Lost in the Loader

The many faces of the PE File Format

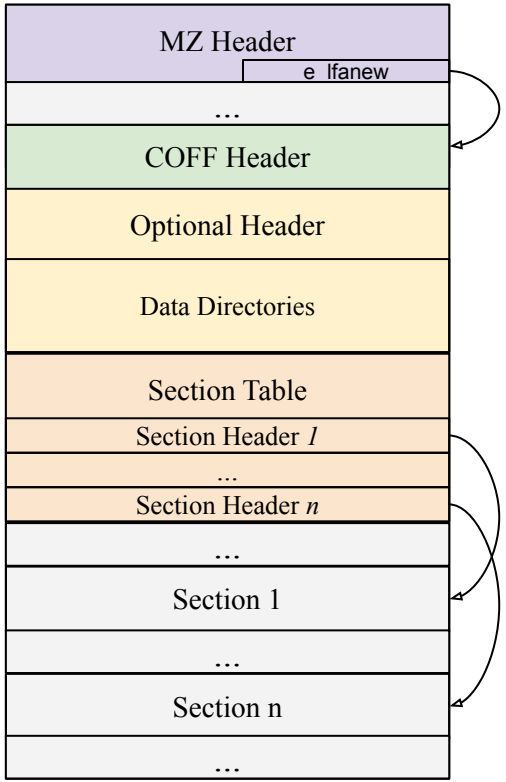
Dario Nisi (EURECOM)
Mariano Graziano (Cisco Talos)
Yanick Fratantonio (Cisco Talos)
Davide Balzarotti (EURECOM)

The PE File Format

- Standard Format for Programs in Windows
- Based on MS-DOS MZ Format and COFF
- Defines the “headers” Structure

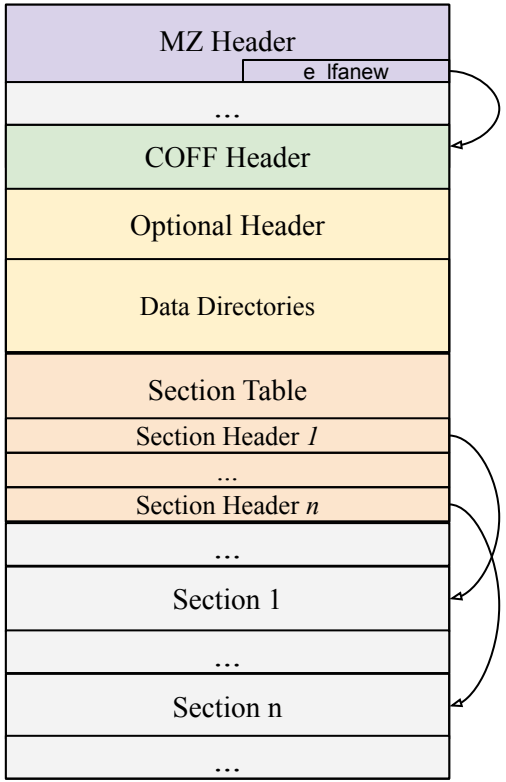
PE Headers

- MZ Header
  - Points to COFF Header
- COFF Header
  - Generic Fields (ISA, Executable/DLL, ...)
- “Optional” Header
  - Not really optional for Executables
  - Entry Point
  - Section/File Alignments
  - Base Address
  - Data Directories



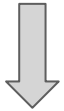
Section Table

- Define the Program's "Memory Image"
- Each Section has
  - Address/Size in Memory
  - Offset/Size in the File
  - Permission level (RWX)
- In theory, up to 2^{16} Sections
  - NumberOfSections is 16 bits



The Subtle Problem of the PE Ecosystem

No reference implementation



Reimplementation is the de facto rule

Not Comprehensive Specifications

Room for Implementation Choices

Discrepancies

Implications of PE discrepancies

Shane Huntley

- Examples:
  - Programs running under Windows 7 but not Windows 10
    - Potential evasion of dynamic analysis sandboxes
  - Reverse engineer tools give wrong mapping
    - Static analysis tools may get confused!

