Fuzzing the Native NTFS Read-Write Driver in the Linux Kernel

Edward Lo, Chiachih Wu

Agenda

- About us
- Motivation
- Linux file system 101
- Challenges to file system fuzzing
- Papora – the efficient file system fuzzer for NTFS
- Evaluation
- Takeaways

About Us

- Edward Lo
  - Security researcher at Amber Group
  - Survey and apply feasible fuzzing technology to blockchain clients
  - Internal auditing on blockchain projects

- Chiachih Wu (@chiachih_wu)
  - Head of web3 security team at Amber Group
  - Blockchain security auditing/researching
  - Blockchain data analytics

Motivation

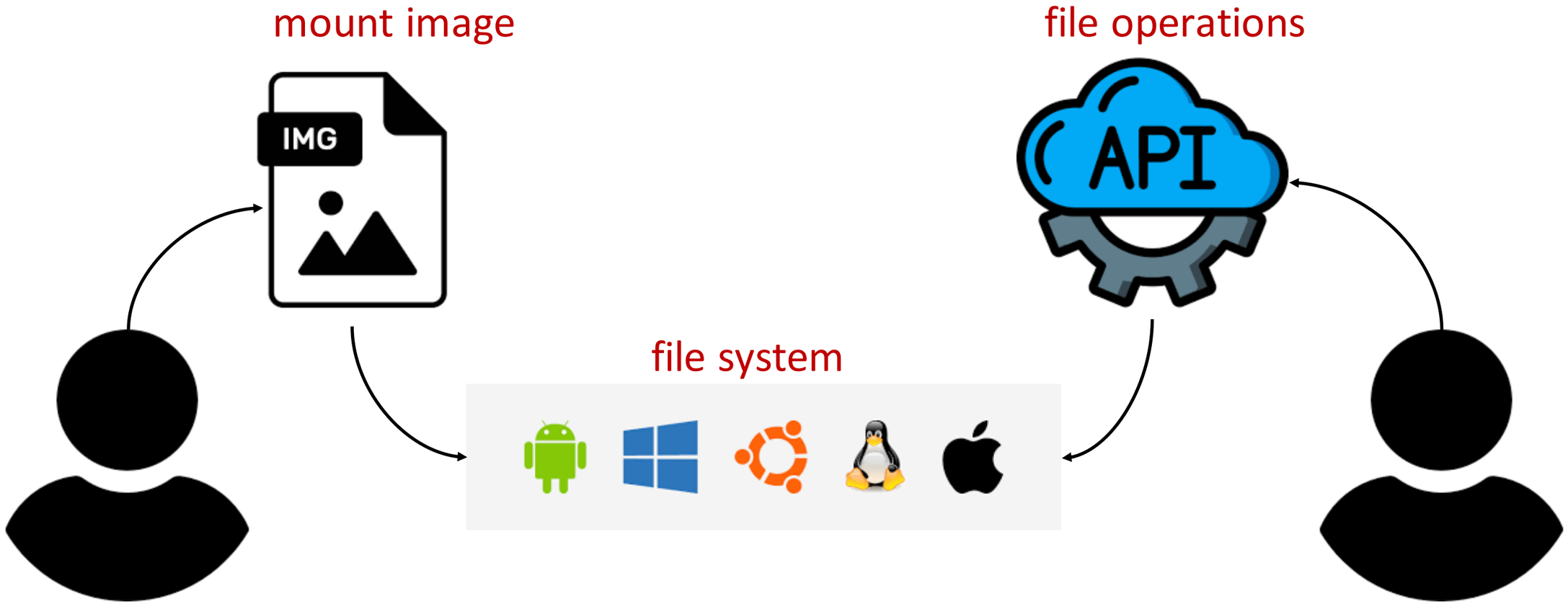
- NTFS3 was firstly upstreamed to Linux kernel in late 2021
- A new file system is complicated enough to have some bugs
- Existing fuzzers cannot efficiently fuzz a new file system

NTFS

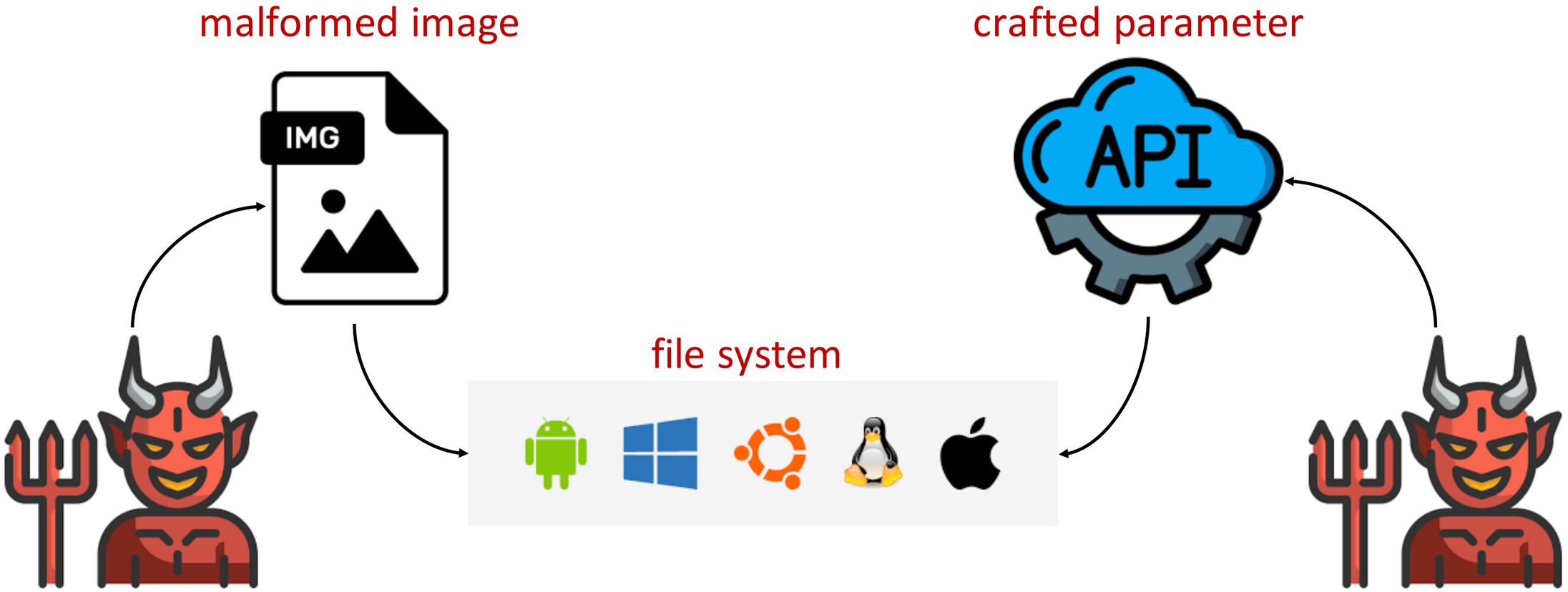
- New Technology File System (NTFS)
  - A proprietary journaling file system developed by Microsoft
  - Default file system of the Windows NT family starting NT 3.1

