YARA RULES

MALWARE ANALYSIS AND INCIDENT FORENSICS

M.Sc. in Cyber Security

A.Y. 2025/2026

SAPIENZA
UNIVERSITÀ DI ROMA

CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

WHAT

What is YARA?

- Think of it of a sort of “grep” on steroids
  - To quickly check if a new observable matches some known suspicious characteristics
  - To search for a new bit of information in a large dataset
- Much more
  - Helps you keep an organized library of matching rules for different threats based on observed IoCs

What is not

- An antivirus
- A host-based IDS

The good point is that YARA is at the same time easy to use, flexible, and extremely powerful.

WHAT

A tool that you can download/compile for your OS

```
$ 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 BASIC RULES

At a minimum, a rule must have a name and a condition

The simplest possible rule is:

```
rule dummy { condition: false }
```

That rule does nothing. Inversely, this rule matches on anything:

```
rule dummy { condition: true }
```

Slightly more useful example that will match on any file over 500 KB:

```
rule over_500kb {condition: filesize > 500KB}
```

YARA BASIC RULES

Standard structure for Yara rules:

```
rule silent_banker : banker {

meta:
  description = "This is just an example"
  threat_level = 3
  in_the_wild = true

strings:
   $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
   $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
   $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

condition:
  all of them
}
```

YARA BASIC RULES

- **Meta** section - consists of a set of arbitrary key-value pairs
  - Can be used to describe the rule and/or the type of content that it matches
  - Meta values can be strings, integers, decimals, or booleans
  - When a match occurs, an application that is using YARA sees the meta values
- **Strings** section - defines variables for content to be matched
  - Hex byte patterns (between {}, can feature wildcards and “jumps”)
    => often used to identify unique code (e.g., an unpacking mechanism)
  - Text strings
  - Regular expressions (between //)

YARA BASIC RULES

- **Conditions** section - define matching conditions
  - A few examples:

CONDITION	MEANING
`any of them`	The rule will match on anything containing any of the strings defined in the rule
`all of them`	The rule will only match if all of the defined strings are in the content
`3 of them`	The rule will match anything containing at least three of the defined strings
`$a and 3 of ($s*)`	Match content that contains string $a and at least three strings whose variable name begins with $s

  - Check the docs for full syntax

YARA EXAMPLE

```
rule pdf_1.7_contains_few_links {

meta:
  author = "Sean Whalen"
  last_updated = "2017-06-08"
  tlp = "white"
  category = "malicious"
  confidence = "medium"
  killchain_phase = "exploit"
  description = "A PDFv1.7 that contains one or two links - a common phishing tactic"

strings:
  $pdf_magic = {25 50 44 46}
  $s_anchor_tag = "<a " ascii wide nocase
  $s_uri = /\(http.+\)/ ascii wide nocase

condition:
  $pdf_magic at 0 and (#s_anchor_tag == 1 or (#s_uri > 0 and #s_uri < 3))
}
```

IMPORTS AND MODULES

Several modules can be imported

- Example: PE interpreter

```
import “PE”

rule suspicious_pe {

 Strings:
      $str = “procmon”

 Condition:
      pe.imports( “kernel32.dll”, “CreateProcess” ) and
      pe.imports( “wininet.dll”, “HttpSendRequest” ) and
      $str
}
```

REFERENCES

You can reference other rules:

```
rule RULE_EVILMD5 {
    meta:
        description = “Suspicious MD5 Hash”
    condition:
        md5 matches /^3bb34a700e8d21acfdfe0f09208a7c01$/
}
rule RULE_EVILPATH {
    meta:
        description = “Suspicious File Path”
    condition:
        path contains “/appdata/roaming/”
}
rule RULE_SUSP_BHV {
    meta:
        description = “Suspicious Behavior”
    condition:
        RULE_EVILMD5 and RULE_EVILPATH
}
```

EXAMPLE #1

Detecting Zeus through elements of its typical HTTP request pattern:

```
import "pe"
rule Windows_Malware_Zeus : Zeus_1134 {
    meta:
        author = "Xylitol xylitol@malwareint.com"
        date = "2014-03-03"
        description = "Match first two bytes, protocol and string present in Zeus
1.1.3.4"
        reference = "http://www.xylibox.com/2014/03/zeus-1134.html"
    strings:
        $mz = {4D 5A}
        $protocol1 = "X_ID: "
        $protocol2 = "X_OS: "
        $protocol3 = "X_BV: "
        $stringR1 = "InitializeSecurityDescriptor"
        $stringR2 = "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1)"
    condition:
        ($mz at 0 and all of ($protocol*) and ($stringR1 or $stringR2))
}
```

EXAMPLE #2

Identify PE files

```
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...|
```

Points to PE header location

EXAMPLE #2

Identify PE files:

```
rule is_PE_file {

    strings:
        $mz = "MZ"
        $pe = "PE"

    condition:
        ($mz at 0) and
        ($pe at (uint32(0x3c)))
}
```

MORE ADVANCED CASES

Obfuscated string

```
00415393  C6 45 CC 53 C6 45 CD 6F C6 45 CE 66 C6 45 CF 74  .E.S.E.o.E.f.E.t
004153A3  C6 45 D0 77 C6 45 D1 61 C6 45 D2 72 C6 45 D3 65  .E.w.E.a.E.r.E.e
004153B3  C6 45 D4 5C C6 45 D5 4D C6 45 D6 69 C6 45 D7 63  .E.\.E.M.E.i.E.c
004153C3  C6 45 D8 72 C6 45 D9 6F C6 45 DA 73 C6 45 DB 6F  .E.r.E.o.E.s.E.o
004153D3  C6 45 DC 66 C6 45 DD 74 C6 45 DE 5C C6 45 DF 57  .E.f.E.t.E.\.E.W
004153E3  C6 45 E0 69 C6 45 E1 6E C6 45 E2 64 C6 45 E3 6F  .E.i.E.n.E.d.E.o
004153F3  C6 45 E4 77 C6 45 E5 73 C6 45 E6 5C C6 45 E7 43  .E.w.E.s.E.\.E.C
00415403  C6 45 E8 75 C6 45 E9 72 C6 45 EA 72 C6 45 EB 65  .E.u.E.r.E.r.E.e
00415413  C6 45 EC 6E C6 45 ED 74 C6 45 EE 56 C6 45 EF 65  .E.n.E.t.E.V.E.e
00415423  C6 45 F0 72 C6 45 F1 73 C6 45 F2 69 C6 45 F3 6F  .E.r.E.s.E.i.E.o
00415433  C6 45 F4 6E C6 45 F5 5C C6 45 F6 52 C6 45 F7 75  .E.n.E.\.E.R.E.u
00415443  C6 45 F8 6E                                     .E.n
```

Can you spot any recognizable string?

MORE ADVANCED CASES

Obfuscated string