- Problematic for malware analysis & detection
  - Static/Dynamic analysis evasion
  - Antivirus signature bypass

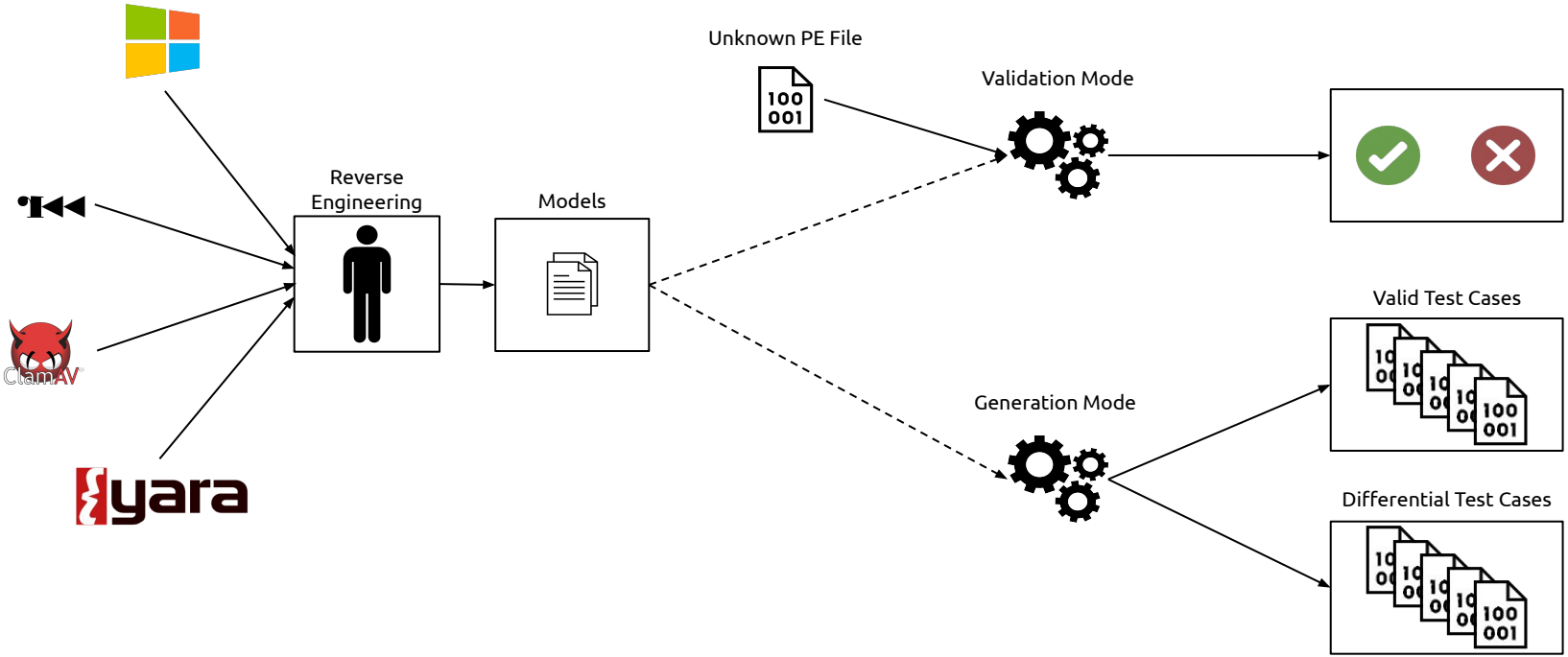
Examples from Previous Work

- Petsios et al. @S&P 2017, Kim et al. @CCS 2017
  - Showed how malware can bypass AVs by forging PE headers
- Corkami PE Collection by A. Albertini
  - Some samples execute only on some Windows Version
- Making a Multi-Windows PE in PoC||GTFO 0x01
  - Executables with different behaviors exploiting relocation discrepancies
- ELF Crafting @r2con2019
  - 1-Byte ELF Parser Breaker which Linux recognizes as valid

