.3 EMF Records

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Record Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.1</td>
<td><strong>Bitmap Record Types</strong></td>
<td></td>
</tr>
<tr>
<td>2.3.1.1</td>
<td>EMR_ALPHABLEND Record</td>
<td>76</td>
</tr>
<tr>
<td>2.3.1.2</td>
<td>EMR_BITBLT Record</td>
<td>80</td>
</tr>
<tr>
<td>2.3.1.3</td>
<td>EMR_MASKBLT Record</td>
<td>83</td>
</tr>
<tr>
<td>2.3.1.4</td>
<td>EMR_PLGBLT Record</td>
<td>87</td>
</tr>
<tr>
<td>2.3.1.5</td>
<td>EMR_SETDIBITSTODEVICE Record</td>
<td>90</td>
</tr>
<tr>
<td>2.3.1.6</td>
<td><strong>EMR_STRETCHBLT Record</strong></td>
<td>93</td>
</tr>
<tr>
<td>2.3.1.7</td>
<td>EMR_STRETCHDIBITS Record</td>
<td>95</td>
</tr>
<tr>
<td>2.3.1.8</td>
<td>EMR_TRANSPARENTBLT Record</td>
<td>98</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Clipping Record Types</td>
<td>101</td>
</tr>
</tbody>
</table>
We happen to have a vulnerability in **StretchBlt**
GDI -> **EMF** -> **RCE: DEMO**
Future work

- Win32k
  - DirectX drivers
  - Win32 callbacks (callbacks in general)
- Bochspwn Reloaded + Generated corpus (450k)
- Open source
- The rest of the kernel
Summary

50 CVEs
Summary

50 CVEs

Wanted Kernel

kAFL
Summary

- 50 CVEs
- kAFL
- syzkaller

Wanted Kernel
Syscall fuzzing
Summary

50 CVEs

kAFL

syzkaller

Wanted Kernel

Syscall fuzzing

Porting

WSL
Summary

Wanted Kernel

Syscall fuzzing

50 CVEs

kAFL

syzkaller

Win32k

WSL

Porting

No luck
Summary

- 50 CVEs
- Wanted Kernel
- kAFL
- Syscall fuzzing
- syzkaller
- Grammar mining
- Win32k
- 8 Vulns

Porting

No luck
Summary

WSL

50 CVEs

Wanted Kernel

kAFL

Syscall fuzzing

syzkaller

Grammar mining

Win32k

LPE → RCE

8 Vulns

EMF

Porting

No luck
Special Thanks
Dynamic Tools team @Google
Omri Herscovici
Ran Menscher

Summary

50 CVEs
Wanted Kernel
kAFL
Syscall fuzzing
syzkaller
Grammar mining
Win32k
LPE → RCE
8 Vulns
EMF

Porting
No luck
Questions?

@YoavAlon

@NetanelBenSimon