- There are ways to work with NTFS from Linux
  - NTFS: an old implementation that supports read and limited write on NTFS drive
  - NTFS-3G: a full-featured, R/W FUSE (Filesystem in Userspace) package
  - NTFS3: a fully functional R/W NTFS driver, upstreamed in Linux kernel v5.15

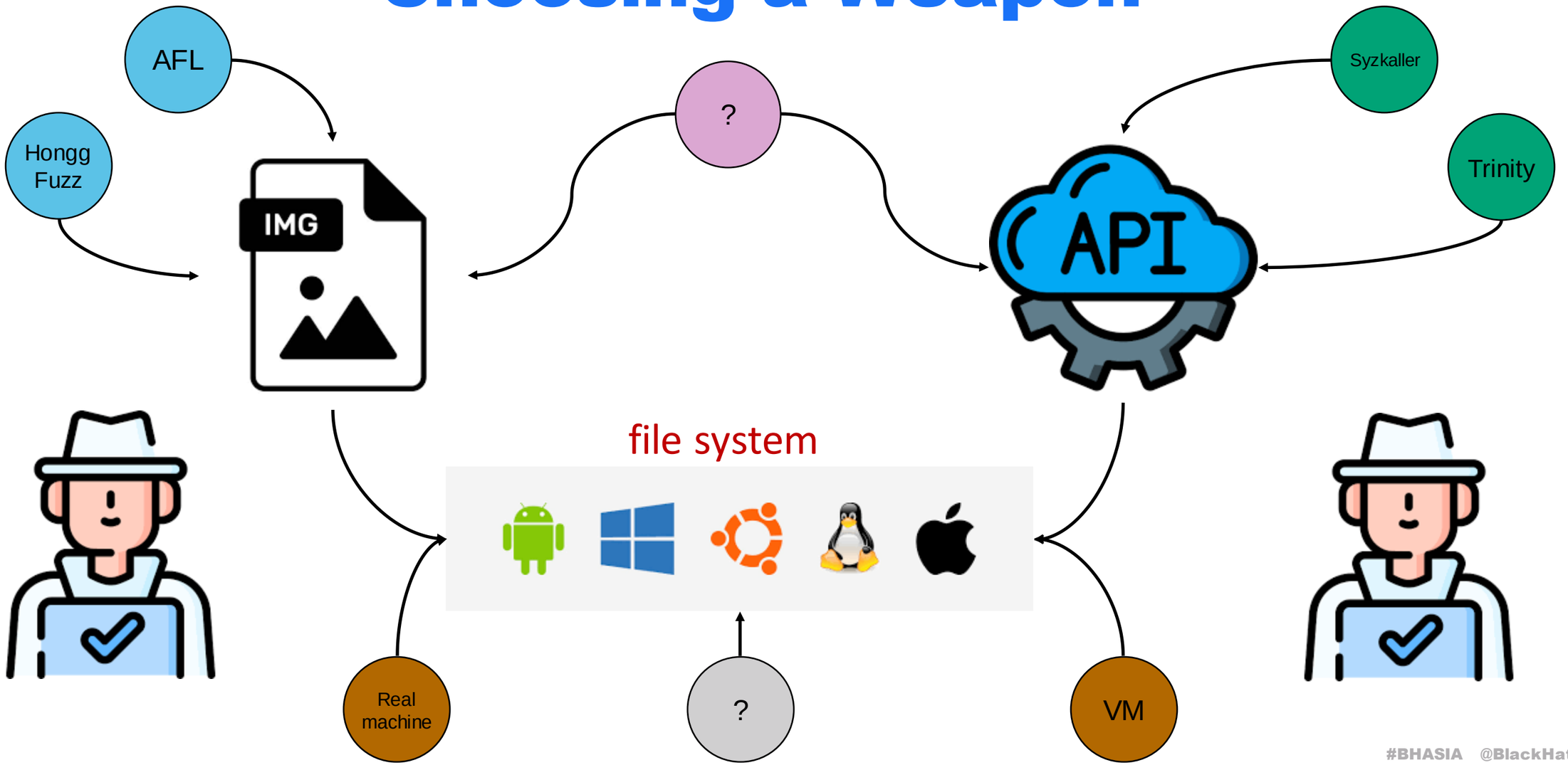
File System



Attack Vectors

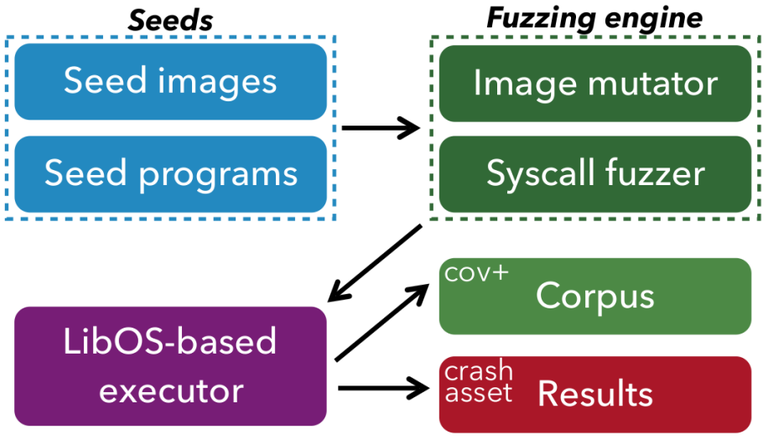


Choosing a Weapon



Janus

- A coverage-driven fuzzer that efficiently and effectively test images and file operations in a joint manner (published in IEEE S&P '19)