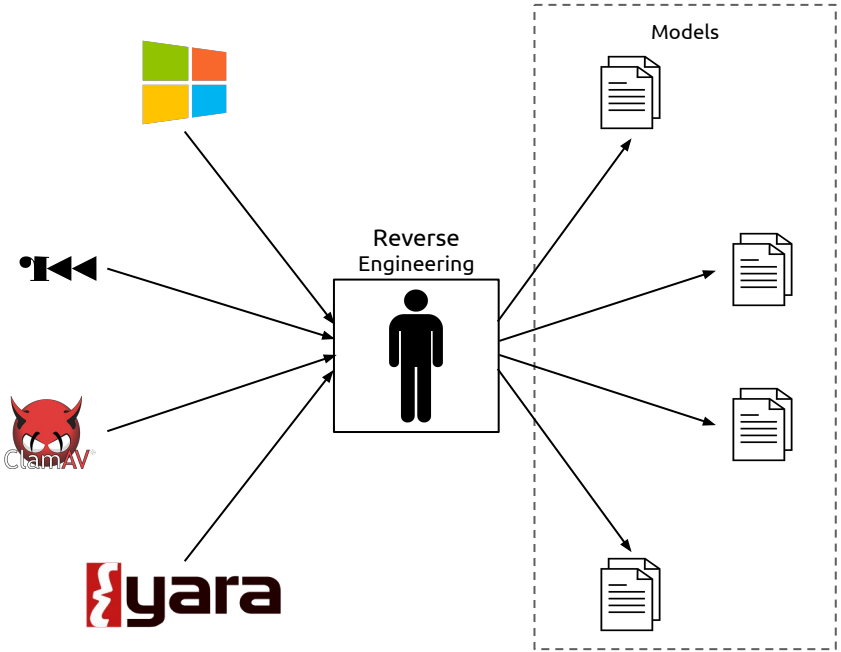
Only single edge cases so far, no systematic study!

Systematic Approach to find Discrepancies

The Big Picture



Constraints Modelling



Modelling Phase

- Collect knowledge on the operations performed by parsing routines
- Three types of operations
  - Structural Checks (mandatory headers, magic numbers, …)
  - Compliance Checks (supported ISA, supported OS versions, …)
  - Memory Mapping (generates the memory image)

All these operations can be modelled as constraints on the input!

Language for Modelling Constraints

- Types of Statements
  - Input Declaration

INPUT statements

Original Program

```
void *inputFile = SomeReadFileFunction();
```

Model

```
INPUT inputFile 2048
```

Language for Modelling Constraints

- Types of Statements
  - Input Declaration
  - Symbol definition

Symbol Definition

Original Program

```
void *inputFile = SomeReadFileFunction();

IMAGE_MZ_HEADER *mzHeader = &inputFile;
```

Model

```
INPUT inputFile 2048

P: mzHeader <- inputFile[0, sizeof
IMAGE_MZ_HEADER] as IMAGE_MZ_HEADER
```

Language for Modelling Constraints

- Types of Statements
  - Input Declaration
  - Symbol definition
  - Predicates

Language for Modelling Constraints

- Types of Statements
  - Input Declaration
  - Symbol definition
  - Predicates
    - Terminal/Non-terminal Predicates

(Terminal) Predicates

Original Program

```
void *inputFile = SomeReadFileFunction();

IMAGE_MZ_HEADER *mzHeader = &inputFile;


if (mzHeader->Magic[0] != "M" ||
    mzHeader->Magic[1] != "Z") {
  return FALSE;
}
```

Model

```
INPUT inputFile 2048

P: mzHeader <- inputFile[0, sizeof
IMAGE_MZ_HEADER] as IMAGE_MZ_HEADER

V1: EQ mzHeader.magic[0] "M" term
V2: EQ mzHeader.magic[1] "Z" term
```

Language for Modelling Constraints

- Types of Statements
  - Input Declaration
  - Symbol definition
  - Predicates
    - Terminal/Non-terminal Predicates
  - Conditional Statements

Conditional Statements

Original Program

```
if (optHeader->SectionAlignment > 0x1000) {

  ...
  if (coff->NumberOfSections == 0)
    return FALSE:

  ...

}
```

Model

```
V1: UGE optHeader->SectionAlignment 0x1000


V2(V1): NEq coff->NumberOfSections 0 term
```

