Context Switching your Kernel Fuzzing @BSides Stuttgart 2019

Eric Sesterhenn <eric.sesterhenn@x41-dsec.de>

2019

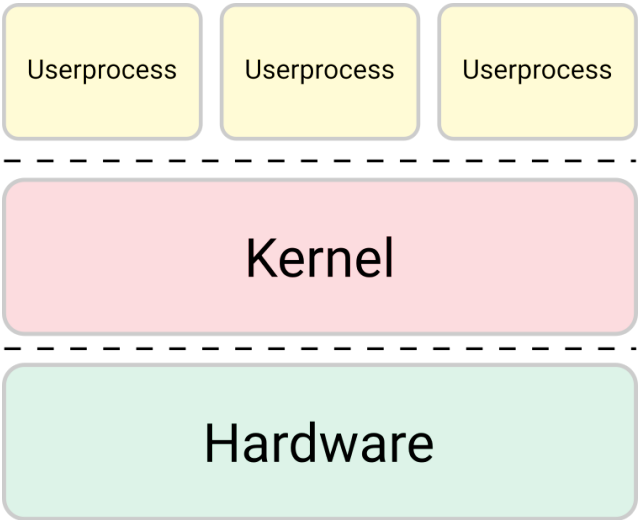
X41 D-SEC GmbH

whoami

Eric Sesterhenn

- Pentesting/Code Auditing at X41
- Kernel stuff every now and then
- Recent topics: smartcards+fax machines

Kernel and Userspace



Why Kernel Fuzzing

- Kernel bugs offer high privileges
- Might allow sandbox escape
- In some cases even remotely exploitable
- Whatever, its fun :)

EULA

My code is crappy, I will not release anything related to this

To determine which one of the two separate sets of license terms below apply to you, check the license designation. This is printed either on your product key, or might be shown on your Certificate of Authenticity, or on the download page if you obtained the software online. If your designation is "FPP," "Retail" or "PIPC," then the Retail License Terms below apply to you. If OEM, then the OEM License Terms below apply to you. If you need help, please go to microsoft.com/office/eula to determine which license you have.

RETAIL LICENSE TERMS

Thank you for choosing Microsoft Office 2013. This is a license agreement between you and Microsoft Corporation (or, based on where you live, one

How do Others Fuzz the Linux Kernel

- fsfuzzer - bitflips in fs images
  `https://people.redhat.com/sgrubb/files/`
- trinity - random syscalls
  `https://codemonkey.org.uk/projects/trinity/`
- syzcaller - coverage guided syscalls
  `https://github.com/google/syzkaller`
- difuze - llvm analysis of ioctl `https://github.com/ucsb-seclab/difuze`
- kafl - afl for qemu `https://github.com/RUB-SysSec/kAFL`

What I dislike

- Fuzzing the currently running kernel might cause instabilities / side effects
- Virtualization and instrumentation might cause overhead
- Not enough pain for me ... ;-)

Why not Both?

Why don't we run the kernelcode in userspace?

Let the porting begin

- Ported EXT2 some years ago to userspace
- Crudely ripping stuff apart and adding glue
- Did so again with ASN1 parsers, QNXFS and CRAMFS

Head -> Table

```
struct page {
        unsigned long flags;
        struct address_space *mapping;
};
#define sysfs_remove_mount_point(a, b) 0
#define sysfs_create_file_ns(a, b, c) 0
#define sysfs_create_files(a, b) 0
#define sysfs_chmod_file(a, b, c) 0
#define kvmalloc(a, b) malloc(a)
#define kvfree(a) free((void *)a)
```

After a lot of pain... - sprint_oid()

```
while (v < end) {
        num = 0;
        n = *v++;
        ...
        num = n;
        ...
        ret += count = snprintf(buffer, bufsize, ".%lu", num);
        buffer += count;
        bufsize -= count;
        if (bufsize == 0)
        return -1;
}
```

Close, but no cigar

```
int vsnprintf(char *buf, size_t size, const char *fmt, va_list args)
{
        ...
        /* Reject out-of-range values early.  Large positive sizes are
        used for unknown buffer sizes. */
        if (WARN_ON_ONCE(size > INT_MAX))
                return 0;
```

Schmerz! What about UML?

User-mode Linux (UML) enables multiple virtual Linux kernel-based operating systems (known as guests) to run as an application within a normal Linux system (known as the host).

How?

```
make ARCH=um defconfig
make ARCH=um
./linux
```

How to AFL?

```
cp /usr/bin/ar /path-to-afl/afl-ar
cp /usr/bin/objcopy /path-to-afl/afl-objcopy
cp /usr/bin/nm /path-to-afl/afl-nm
cp /usr/bin/ld /path-to-afl/afl-ld
cp /usr/bin/objdump /path-to-afl/afl-objdump
CROSS_COMPILE=/path-to-afl/afl- make ARCH=um
```

How to AFL?

- *arch/x86/um/vdso/checkundef.sh* doesnt like us - just ***exit 0***
- *arch/x86/um/vdso/vma.c* ***init_vdso()*** neither - simply ***return -ENOMEM***
- Some *.config* tweaks

How to AFL?

- Lets start fuzzing filesystems ... seems easy
- `./linux udb0=root_fs`
- insert ***machine_halt()*** in *init/do_mounts.c* to exit as early as possible

Damn again!