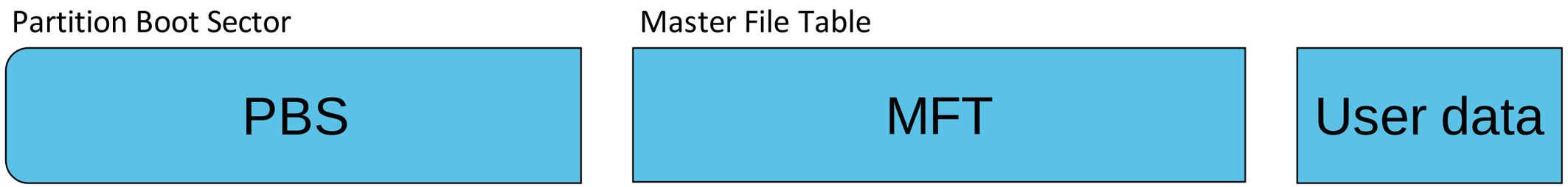
- However, we can't use it for our target file system (NTFS)
  - Need a specific image parser for NTFS (more about it later)
  - The library (Linux kernel library) used by executor was obsolete (v5.3) and inactive at the surveying time
  - KASAN patch integration and modification for the evolving new kernel

Challenges to Image Fuzzing

- Images are large
  - Only metadata matters
  - Mutation on user data is basically a waste of time
- Each file system has its own metadata structure design
  - Need to develop a specific parser for the file system
- Checksums
  - Corrupted after mutation, which could lead to mount fails

Challenges to Image Fuzzing - NTFS

- NTFS image format



Offset	Field	Remark
0x00	jump code	Jump to boot code
0x03	OEM ID	“NTFS “
0x0B	Bytes per sector	
0x0D	Sectors per cluster	
…	…	…
0x01FE	End of sector mark	value = 0xAA55

Entry	File name	Purpose
0	$MFT	Metadata for all files
1	$MFTMirr	Duplicate of the first 4 entries of $MFT
2	$LogFile	Transaction log
3	$Volume	Volume information
…	…	…
26	$Extend\$Reparse	Reparse point data

Challenges to Image Fuzzing - PBS

```
if (memcmp(boot->system_id, "NTFS    ", sizeof("NTFS    ") - 1)) {
        ntfs_err(sb, "Boot's signature is not NTFS.");
        goto out;
}
```

→ OEM ID must equal “NTFS “ (4 spaces)

```
boot_sector_size = ((u32)boot->bytes_per_sector[1] << 8) |
                   boot->bytes_per_sector[0];
if (boot_sector_size < SECTOR_SIZE ||
    !is_power_of_2(boot_sector_size)) {
        ntfs_err(sb, "Invalid bytes per sector %u.", boot_sector_size);
        goto out;
}
```

→ Sector size >= 512 and must be a power of 2

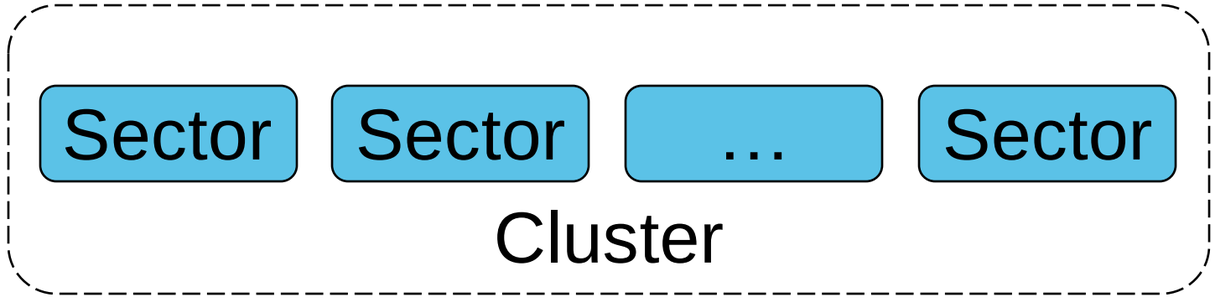
```
sct_per_clst = true_sectors_per_clst(boot);
if ((int)sct_per_clst < 0 || !is_power_of_2(sct_per_clst)) {
        ntfs_err(sb, "Invalid sectors per cluster %u.", sct_per_clst);
        goto out;
}
```

→ Cluster size must be a power of 2

More sanity checks…

Challenges to Image Fuzzing - MFT

- The unit of disk space that NTFS uses is a cluster, which is a collections of sectors



- NTFS applies a concept – fixup, to protect the integrity of some important metadata
  - FILE Records in the $MFT
  - INDX Records in directories and other indexes
  - RCRD Records in the $LogFile…and other critical metadata

Fixup – Write

Offset	Data	Description
0x00	…	Metadata header
0x30	0x12 | 0x34 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00	0x30-0x31: update sequence number 0x32- : update sequence array
0x1F8	0x11 | 0x12 | 0x13 | 0x14 | 0x15 | 0x16 | 0x17 | 0x18	End of sector 1
0x3F8	0x21 | 0x22 | 0x23 | 0x24 | 0x25 | 0x26 | 0x27 | 0x28	End of sector 2
0x5F8	0x31 | 0x32 | 0x33 | 0x34 | 0x35 | 0x36 | 0x37 | 0x38	End of sector 3
…	…	…

