



Mastering YARA Rules: A Step-by-Step Guide

Learn to Hunt Malware with Precision

Sara Khanchi
INCS 745 – NYIT

Outline

- Introduction to YARA
- Anatomy of a YARA Rule
- Writing YARA Rule
 - Using Wildcards and Placeholders
 - Matching Multiple Strings with Conditions
 - Matching Assembly Code and Hex Patterns
 - Regular Expressions in YARA
- Conclusion

What is YARA? Why Use It?

- YARA = Yet Another Recursive Acronym
- “The pattern matching swiss knife for malware researchers (and everyone else)”
- Used in antivirus engines, threat hunting, incident response
- Hosted on GitGub <http://plusvic.github.io/yara/>
- **Pattern matching:**
 - strings (ASCII, UCS-2)
 - regular expressions
 - binary patterns (hex strings)
- **Classification:**
 - on input: combination of strings
 - on output: tags, metadata



Introduction

- What YARA is **NOT**
 - Not a virus scanner
 - Not a correlation engine
 - Not a bayesian classifier
 - No artificial intelligence (AI) involved

Introduction

- What YARA **IS**
 - A “better grep”
- Use cases:
 - Finding interesting entries on pastebin.com ...
 - Triage data
 - Preprocess files to direct reverse engineering efforts
- Integrate it into your projects:
 - C library
 - Python bindings <https://github.com/plusvic/yara/tree/master/yara-python>
 - Ruby bindings <https://github.com/SpiderLabs/yara-ruby>

Introduction

- YARA rules are supported by security products and services
 - FireEye appliances
 - Fidelis XPS
 - RSA ECAT
 - Volatility
 - ThreadConnect threat intelligence exchange
 - VirusTotal Intelligence
 - ...

Writing YARA Rules

Anatomy of a Simple YARA Rule

```
rule my_example : tag1 tag2 tag3
```

```
{
```

```
  meta:
```

```
    description = "This is just an example"
```

```
    thread_level = 3
```

```
    in_the_wild = true
```

```
  strings:
```

```
    $a = { 6A 40 68 00 30 00 00 6A 14 8D 91 }
```

```
    $b = /[0-9a-f]{32}/
```

```
    $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
```

```
  condition:
```

```
    $a or ( $b and $c )
```

```
}
```


Yara Options

```
$ yara
usage: yara [OPTION]... [RULEFILE]... FILE
options:
  -t <tag>                print rules tagged as <tag> and ignore the
                           rest. Can be used more than once.
  -i <identifier>          print rules named <identifier> and ignore the
                           rest. Can be used more than once.
  -n                        print only not satisfied rules (negate).
  -g                        print tags.
  -m                        print metadata.
  -s                        print matching strings.
  -d <identifier>=<value>  define external variable.
  -r                        recursively search directories.
  -f                        fast matching mode.
  -v                        show version information.
```

Yara Rule-Editor to Use

- Editors you can use
 - vim (with simple syntax highlighting)
 - gvim (with GUI and syntax highlighting)
 - Emacs
 - gedit
 - Visual Studio
 - Notepad++

Hello World!

Your First YARA Rule

Your First Yara Rule

- Create a file named “hello.yara” with the following contents:

```
rule Hello_World
{
    condition: true
}
```

- Now let the computer greet you:

```
$ yara hello.yara /yara/malware/somefile.txt
```

Note: “/yara/malware/somefile.txt” is one file in your system

Another Yara Rule

- Change the “hello.yara” with the following contents:

```
rule HelloWorld
{
    strings:
        $a = "Hello, world!"
    condition:
        $a
}
```

- Now let the computer greet you:

```
$ yara hello.yara /yara/malware/somefile.txt
```

Note: “/yara/malware/somefile.txt” is one file in your system

Yara Rule=Input from User

- Create a file named “greeting.yara” with the following contents:

```
rule GoodMorning
{
    condition:
        hour < 12 and hour >= 4
}
```

- Create another rule for the GoodEvening greetings

- Now pass different values for “hour” to the rule set:

```
$ yara -d hour=8 greeting.yara /yara/malware/somefile.txt
GoodMorning /yara/files/somefile.txt
```

```
$ yara -d hour=20 greeting.yara /yara/malware/somefile.txt
GoodEvening /yara/files/somefile.txt
```