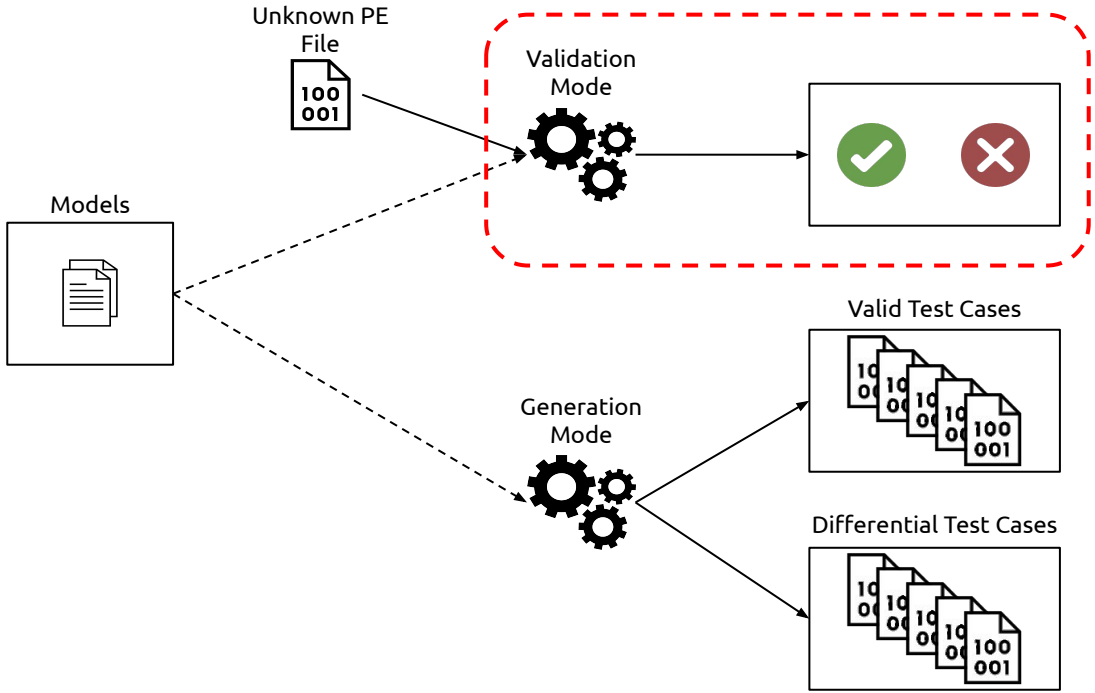
Language for Modelling Constraints

- Types of Statements
  - Input Declaration
  - Symbol definition
  - Predicates
    - Terminal/Non-terminal Predicates
  - Conditional Statements
  - Loops

Language for Modelling Constraints

- Types of Statements
  - Input Declaration
  - Symbol definition
  - Predicates
    - Terminal/Non-terminal Predicates
  - Conditional Statements
  - Loops
  - C-like structured types declaration