Before write

1. Update Sequence Number + 1
2. Copy last 2 bytes of each sector into the update sequence array
3. Write the new USN to the end of each sector
4. Write back to disk

Offset	Data	Description
0x00	…	Metadata header
0x30	0x12 | 0x35 | 0x17 | 0x18 | 0x27 | 0x28 | 0x37 | 0x38	0x30-0x31: update sequence number 0x32- : update sequence array
0x1F8	0x11 | 0x12 | 0x13 | 0x14 | 0x15 | 0x16 | 0x12 | 0x35	End of sector 1
0x3F8	0x21 | 0x22 | 0x23 | 0x24 | 0x25 | 0x26 | 0x12 | 0x35	End of sector 2
0x5F8	0x31 | 0x32 | 0x33 | 0x34 | 0x35 | 0x36 | 0x12 | 0x35	End of sector 3

(Cont'd)

```
u16 fo = le16_to_cpu(rhdr->fix_off);
u16 fn = le16_to_cpu(rhdr->fix_num);

if ((fo & 1) || fo + fn * sizeof(short) > SECTOR_SIZE || !fn-- ||
    fn * SECTOR_SIZE > bytes) {
        return false;
}

/* Get fixup pointer. */
fixup = Add2Ptr(rhdr, fo);

if (*fixup >= 0x7FFF)
        *fixup = 1;
else
        *fixup += 1;

sample = *fixup;

ptr = Add2Ptr(rhdr, SECTOR_SIZE - sizeof(short));

while (fn--) {
        *++fixup = *ptr;
        *ptr = sample;
        ptr += SECTOR_SIZE / sizeof(short);
}
return true;
```

Fixup – Read

Offset	Data	Description
0x00	…	Metadata header
0x30	0x12 | 0x35 | 0x17 | 0x18 | 0x27 | 0x28 | 0x37 | 0x38	0x30-0x31: update sequence number 0x32- : update sequence array
0x1F8	0x11 | 0x12 | 0x13 | 0x14 | 0x15 | 0x16 | 0x12 | 0x35	End of sector 1
0x3F8	0x21 | 0x22 | 0x23 | 0x24 | 0x25 | 0x26 | 0x12 | 0x35	End of sector 2
0x5F8	0x31 | 0x32 | 0x33 | 0x34 | 0x35 | 0x36 | 0x12 | 0x35	End of sector 3
…	…	…

After read

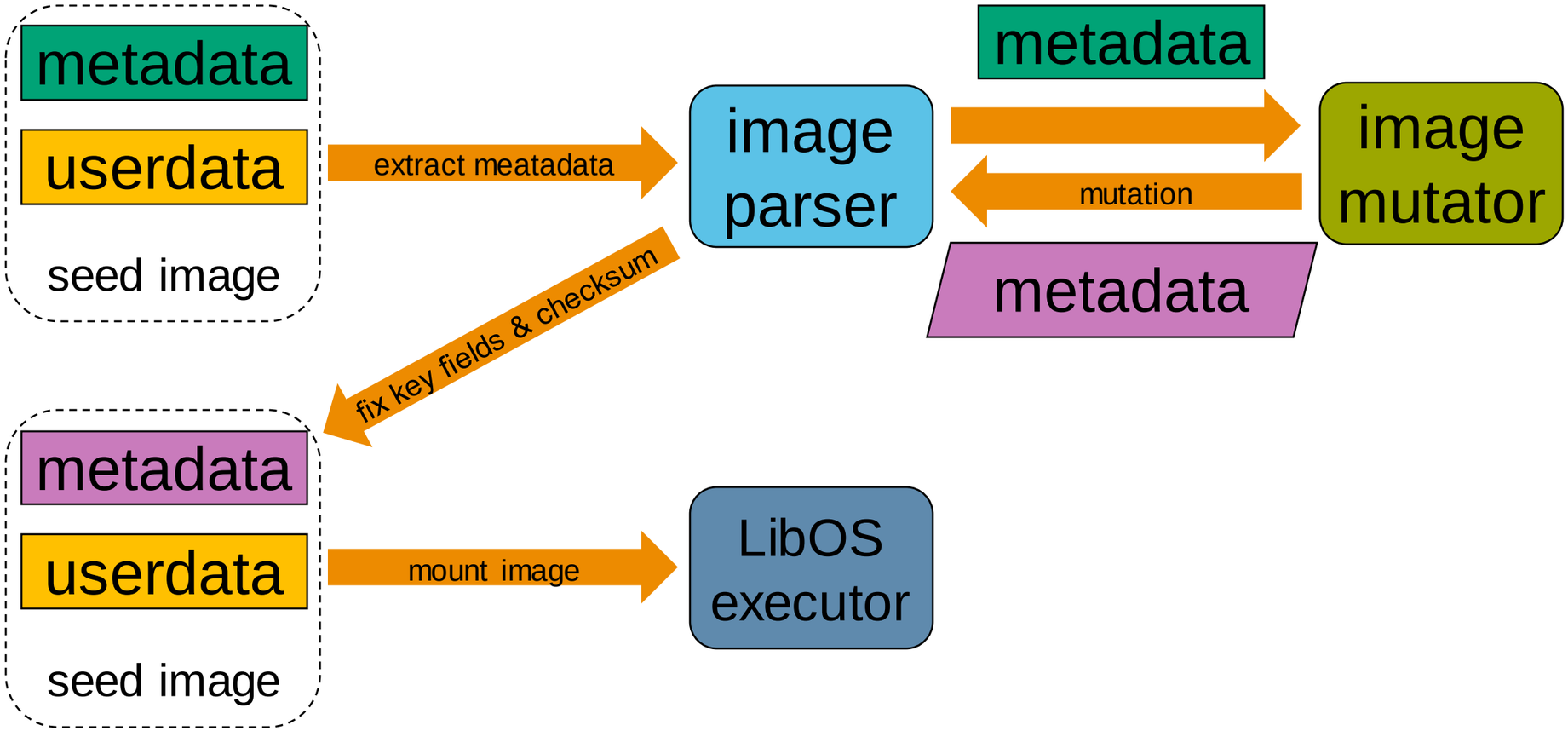
1. Compare the USN against last 2 bytes of each sector, make sure they are the same
2. Check fail could mean a bad sector, disk corruption or system error
3. Copy the corresponding fixup back to the last 2 bytes of each sector