Yara Rule-Counting Strings

- Use # to count string matches:

```
rule ManyOccurrences
{
    strings:
        $a = "malicious"
    condition:
        #a > 5
}
```

```
$ yara rules.yara /yara/malware/somefile.txt
```

Detecting File Types

Detecting File Types=Executables

- A simple specification for PE files (Windows executable)
- Task: To find any files in Portable Executable (“PE”) format
- Simple specification: File must contain the strings “MZ” and “PE”

| | | | |
|----------|-------------------------|-------------------------|------------------|
| 00000000 | 4d 5a 90 00 03 00 00 00 | 04 00 00 00 ff ff 00 00 | MZ..... |
| 00000010 | b8 00 00 00 00 00 00 00 | 40 00 00 00 00 00 00 00 |@..... |
| 00000020 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 00000030 | 00 00 00 00 00 00 00 00 | 00 00 00 00 c8 00 00 00 | |
| 00000040 | 0e 1f ba 0e 00 b4 09 cd | 21 b8 01 4c cd 21 54 68 |!...L.!Th |
| 00000050 | 69 73 20 70 72 6f 67 72 | 61 6d 20 63 61 6e 6e 6f | is program canno |
| 00000060 | 74 20 62 65 20 72 75 6e | 20 69 6e 20 44 4f 53 20 | t be run in DOS |
| 00000070 | 6d 6f 64 65 2e 0d 0d 0a | 24 00 00 00 00 00 00 00 | mode....\$..... |
| 00000080 | 65 cd 43 c7 21 ac 2d 94 | 21 ac 2d 94 21 ac 2d 94 | e.C.!.-.!.-.!.- |
| 00000090 | 21 ac 2c 94 25 ac 2d 94 | e2 a3 70 94 24 ac 2d 94 | !.,.%.-...p.\$.- |
| 000000a0 | c9 b3 26 94 23 ac 2d 94 | 52 69 63 68 21 ac 2d 94 | ..&.#.-.Rich!.- |
| 000000b0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | |
| 000000c0 | 00 00 00 00 00 00 00 00 | 50 45 00 00 4c 01 03 00 |PE..L... |

Identify Executable Files

- Create a new file, named “executable.yara”
- Start with a blank rule:

```
rule PE_file
{
}
```

Identify Executable Files

- Now add the two strings:

```
rule PE_file
{
    strings:
        $mz = "MZ"
        $pe = "PE"
}
```

- Note: Strings are case-sensitive by default!

Identify Executable Files

- A portable executable file MUST contain both strings. So, add the proper condition:

```
rule PE_file
{
    strings:
        $mz = "MZ"
        $pe = "PE"
    condition:
        $mz and $pe
}
```

- Test your rule file:

```
$ yara -r executable.yara /yara/malware
```

Identify Executable Files

- More constraints:
 - “MZ” at offset 0
 - UInt32 at offset 0x3c points to “PE”
- Refine your condition section:

condition:

(\$mz **at 0**) and

(\$pe **at (uint32(0x3c))**)
- Test your rule file again:

```
$ yara -r executable.yara /yara/malware
```

Identify Executable Files


- This is how your rule should look like:

```
rule PE_file
{
    strings:
        $mz = "MZ"
        $pe = "PE"
    condition:
        ($mz at 0) and
        ($pe at (uint32(0x3c)))
}
```

PE format

- PE = Portable Executable
- Structured format for Windows executable files
- Supporting documents in `/yara/doc/PE`
 - Overview by Ange Albertini
 - Specification v8.3 by Microsoft (2013)

PE format

PE¹⁰¹ a windows executable walkthrough  Ange Albertini corkami.com

Dissected PE

simple.exe

header
technical details about the executable

sections
contents of the executable

DOS header
shows the DOS header

PE header
shows the PE header

optional header
shows the optional header

data directories
points to data structures, imports, etc.