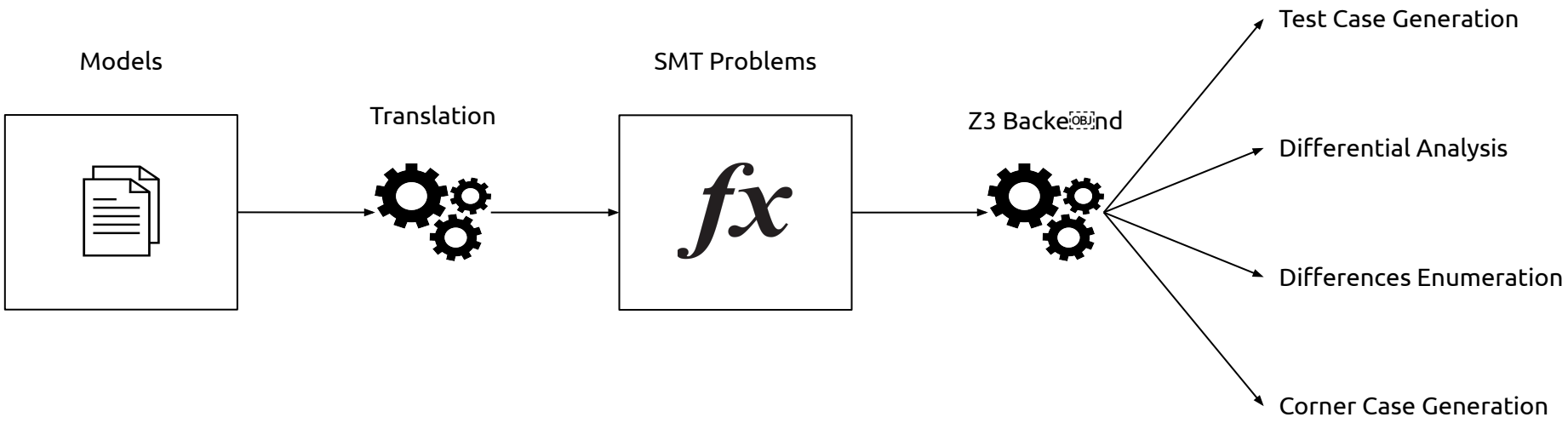
Analysis Framework



Validation Mode

- Determine whether an input file is a valid PE according to a model
- “Procedural interpretation” of the model
  - Evaluate each statement using the data from the input file
- The file is valid if all terminal statements are true, invalid otherwise

Generation Mode



Model ⇔ SMT Equivalence

- Input Declaration creates unconstrained BitVector
- Symbol/Predicate becomes mathematical formulas
- Loops are unrolled up to a certain number of iterations
- Structured types are translated into offsets/slices in the BitVector
- Terminal Predicates are combined (logic conjunction) to create the statement to assert

The solutions of the SMT problem are valid PE files according to the model

Differential Test Case Generation

How to generate a file valid for one model but invalid for second one?

- Solve the $P_1 \wedge \neg P_2$!

- Bonus: the analysis framework tells the broken constraints in P_2

Differences Enumeration

How to find *all* the differences between two models?

- Euristic iterative approach
- For each iteration, solve an SMT problem that asserts some of the previously negated constraints in P_2

Corner Case Generation

How to create many samples that are structurally different?

- Iterative approach that leverages non-terminal statements
- For each iteration, try asserting a different combination of non-terminal statements
  - 2^n iterations (with *n* non-terminal statements)

Customizing Generated Samples

How to fine tune the sections' content in the generated samples?

- Add constraints that section content as a constraint of the model
- We can create samples that execute custom code!

Results & Findings

Modelled Software

- Windows Program Loader (XP, 7, 10)
  - Both Kernel-space and User-space
- ClamAV
  - PE format-specific signature engine
- Radare2
  - PE memory mapping
- Yara
  - PE module

Windows vs Windows

- We actually found discrepancies!!
  - Win7 doesn't accept executables with *ImageBase = 0*
  - Win7 and 10 check *SizeOfHeaders* under specific conditions, XP does not
  - Win7 and XP accept relocation types that 10 does not
  - Win7 and 10 discard binaries with entry point within the header
  - Win7 does not load binaries whose *SizeOfImage* is smaller than the offset of the last byte of the section table

Windows vs. ClamAV

- We actually found discrepancies!!
  - Number of sections (ClamAV: max 96, Windows: max 65k)
  - SizeOfOptionalHeader (ClamAV: ≥ 92, Windows: no constraints)
  - Section Virtual Address (ClamAV: must always be aligned, Windows: under certain conditions not necessarily aligned)

- Acknowledged by developers as bugs. Fixes coming soon.