Offset	Data	Description
0x00	…	Metadata header
0x30	0x12 | 0x35 | 0x17 | 0x18 | 0x27 | 0x28 | 0x37 | 0x38	0x30-0x31: update sequence number 0x32- : update sequence array
0x1F8	0x11 | 0x12 | 0x13 | 0x14 | 0x15 | 0x16 | 0x17 | 0x18	End of sector 1
0x3F8	0x21 | 0x22 | 0x23 | 0x24 | 0x25 | 0x26 | 0x27 | 0x28	End of sector 2
0x5F8	0x31 | 0x32 | 0x33 | 0x34 | 0x35 | 0x36 | 0x37 | 0x38	End of sector 3

(Cont'd)

```
fo = le16_to_cpu(rhdr->fix_off);
fn = simple ? ((bytes >> SECTOR_SHIFT) + 1) :
                    le16_to_cpu(rhdr->fix_num);

/* Check errors. */
if ((fo & 1) || fo + fn * sizeof(short) > SECTOR_SIZE || !fn-- ||
    fn * SECTOR_SIZE > bytes) {
        return -EINVAL; /* Native chkntfs returns ok! */
}

/* Get fixup pointer. */
fixup = Add2Ptr(rhdr, fo);
sample = *fixup;
ptr = Add2Ptr(rhdr, SECTOR_SIZE - sizeof(short));
ret = 0;

while (fn--) {
        /* Test current word. */
        if (*ptr != sample) {
                /* Fixup does not match! Is it serious error? */
                ret = -E_NTFS_FIXUP;
        }

        /* Replace fixup. */
        *ptr = *++fixup;
        ptr += SECTOR_SIZE / sizeof(short);
}

return ret;
```

Papora Image Parser

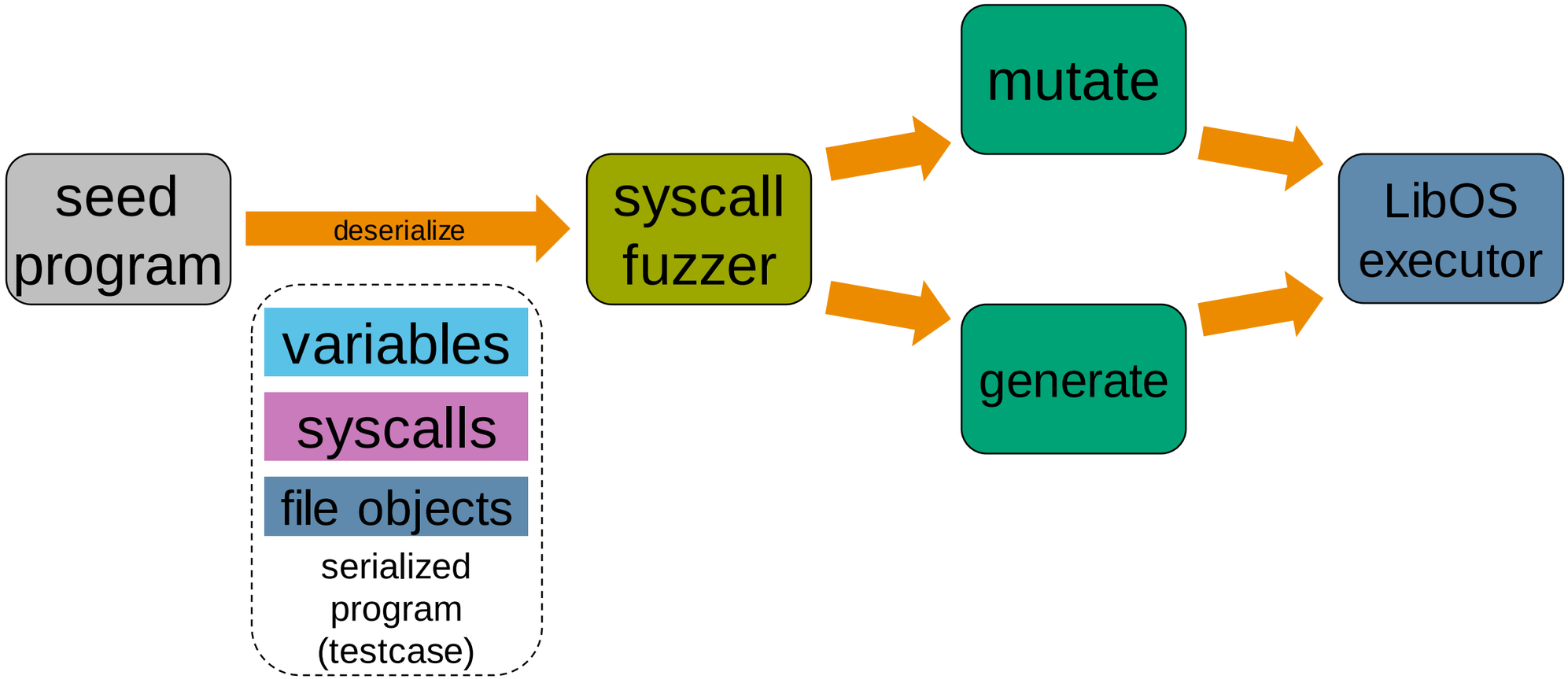


Challenges to Syscall Fuzzing

- What to generate
  - system calls for file operations
- How to mutate
  - The fuzzer should know how to mutate each arguments of the system calls
  - A valid ***fd***, combination of flags, pre-allocated buffers …
- Context awareness
  - The context should be maintained across each system calls

```
int fd = open("papora.seed", …);
read(fd, buf, 256);
close(fd);
```