sections table
addresses how the file is loaded in memory

code
the code of the executable

imports
links between the executable and external libraries

data
information used by the code

| Hexadecimal dump | ASCII dump | Fields | Values | Explanation |
|---|------------|----------------------|-------------------|---|
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | e_magic | "MZ" | constant signature |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | e_filesize | 0x0 | offset of the PE Header |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | signature | "PE" | constant signature |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | machine | 0x14C (Intel 386) | processor architecture |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | numberofsections | 3 | number of sections |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | sizeofoptionalheader | 0x0 | relative offset of the section table |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | characteristics | 0x00000000 | EXECUTABLE... |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | magic | 0x10B (32b) | 32 bits |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | addressofentrypoint | 0x1000 | where execution starts |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | baseofimage | 0x400000 | address where the file should be mapped in memory |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | sectionalignment | 0x1000 | where sections should start in memory |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | filealignment | 0x200 | where sections should start on the disk |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | majorversion | 4 (or 4 or later) | required version of Windows |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | minorversion | 0x0000 | total memory space required |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | sizeofheaders | 0x200 | total size of the headers |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | subsystem | 2 (GUI) | dividing the document into |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | numberofdataentries | 16 | number of data structures |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | | importtable | 0x1000 | RVA of the imports |

| Sections table | | | | | |
|----------------|-----------------|--------------|------------------|---------------------|---------------------|
| Name | Virtual Address | Virtual Size | Size of raw data | Pointer to raw data | Characteristics |
| .text | 0x1000 | 0x200 | 0x200 | 0x1000 | CODE, EXECUTE, READ |
| .data | 0x1000 | 0x200 | 0x200 | 0x1000 | DATA, READ, WRITE |

For each section, a **SectionData** sized block is read from the file at **PointerToRawData** offset. It will be loaded in memory at address **ImageBase + VirtualAddress** in a **Virtual** sized block, with specific characteristics.

| x86 assembly | | Equivalent C code |
|---------------|--|---|
| push 0 | | |
| push 0x401007 | | |
| call 0x401007 | | MessageBox(0, "Hello world!", "simple PE executable", 0); |
| push 0 | | |
| call 0x401007 | | MessageBox(0, "Hello world!", "simple PE executable", 0); |