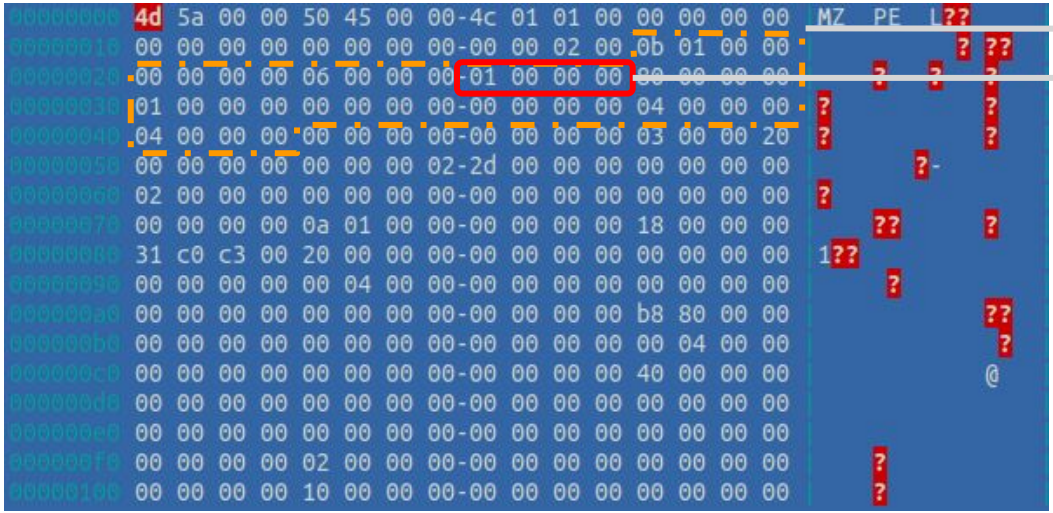
Memory Mapping Discrepancies

- The Source of All Evil: handling *SectionAlignment* < Page Size (4KB in Intel)
  - Windows: maps the file in memory as is, regardless of the section table
  - Radare2, yara, ClamAV: infer mapping from section table

- We could not find any documentation about this behavior

Notable Test Case

Notable Test Case



Notable Test Case

What Windows loads in memory

```
kd> dd 10000
00010000  00005a4d 00004550 0001014c 00000000
00010010  00000000 00000000 00020000 0000010b
00010020  00000000 00000006 00000001 00000080
00010030  00000001 00000000 00000000 00000004
00010040  00000004 00000000 00000000 20000003
00010050  00000000 02000000 0000002d 00000000
00010060  00000002 00000000 00000000 00000000
00010070  00000000 0000010a 00000000 00000018
kd> dd 10080
00010080  00c3c031 00000020 00000000 00000000
00010090  00000000 00000400 00000000 00000000
000100a0  00000000 00000000 00000000 000080b8
000100b0  00000000 00000000 00000000 00000400
000100c0  00000000 00000000 00000000 00000040
000100d0  00000000 00000000 00000000 00000000
000100e0  00000000 00000000 00000000 00000000
000100f0  00000000 00000002 00000000 00000000
kd> g 10080
001b:00010080 31c0            xor     eax,eax
```

What radare thinks Windows loads in memory

```
[0x00010000 [Xadvc]0 0% 896 t]> xc @ oeax
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F
0x00010000  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010010  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010020  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010030  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010040  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010050  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010060  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010070  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010080  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010090  ffff ffff ffff ffff ffff ffff ffff ffff
0x000100a0  ffff ffff ffff ffff ffff ffff ffff ffff
0x000100b0  ffff ffff ffff ffff ffff ffff ffff ffff
0x000100c0  ffff ffff ffff ffff ffff ffff ffff ffff
0x000100d0  ffff ffff ffff ffff ffff ffff ffff ffff
0x000100e0  ffff ffff ffff ffff ffff ffff ffff ffff
0x000100f0  ffff ffff ffff ffff ffff ffff ffff ffff
0x00010100  ffff ffff ffff ffff ffff ffff ffff ffff
```

Evidence of Use in the Wild

Malware Hunting Campaign

- VirusTotal LiveHunt
- Yara rules matching discovered discrepancy sources
  - Except the relocation handling (due to limitation of the Yara language)
- 7th to 19th October 2020
- ~5M samples scanned (est. from the average number of weekly submissions reported by VirusTotal)

Malware Hunt Campaign Results

- 467 samples reported
- Each rule matched at least once
- 301 have SectionAlignment < 4096 (may trigger different mappings)
- 77 had more than 96 sections (invalid for ClamAV)
  - Some had exactly 97 sections 🤔

Conclusions & Takeaways

Takeaways

- No “one way” to handle PE format: different versions of Windows handle it differently
- Need for clearer (formal?) specifications and reference implementations (takes time)
- Security tools should model more than one version of the Windows loader
- Our language and framework ease reverse engineering efforts to build these models
- Framework and models available at https://github.com/eurecom-s3/loaders_modeling

Questions?