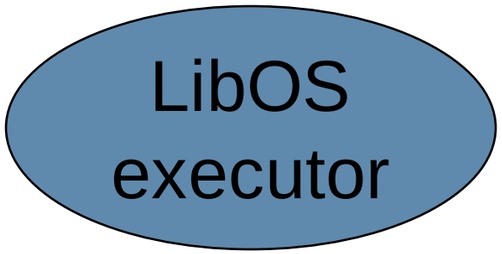
Papora Syscall Fuzzer



Challenges to Executor

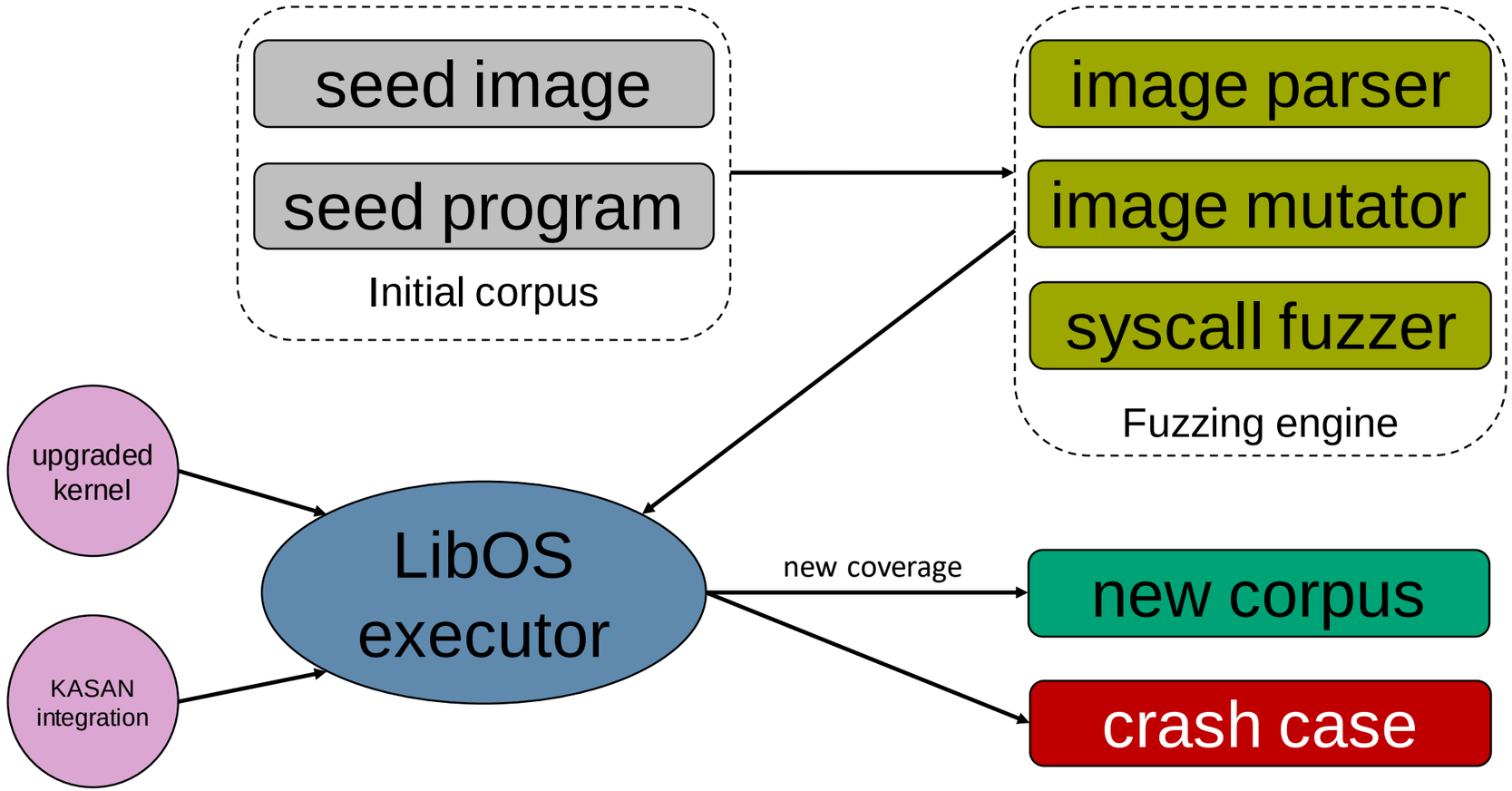
Speed	Fast	Slow
Scalability	Buy more devices ($)	Spawn more VM
Management (reboot / debug / etc)	Hard	Easy
Risk	High (bricked)	Low (out-of-mem?)

(Cont'd)



Pros	Cons
• Fast execution • Easy management (reboot / debug / etc) • Easy to scale • Easy to reproduce (non-aging kernel)	• Since LKL is an arch of Linux, there are some limitations of current implementation, e.g., !MMU / !SMP / etc • Kernel upgrading effort

Papora Workflow



Evaluation

- Run Syzkaller for 1 month with the customized syz-lang description
  - Constrain the system calls to file operations only
  - No interesting outcome ☹
- Run Papora for 3 months intermittently
  - Upgrade LKL whenever new kernel is available (v5.15 ➔ v6.0)
  - Identified 12 issues

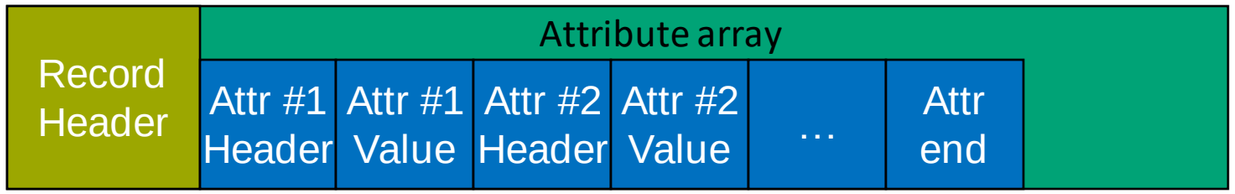
Commit	Bug Type	Root Cause
0b66046	NPD	Sanity check miss
e19c627	OOB Read	Arithmetic overflow
6db6208	OOB Read	Sanity check miss
2681631	NPD	Sanity check miss
c1ca8ef	NPD	Implementation flaw
4f1dc7d	Heap Corruption	Sanity check miss
bfcdbae	OOB Read	Sanity check miss
******	OOB Read	Sanity check miss
******	Heap Corruption	Type confusion
******	OOB Read	Sanity check miss
4d42ecd	OOB Read	Sanity check miss
54e4570	OOB Write	Sanity check miss