| Imports structures | | Consequences |
|--------------------|------------|--------------|
| ImportName | ImportData | |
| 0x200C | 0x200C | |
| 0x200D | 0x200D | |
| 0x200E | 0x200E | |
| 0x200F | 0x200F | |
| 0x2010 | 0x2010 | |
| 0x2011 | 0x2011 | |
| 0x2012 | 0x2012 | |
| 0x2013 | 0x2013 | |
| 0x2014 | 0x2014 | |
| 0x2015 | 0x2015 | |
| 0x2016 | 0x2016 | |
| 0x2017 | 0x2017 | |
| 0x2018 | 0x2018 | |
| 0x2019 | 0x2019 | |
| 0x201A | 0x201A | |
| 0x201B | 0x201B | |
| 0x201C | 0x201C | |
| 0x201D | 0x201D | |
| 0x201E | 0x201E | |
| 0x201F | 0x201F | |
| 0x2020 | 0x2020 | |
| 0x2021 | 0x2021 | |
| 0x2022 | 0x2022 | |
| 0x2023 | 0x2023 | |
| 0x2024 | 0x2024 | |
| 0x2025 | 0x2025 | |
| 0x2026 | 0x2026 | |
| 0x2027 | 0x2027 | |
| 0x2028 | 0x2028 | |
| 0x2029 | 0x2029 | |
| 0x202A | 0x202A | |
| 0x202B | 0x202B | |
| 0x202C | 0x202C | |
| 0x202D | 0x202D | |
| 0x202E | 0x202E | |
| 0x202F | 0x202F | |
| 0x2030 | 0x2030 | |
| 0x2031 | 0x2031 | |
| 0x2032 | 0x2032 | |
| 0x2033 | 0x2033 | |
| 0x2034 | 0x2034 | |
| 0x2035 | 0x2035 | |
| 0x2036 | 0x2036 | |
| 0x2037 | 0x2037 | |
| 0x2038 | 0x2038 | |
| 0x2039 | 0x2039 | |
| 0x203A | 0x203A | |
| 0x203B | 0x203B | |
| 0x203C | 0x203C | |
| 0x203D | 0x203D | |
| 0x203E | 0x203E | |
| 0x203F | 0x203F | |
| 0x2040 | 0x2040 | |
| 0x2041 | 0x2041 | |
| 0x2042 | 0x2042 | |
| 0x2043 | 0x2043 | |
| 0x2044 | 0x2044 | |
| 0x2045 | 0x2045 | |
| 0x2046 | 0x2046 | |
| 0x2047 | 0x2047 | |
| 0x2048 | 0x2048 | |
| 0x2049 | 0x2049 | |
| 0x204A | 0x204A | |
| 0x204B | 0x204B | |
| 0x204C | 0x204C | |
| 0x204D | 0x204D | |
| 0x204E | 0x204E | |
| 0x204F | 0x204F | |
| 0x2050 | 0x2050 | |
| 0x2051 | 0x2051 | |
| 0x2052 | 0x2052 | |
| 0x2053 | 0x2053 | |
| 0x2054 | 0x2054 | |
| 0x2055 | 0x2055 | |
| 0x2056 | 0x2056 | |
| 0x2057 | 0x2057 | |
| 0x2058 | 0x2058 | |
| 0x2059 | 0x2059 | |
| 0x205A | 0x205A | |
| 0x205B | 0x205B | |
| 0x205C | 0x205C | |
| 0x205D | 0x205D | |
| 0x205E | 0x205E | |
| 0x205F | 0x205F | |
| 0x2060 | 0x2060 | |
| 0x2061 | 0x2061 | |
| 0x2062 | 0x2062 | |
| 0x2063 | 0x2063 | |
| 0x2064 | 0x2064 | |
| 0x2065 | 0x2065 | |
| 0x2066 | 0x2066 | |
| 0x2067 | 0x2067 | |
| 0x2068 | 0x2068 | |
| 0x2069 | 0x2069 | |
| 0x206A | 0x206A | |
| 0x206B | 0x206B | |
| 0x206C | 0x206C | |
| 0x206D | 0x206D | |
| 0x206E | 0x206E | |
| 0x206F | 0x206F | |
| 0x2070 | 0x2070 | |
| 0x2071 | 0x2071 | |
| 0x2072 | 0x2072 | |
| 0x2073 | 0x2073 | |
| 0x2074 | 0x2074 | |
| 0x2075 | 0x2075 | |
| 0x2076 | 0x2076 | |
| 0x2077 | 0x2077 | |
| 0x2078 | 0x2078 | |
| 0x2079 | 0x2079 | |
| 0x207A | 0x207A | |
| 0x207B | 0x207B | |
| 0x207C | 0x207C | |
| 0x207D | 0x207D | |
| 0x207E | 0x207E | |
| 0x207F | 0x207F | |
| 0x2080 | 0x2080 | |
| 0x2081 | 0x2081 | |
| 0x2082 | 0x2082 | |