- Lots of non-reproducible crashes (0.5%)
- Debugging...
- Not caused by me :(

```
#66 0x0000000060024dad in write_sigio_workaround () at
↪  arch/um/os-Linux/sigio.c:309
#67 sigio_broken (fd=<optimized out>, read=<optimized out>) at
↪  arch/um/os-Linux/sigio.c:344
#68 0x0000000060003c44 in rng_init () at arch/um/drivers/random.c:137
#69 0x0000000060016429 in do_one_initcall (fn=0x60003bbc <rng_init>) at
↪  init/main.c:887
#70 0x0000000060001d7b in do_initcall_level (level=<optimized out>) at
↪  ./include/linux/init.h:131
#71 do_initcalls () at init/main.c:964
#72 do_basic_setup () at init/main.c:982
#73 kernel_init_freeable () at init/main.c:1138
#74 0x0000000060236ef3 in kernel_init (unused=<optimized out>) at
↪  init/main.c:1055
#75 0x0000000060017ca1 in new_thread_handler () at arch/um/kernel/process.c:133
#76 0x0000000000000000 in ?? ()
```

Third time is the charm

- Lets combine the approaches
- Use UML for easy AFL/Userspace compilation

Link UML

```
$ afl-gcc  ./fs/cifs/asn1.o
afl-cc 2.51b by <lcamtuf@google.com>
/usr/bin/ld: /usr/lib/gcc/ .. /Scrt1.o: in function `_start':
(.text+0x20): undefined reference to `main'
/usr/bin/ld: ./fs/cifs/asn1.o: in function `kmalloc_array':
./include/linux/slab.h:668: undefined reference to `__kmalloc'
/usr/bin/ld: ./fs/cifs/asn1.o: in function `asn1_oid_decode':
fs/cifs/asn1.c:466: undefined reference to `kfree'
/usr/bin/ld: ./fs/cifs/asn1.o: in function `decode_negTokenInit':
fs/cifs/asn1.c:526: undefined reference to `kfree'
/usr/bin/ld: fs/cifs/asn1.c:526: undefined reference to `kfree'
collect2: error: ld returned 1 exit status
```

Link UML

```
void *__kmalloc(size_t size, int flags) {
        return malloc(size);
}

void kfree(void *objp) {
        free(objp);
}

int main(int argc, char **argv) {
        ...
        return decode_negTokenInit(blob, i, server);
}
```

What else

- *-Wl,–unresolved-symbols=ignore-in-object-files*
- *AFL_HARDEN / AFL_USE_ASAN work as well*

What to fuzz

- I dont care about the actual bugs, only the approach
- Priority: Easy to fuzz targets
- `egrep -r "\(.*char \*.*,.*size_t .*\)"`

Lets try

842 compression is used by zram, yeah unlikely attack surface, maybe wireguard uses it at some point ;-)

```
AFL_HARDEN=1 ~/tools/afl/afl-2.51b/afl-gcc test.c
↪  ./lib/842/842_decompress.o
↪  -Wl,--unresolved-symbols=ignore-in-object-files
```

Baam

```
--- a/lib/842/842_decompress.c
+++ b/lib/842/842_decompress.c
@@ -211,6 +211,8 @@ static int __do_index(struct sw842_param *p, u8 size, u8 bits, u64 fsize)
(unsigned long)total,
(unsigned long)beN_to_cpu(&p->ostart[offset], size));

+       if (p->olen < size)
+               return -EINVAL;
        memcpy(p->out, &p->ostart[offset], size);
        p->out += size;
        p->olen -= size;
@@ -354,6 +356,8 @@ int sw842_decompress(const u8 *in, unsigned int ilen,
if (!bytes || bytes > SHORT_DATA_BITS_MAX)
return -EINVAL;

+                       if (p.olen < bytes)
+                               return -EINVAL;
                        while (bytes-- > 0) {
                                ret = next_bits(&p, &tmp, 8);
                                if (ret)
```

Google VPD

- ChromeOS Vital Product Data (VPD) Binary Blob
- Parsed in linux kernel
- Should usually be a trusted source

Baam

```
--- a/drivers/firmware/google/vpd_decode.c
+++ b/drivers/firmware/google/vpd_decode.c
@@ -69,7 +69,7 @@ int vpd_decode_string(const s32 max_len, const u8 *input_buf, s32 *consumed,
                /* key */
                res = vpd_decode_len(max_len - *consumed, &input_buf[*consumed],
                                     &key_len, &decoded_len);
-               if (res != VPD_OK || *consumed + decoded_len >= max_len)
+               if (res != VPD_OK || *consumed + decoded_len + key_len >= max_len)
                        return VPD_FAIL;

                *consumed += decoded_len;
@@ -79,7 +79,7 @@ int vpd_decode_string(const s32 max_len, const u8 *input_buf, s32 *consumed,
                /* value */
                res = vpd_decode_len(max_len - *consumed, &input_buf[*consumed],
                                     &value_len, &decoded_len);
-               if (res != VPD_OK || *consumed + decoded_len > max_len)
+               if (res != VPD_OK || *consumed + decoded_len + value_len > max_len)
                        return VPD_FAIL;

                *consumed += decoded_len;
```

Next Steps / Future work

- Userspace fuzzing with clang/libfuzzer
- Add more fuzzing targets
- Think about in-kernel IPC

Thanks

- Q & A
- VOTE!!!
- eric.sesterhenn@x41-dsec.de