******: not upstreamed
Type 1: Triggered by image mount
Type 2: Triggered by image mount + file operations

Case Study – Type 1

PBS

MFT record



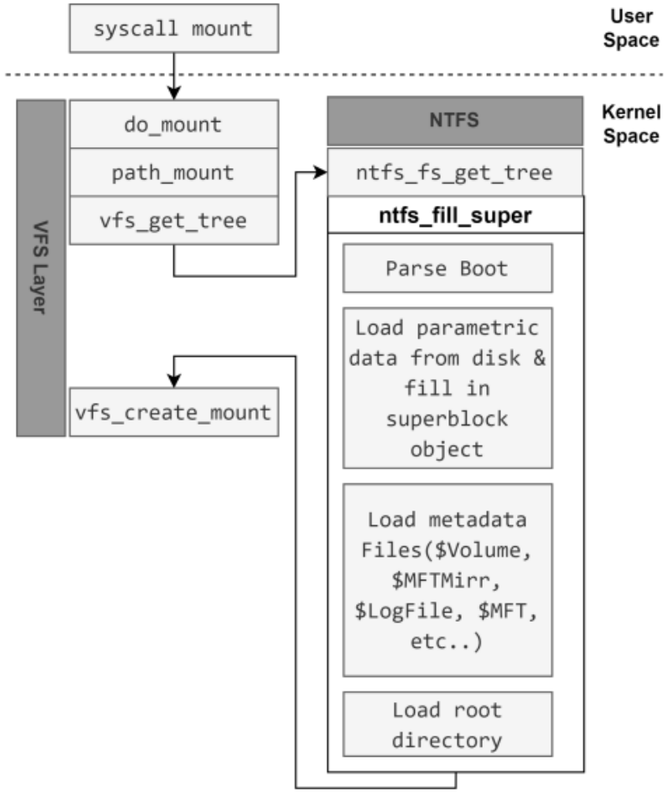
Offset	Field	Remark
0x00	jump code	Jump to boot code
0x03	OEM ID	“NTFS “
0x0B	Bytes per sector	
0x0D	Sectors per cluster	
…	…	…
0x40	MFT entry size	

A positive value denotes the number of clusters of a MFT entry.
➔ 0x1 ➔ 1 cluster (bytes per sector x sectors per cluster)

A negative value denotes the number of bytes of a MFT entry, in which case the size is 2 to the power of the absolute value
➔ 0xF6 ➔ -10 → 2^{10} = 1024

(Cont'd)

```
BUG: kernel NULL pointer dereference, address: 0000000000000158
...
Call Trace:
 <TASK>
 ? ntfs_alloc_inode+0x1a/0x60
 attr_load_runs_vcn+0x2b/0xa0
 mi_read+0xbb/0x250
 ntfs_iget5+0x114/0xd90
 ntfs_fill_super+0x588/0x11b0
 ? put_ntfs+0x130/0x130
 ? snprintf+0x49/0x70
 ? put_ntfs+0x130/0x130
 get_tree_bdev+0x16a/0x260
 vfs_get_tree+0x20/0xb0
 path_mount+0x2dc/0x9b0
 do_mount+0x74/0x90
 __x64_sys_mount+0x89/0xd0
 do_syscall_64+0x3b/0x90
 entry_SYSCALL_64_after_hwframe+0x63/0xcd
```



(Cont'd)

```
static int ntfs_fill_super(struct super_block *sb, struct fs_context *fc)
{
  ...

  /* Parse boot. */
  err = ntfs_init_from_boot(sb, bdev_logical_block_size(bdev),
                            bdev_nr_bytes(bdev));
  if (err)
        goto out;
```

```
static int ntfs_init_from_boot(struct super_block *sb,
                u32 sector_size, u64 dev_size)
{
  ...
  sbi->record_size = record_size = boot->record_size < 0
    ? 1 << (-boot->record_size)
    : (u32)boot->record_size << sbi->cluster_bits;

  if (record_size > MAXIMUM_BYTES_PER_MFT)
        goto out;

  sbi->record_bits = blksize_bits(record_size);
```

```
/* assumes size > 256 */
static inline unsigned int blksize_bits(unsigned int size)
{
        unsigned int bits = 8;
        do {
                bits++;
                size >>= 1;
        } while (size > 256);
        return bits;
}
```

The record_size is derived by the formula. However, the corresponding record_bits is calculated with the assumption that it's larger than 256

Say if we have a boot->record_size = 0xF8 = -8

➔sbi->record_size = 2^8 = 256

➔sbi->record_bits = 9

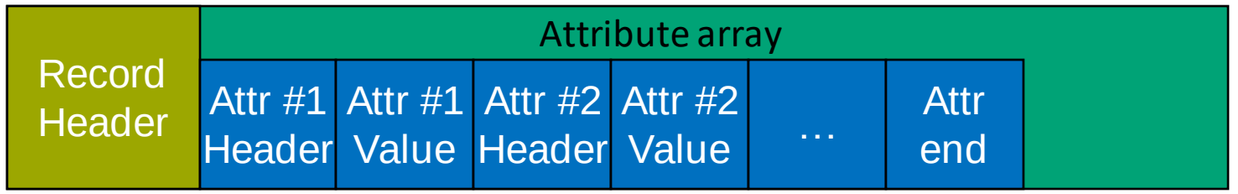
So we have a mismatch here, which will lead to a NPD issue