| 0x2083 | 0x2083 | |
| 0x2084 | 0x2084 | |
| 0x2085 | 0x2085 | |
| 0x2086 | 0x2086 | |
| 0x2087 | 0x2087 | |
| 0x2088 | 0x2088 | |
| 0x2089 | 0x2089 | |
| 0x208A | 0x208A | |
| 0x208B | 0x208B | |
| 0x208C | 0x208C | |
| 0x208D | 0x208D | |
| 0x208E | 0x208E | |
| 0x208F | 0x208F | |
| 0x2090 | 0x2090 | |
| 0x2091 | 0x2091 | |
| 0x2092 | 0x2092 | |
| 0x2093 | 0x2093 | |
| 0x2094 | 0x2094 | |
| 0x2095 | 0x2095 | |
| 0x2096 | 0x2096 | |
| 0x2097 | 0x2097 | |
| 0x2098 | 0x2098 | |
| 0x2099 | 0x2099 | |
| 0x209A | 0x209A | |
| 0x209B | 0x209B | |
| 0x209C | 0x209C | |
| 0x209D | 0x209D | |
| 0x209E | 0x209E | |
| 0x209F | 0x209F | |
| 0x20A0 | 0x20A0 | |
| 0x20A1 | 0x20A1 | |
| 0x20A2 | 0x20A2 | |
| 0x20A3 | 0x20A3 | |
| 0x20A4 | 0x20A4 | |
| 0x20A5 | 0x20A5 | |
| 0x20A6 | 0x20A6 | |
| 0x20A7 | 0x20A7 | |
| 0x20A8 | 0x20A8 | |
| 0x20A9 | 0x20A9 | |
| 0x20AA | 0x20AA | |
| 0x20AB | 0x20AB | |
| 0x20AC | 0x20AC | |
| 0x20AD | 0x20AD | |
| 0x20AE | 0x20AE | |
| 0x20AF | 0x20AF | |
| 0x20B0 | 0x20B0 | |
| 0x20B1 | 0x20B1 | |
| 0x20B2 | 0x20B2 | |
| 0x20B3 | 0x20B3 | |
| 0x20B4 | 0x20B4 | |
| 0x20B5 | 0x20B5 | |
| 0x20B6 | 0x20B6 | |
| 0x20B7 | 0x20B7 | |
| 0x20B8 | 0x20B8 | |
| 0x20B9 | 0x20B9 | |
| 0x20BA | 0x20BA | |
| 0x20BB | 0x20BB | |
| 0x20BC | 0x20BC | |
| 0x20BD | 0x20BD | |
| 0x20BE | 0x20BE | |
| 0x20BF | 0x20BF | |
| 0x20C0 | 0x20C0 | |
| 0x20C1 | 0x20C1 | |
| 0x20C2 | 0x20C2 | |
| 0x20C3 | 0x20C3 | |
| 0x20C4 | 0x20C4 | |
| 0x20C5 | 0x20C5 | |
| 0x20C6 | 0x20C6 | |
| 0x20C7 | 0x20C7 | |
| 0x20C8 | 0x20C8 | |
| 0x20C9 | 0x20C9 | |
| 0x20CA | 0x20CA | |
| 0x20CB | 0x20CB | |
| 0x20CC | 0x20CC | |
| 0x20CD | 0x20CD | |
| 0x20CE | 0x20CE | |
| 0x20CF | 0x20CF | |
| 0x20D0 | 0x20D0 | |
| 0x20D1 | 0x20D1 | |
| 0x20D2 | 0x20D2 | |
| 0x20D3 | 0x20D3 | |
| 0x20D4 | 0x20D4 | |
| 0x20D5 | 0x20D5 | |
| 0x20D6 | 0x20D6 | |
| 0x20D7 | 0x20D7 | |
| 0x20D8 | 0x20D8 | |
| 0x20D9 | 0x20D9 | |
| 0x20DA | 0x20DA | |
| 0x20DB | 0x20DB | |
| 0x20DC | 0x20DC | |
| 0x20DD | 0x20DD | |
| 0x20DE | 0x20DE | |
| 0x20DF | 0x20DF | |
| 0x20E0 | 0x20E0 | |
| 0x20E1 | 0x20E1 | |
| 0x20E2 | 0x20E2 | |
| 0x20E3 | 0x20E3 | |
| 0x20E4 | 0x20E4 | |
| 0x20E5 | 0x20E5 | |
| 0x20E6 | 0x20E6 | |
| 0x20E7 | 0x20E7 | |
| 0x20E8 | 0x20E8 | |
| 0x20E9 | 0x20E9 | |
| 0x20EA | 0x20EA | |
| 0x20EB | 0x20EB | |
| 0x20EC | 0x20EC | |
| 0x20ED | 0x20ED | |
| 0x20EE | 0x20EE | |
| 0x20EF | 0x20EF | |
| 0x20F0 | 0x20F0 | |
| 0x20F1 | 0x20F1 | |
| 0x20F2 | 0x20F2 | |
| 0x20F3 | 0x20F3 | |
| 0x20F4 | 0x20F4 | |
| 0x20F5 | 0x20F5 | |
| 0x20F6 | 0x20F6 | |
| 0x20F7 | 0x20F7 | |
| 0x20F8 | 0x20F8 | |
| 0x20F9 | 0x20F9 | |
| 0x20FA | 0x20FA | |
| 0x20FB | 0x20FB | |
| 0x20FC | 0x20FC | |
| 0x20FD | 0x20FD | |
| 0x20FE | 0x20FE | |
| 0x20FF | 0x20FF | |