Patch

```
diff --git a/fs/ntfs3/super.c b/fs/ntfs3/super.c
index d72a27abf1c83..af9b7947df64e 100644
--- a/fs/ntfs3/super.c
+++ b/fs/ntfs3/super.c
@@ -814,7 +814,7 @@ static int ntfs_init_from_boot(struct super_block *sb, u32 sector_size,
                                                : (u32)boot->record_size
                                                        << sbi->cluster_bits;

-       if (record_size > MAXIMUM_BYTES_PER_MFT)
+       if (record_size > MAXIMUM_BYTES_PER_MFT || record_size < SECTOR_SIZE)
                goto out;

        sbi->record_bits = blksize_bits(record_size);
```

Case Study – Type 2 (CVE-2022-48423)

MFT record



Entry	File name	Purpose
0	$MFT	Metadata for all files
1	$MFTMirr	Duplicate of the first 4 enties of $MFT
2	$LogFile	Transation log
3	$Volume	Volume information
…	…	…
26	$Extend\$Reparse	Reparse point data

```
struct ATTRIB {
        enum ATTR_TYPE type;    // 0x00: The type of this attribute.
        __le32 size;            // 0x04: The size of this attribute.
        u8 non_res;             // 0x08: Is this attribute non-resident?
        u8 name_len;            // 0x09: This attribute name length.
        __le16 name_off;        // 0x0A: Offset to the attribute name.
        __le16 flags;           // 0x0C: See ATTR_FLAG_XXX.
        __le16 id;              // 0x0E: Unique id (per record).

        union {
                struct ATTR_RESIDENT res;     // 0x10
                struct ATTR_NONRESIDENT nres; // 0x10
        };
};
```

(Cont'd)

BUG: KASAN: slab-out-of-bounds in ni_create_attr_list+0x1e1/0x850
Write of size 426 at addr ffff88800632f2b2 by task exp/255
...
Call Trace:
<TASK>
dump_stack_lvl+0x49/0x63
...
kasan_report+0xa7/0x130
...
memcpy+0x3c/0x70
ni_create_attr_list+0x1e1/0x850
...
ni_ins_attr_ext+0x52c/0x5c0
...
ni_insert_resident+0xc0/0x1c0
...
ntfs_setxattr+0x114/0x5c0
...
vfs_setxattr+0x128/0x300
do_setxattr+0xb8/0x170
setxattr+0x126/0x140
...
__x64_sys_setxattr+0x6d/0x80
...

Allocated by task 255:
kasan_save_stack+0x26/0x50
__kasan_kmalloc+0x88/0xb0
__kmalloc+0x192/0x320
ni_create_attr_list+0x11e/0x850
ni_ins_attr_ext+0x52c/0x5c0
ni_insert_attr+0x1ba/0x420
ni_insert_resident+0xc0/0x1c0
ntfs_set_ea+0x6bf/0xb30
ntfs_setxattr+0x114/0x5c0
__vfs_setxattr+0xda/0x120
__vfs_setxattr_noperm+0x93/0x300
__vfs_setxattr_locked+0x141/0x160
vfs_setxattr+0x128/0x300
do_setxattr+0xb8/0x170
setxattr+0x126/0x140
path_setxattr+0x164/0x180
__x64_sys_setxattr+0x6d/0x80
do_syscall_64+0x3b/0x90
entry_SYSCALL_64_after_hwframe+0x63/0xcd

The buggy address belongs to the object at ffff88800632f000
which belongs to the cache kmalloc-1k of size 1024

(Cont'd)

```
int ni_create_attr_list(struct ntfs_inode *ni)
{
  ...
  le = kmalloc(al_aligned(rs), GFP_NOFS);
  if (!le) {
        err = -ENOMEM;
        goto out;
  }
  ...
  for (; (attr = mi_enum_attr(&ni->mi, attr)); le = Add2Ptr(le, sz)) {
    sz = le_size(attr->name_len);
    le->type = attr->type;
    le->size = cpu_to_le16(sz);
    le->name_len = attr->name_len;
    le->name_off = offsetof(struct ATTR_LIST_ENTRY, name);
    le->vcn = 0;
    if (le != ni->attr_list.le)
        le->ref = ni->attr_list.le->ref;
    le->id = attr->id;

    if (attr->name_len)
        memcpy(le->name, attr_name(attr),
               sizeof(short) * attr->name_len);
    ...
  }
```

```
static inline size_t al_aligned(size_t size)
{
        return (size + 1023) & ~(size_t)1023;
}
```

```
struct ATTRIB *mi_enum_attr(struct mft_inode *mi, struct ATTRIB *attr)
{
  ...
  asize = le32_to_cpu(attr->size);
  ...
  /* Check size of attribute. */
  if (!attr->non_res) {
    if (asize < SIZEOF_RESIDENT)
        return NULL;

    t16 = le16_to_cpu(attr->res.data_off);

    if (t16 > asize)
        return NULL;

    t32 = le32_to_cpu(attr->res.data_size);
    if (t16 + t32 > asize)
        return NULL;

    return attr;
  }

  /* Check some nonresident fields. */
  if (attr->name_len &&
    le16_to_cpu(attr->name_off) + sizeof(short) * attr->name_len >
      le16_to_cpu(attr->nres.run_off)) {
    return NULL;
  }
```

Patch

```
diff --git a/fs/ntfs3/record.c b/fs/ntfs3/record.c
index 66eb11e0965ef..a952cd7aa7a4b 100644
--- a/fs/ntfs3/record.c
+++ b/fs/ntfs3/record.c
@@ -265,6 +265,11 @@ struct ATTRIB *mi_enum_attr(struct mft_inode *mi, struct ATTRIB *attr)
                if (t16 + t32 > asize)
                        return NULL;

+               if (attr->name_len &&
+                   le16_to_cpu(attr->name_off) + sizeof(short) * attr->name_len > t16) {
+                       return NULL;
+               }
+
                return attr;
        }
```

Black Hat Sound Bytes

- Complicated and hard-to-fuzz software are good targets for security researchers
- File system maintainers should pay more attention on metadata integrity
- Users should be cautious on mounting an disk image