| Strings | |
|---------|--------|
| 0x1000 | 0x1000 |
| 0x1001 | 0x1001 |
| 0x1002 | 0x1002 |
| 0x1003 | 0x1003 |
| 0x1004 | 0x1004 |
| 0x1005 | 0x1005 |
| 0x1006 | 0x1006 |
| 0x1007 | 0x1007 |
| 0x1008 | 0x1008 |
| 0x1009 | 0x1009 |
| 0x100A | 0x100A |
| 0x100B | 0x100B |
| 0x100C | 0x100C |
| 0x100D | 0x100D |
| 0x100E | 0x100E |
| 0x100F | 0x100F |
| 0x1010 | 0x1010 |
| 0x1011 | 0x1011 |
| 0x1012 | 0x1012 |
| 0x1013 | 0x1013 |
| 0x1014 | 0x1014 |
| 0x1015 | 0x1015 |
| 0x1016 | 0x1016 |
| 0x1017 | 0x1017 |
| 0x1018 | 0x1018 |
| 0x1019 | 0x1019 |
| 0x101A | 0x101A |
| 0x101B | 0x101B |
| 0x101C | 0x101C |
| 0x101D | 0x101D |
| 0x101E | 0x101E |
| 0x101F | 0x101F |
| 0x1020 | 0x1020 |
| 0x1021 | 0x1021 |
| 0x1022 | 0x1022 |
| 0x1023 | 0x1023 |
| 0x1024 | 0x1024 |
| 0x1025 | 0x1025 |
| 0x1026 | 0x1026 |
| 0x1027 | 0x1027 |
| 0x1028 | 0x1028 |
| 0x1029 | 0x1029 |
| 0x102A | 0x102A |
| 0x102B | 0x102B |
| 0x102C | 0x102C |
| 0x102D | 0x102D |
| 0x102E | 0x102E |
| 0x102F | 0x102F |
| 0x1030 | 0x1030 |
| 0x1031 | 0x1031 |
| 0x1032 | 0x1032 |
| 0x1033 | 0x1033 |
| 0x1034 | 0x1034 |
| 0x1035 | 0x1035 |
| 0x1036 | 0x1036 |
| 0x1037 | 0x1037 |
| 0x1038 | 0x1038 |
| 0x1039 | 0x1039 |
| 0x103A | 0x103A |
| 0x103B | 0x103B |
| 0x103C | 0x103C |
| 0x103D | 0x103D |
| 0x103E | 0x103E |
| 0x103F | 0x103F |
| 0x1040 | 0x1040 |
| 0x1041 | 0x1041 |
| 0x1042 | 0x1042 |
| 0x1043 | 0x1043 |
| 0x1044 | 0x1044 |
| 0x1045 | 0x1045 |
| 0x1046 | 0x1046 |
| 0x1047 | 0x1047 |
| 0x1048 | 0x1048 |
| 0x1049 | 0x1049 |
| 0x104A | 0x104A |
| 0x104B | 0x104B |
| 0x104C | 0x104C |
| 0x104D | 0x104D |
| 0x104E | 0x104E |
| 0x104F | 0x104F |
| 0x1050 | 0x1050 |
| 0x1051 | 0x1051 |
| 0x1052 | 0x1052 |
| 0x1053 | 0x1053 |
| 0x1054 | 0x1054 |
| 0x1055 | 0x1055 |
| 0x1056 | 0x1056 |
| 0x1057 | 0x1057 |
| 0x1058 | 0x1058 |
| 0x1059 | 0x1059 |
| 0x105A | 0x105A |
| 0x105B | 0x105B |
| 0x105C | 0x105C |
| 0x105D | 0x105D |
| 0x105E | 0x105E |
| 0x105F | 0x105F |
| 0x1060 | 0x1060 |
| 0x1061 | 0x1061 |
| 0x1062 | 0x1062 |
| 0x1063 | 0x |

Identify Other Files

- Write a rule to detect a PDF file
- Write a rule to detect a ZIP file
- What is needed?

Identify Other Files

- Write a rule to detect a PDF file
- Write a rule to detect a ZIP file
- What is needed?
 - This is what you need: [List](#)

Detecting PDF File Type

- Write a rule to detect the PDF type

```
rule DetectPDF
{
  strings:
    $pdf = { 25 50 44 46 } // %PDF
  condition:
    $pdf at 0
}
```

Checking Specific Values at a Location

- The malware version is at byte 0x20:

```
rule VersionCheck
{
    strings:
        $ver = "v1.2.3"
    condition:
        $ver at 0x20
}
```

Sets of Strings (Multiple Indicators)

- Match at least two indicators:

```
rule MultipleIndicators
{
    strings:
        $a = "keylogger"
        $b = "stealer"
        $c = "injector"
    condition:
        2 of ($a,$b,$c)
}
```

Using Regular Expressions

- Detecting data exfiltration via email :

```
rule EmailPattern
```

```
{
```

```
  strings:
```

```
    $email = /[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}/
```

```
  condition:
```

```
    $email
```

```
}
```

Matching on Machine Code

Matching Assembly Code Patterns

- Check code patterns: Detecting known shellcode patterns

```
rule SuspiciousASM
{
  strings:
    $asm = { 8B ?? ?? 89 ?? ?? 8D }
  condition:
    $asm
}
```

- 8B = MOV, 8D = LEA (common in shellcode)

About the malware samples

- Objective
 - How to build binary signatures that match on x86 machine code
- Sample: 44efa4accc42aa55d7843ec69161c8ca:
 - The sample is a backdoor belonging to the Hoardy family.

The first decryption routine

- Disassembly of sample 44efa4accc42aa55d7843ec69161c8ca:

```
.text:00401723
.text:00401723 89 45 E8
.text:00401726 3B C7
.text:00401728 7D 18
.text:0040172A 8A 88 F0 E8 40 00
.text:00401730 32 C8
.text:00401732 2A C8
.text:00401734 80 E9 5A
.text:00401737 88 88 F0 E8 40 00
.text:0040173D 83 C0 01
.text:00401740 EB E1
```

```
decrypt:
mov     [ebp+0BB4h+var_BCC], eax
cmp     eax, edi
jge     short end
mov     cl, buffer[eax]
xor     cl, al
sub     cl, al
sub     cl, 5Ah
mov     buffer[eax], cl
add     eax, 1
jmp     short decrypt
```

The first decryption routine

- Create a rule file named “hoardy.yara”.
- Create a YARA rule which matches on the bytes that are typeset in bold letters (see previous page).
- Name your rule “crypto1” and tag it as “category”.
- Name the string “\$crypto1”, too.
- Try your rule on the sample.

Practice Time!

- Get one c code executable from the previous labs
- Analyze the executable file
- Write a Yara rule to match a function pattern
- Where to find the instructions?
 - X86 architecture: <http://ref.x86asm.net/coder32.html>

Conclusion

Conclusion

Strings

- Text
 - make use of modifiers: nocase, fullword, ascii, wide
- Hex
 - make use of wildcards and jumps
- Perl compatible regular expressions

Conclusion

Condition

- Sets
 - 2 of (\$a,\$b,\$c)
 - any of them
 - all of them
- Count number of string matches: #string
- Iterator “for”
- Offsets:
 - at offset
 - Entrypoint
- Access raw bytes: int8..int32, uint8..uint32
- Keep your rules simple, reference other rules

Conclusion

Metadata

- Define metadata
 - string
 - integer
 - boolean
- Examples:
 - weight (count of matching bits)
 - Architecture
 - Algorithm
 - endianness
- Use “-m” command line option to display metadata

Conclusion

Maintain a set of rules

- One-file-to-keep-them-all doesn't work well
- Refactor your rules
 - write rules for each common expression (“primitives”)
 - separate files by topic, make use of “include”
- Rule modifiers:
 - “**global**” makes rule a prerequisite for all other rules (e.g. PE header check)
 - “**private**” suppresses output
- Make use of tags and “-t” command line option to select rules

Conclusion

More information

- YARA manuals and wiki at
 - <http://code.google.com/p/yara-project/>

Conclusion

- Format of Yara rule
- Write Yara rule
- PE format
- Write rule for binary matching

Reference

- Some slides' content are borrowed from Andreas Schuster presentation slide on 26th FIRST conference in Boston, 2014:
 - https://www.first.org/resources/papers/conference2014/first_2014_-_schuster-andreas-yara_basic_and_advanced_20140619.pdf
- The slides are changed to match the requirement of the class