```
00415393  C6 45 CC 53 C6 45 CD 6F C6 45 CE 66 C6 45 CF 74  .E.S.E.o.E.f.E.t
004153A3  C6 45 D0 77 C6 45 D1 61 C6 45 D2 72 C6 45 D3 65  .E.w.E.a.E.r.E.e
004153B3  C6 45 D4 5C C6 45 D5 4D C6 45 D6 69 C6 45 D7 63  .E.\.E.M.E.i.E.c
004153C3  C6 45 D8 72 C6 45 D9 6F C6 45 DA 73 C6 45 DB 6F  .E.r.E.o.E.s.E.o
004153D3  C6 45 DC 66 C6 45 DD 74 C6 45 DE 5C C6 45 DF 57  .E.f.E.t.E.\.E.W
004153E3  C6 45 E0 69 C6 45 E1 6E C6 45 E2 64 C6 45 E3 6F  .E.i.E.n.E.d.E.o
004153F3  C6 45 E4 77 C6 45 E5 73 C6 45 E6 5C C6 45 E7 43  .E.w.E.s.E.\.E.C
00415403  C6 45 E8 75 C6 45 E9 72 C6 45 EA 72 C6 45 EB 65  .E.u.E.r.E.r.E.e
00415413  C6 45 EC 6E C6 45 ED 74 C6 45 EE 56 C6 45 EF 65  .E.n.E.t.E.V.E.e
00415423  C6 45 F0 72 C6 45 F1 73 C6 45 F2 69 C6 45 F3 6F  .E.r.E.s.E.i.E.o
00415433  C6 45 F4 6E C6 45 F5 5C C6 45 F6 52 C6 45 F7 75  .E.n.E.\.E.R.E.u
00415443  C6 45 F8 6E                                     .E.n
```

Repeated pattern with variable ending

MORE ADVANCED CASES

Opcodes

1st \ 2nd	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD						ES PUSH SS	ES POP SS	OR						CS PUSH DS	TWO BYTE
1	ADC								SBB							POP DS
2	AND						ES SEGMENT OVERRIDE	DAA	SUB						CS SEGMENT OVERRIDE	DAS
3	XOR						SS	AAA	CMP						DS	AAS
4	INC								DEC							
5	PUSH								POP							
6	PUSHAD	POPAD	BOUND	ARPL	FS GS SEGMENT OVERRIDE		OPERAND SIZE ADDRESS SIZE SIZE OVERRIDE		PUSH	IMUL	PUSH	IMUL	INS		OUTS	
7	JO	JNO	JB	JNB	JE	JNE	JBE	JA	JS	JNS	JPE	JPO	JL	JGE	JLE	JG
	Jcc															
8	ADD/ADC/AND/XOR OR/SBB/SUB/CMP				TEST		XCHG		MOV REG				MOV SREG	LEA	MOV SREG	POP
9	NOP	XCHG EAX							CWD	CDQ	CALLF	WAIT	PUSHFD	POPFD	SAHF	LAHF
A	MOV EAX				MOVS		CMPS		TEST		STOS		LODS		SCAS	
B	MOV															
C	SHIFT IMM		RETN		LES	LDS	MOV IMM		ENTER	LEAVE	RETF		INT3	INT IMM	INTO	IRETD
D	SHIFT 1 SHIFT CL ROL/ROR/RCL/RCR/SHL/SHR/SAL/SAR				AAM	AAD	SALC	XLAT	FPU							
E	LOOPNZ LOOPZ LOOP CONDITIONAL LOOP			JECXZ	IN IMM		OUT IMM		CALL	JMP	JMPF	JMP SHORT	IN DX		OUT DX	
F	LOCK EXCLUSIVE ACCESS	ICE BP		REPNE REPE CONDITIONAL REPETITION	HLT	CMC	TEST/NOT/NEG [i]MUL/[i]DIV		CLC	STC	CLI	STI	CLD	STD	INC DEC	INC/DEC CALL/JMP PUSH

Source:
Extract from „x86 Opcode Structure and Instruction Overview“
by Daniel Plohmann, Fraunhofer FKIE

MORE ADVANCED CASES

Check MOV's page on the Intel manual

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
REX.W + A3	MOV *moffs64**,RAX	D	Valid	N.E.	Move RAX to (*offset*).
B0+ *rb*	MOV *r8, imm8*	E	Valid	Valid	Move *imm8* to *r8.*
REX + B0+ *rb*	MOV *r8*** , *imm8*	E	Valid	N.E.	Move *imm8* to *r8.*
B8+ *rw*	MOV *r16, imm16*	E	Valid	Valid	Move *imm16* to *r16.*
B8+ *rd*	MOV *r32, imm32*	E	Valid	Valid	Move *imm32* to *r32.*
REX.W + B8+ *rd*	MOV *r64, imm64*	E	Valid	N.E.	Move *imm64* to *r64.*
C6 */0*	MOV *r/m8, imm8*	F	Valid	Valid	Move *imm8* to *r/m8.*
REX + C6 */0*	MOV *r/m8***, *imm8*	F	Valid	N.E.	Move *imm8* to *r/m8.*
C7 */0*	MOV *r/m16, imm16*	F	Valid	Valid	Move *imm16* to *r/m16.*
C7 */0*	MOV *r/m32, imm32*	F	Valid	Valid	Move *imm32* to *r/m32.*
REX.W + C7 */0*	MOV *r/m64, imm32*	F	Valid	N.E.	Move *imm32 sign extended to 64-bits* to *r/m64.*

MORE ADVANCED CASES

Check MOV's page on the Intel manual

Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--][1]		100	04	0C	14	1C	24	2C	34	3C
disp32[2]		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
[EAX]+disp8[3]	01	000	40	48	50	58	60	68	70	78
[ECX]+disp8		001	41	49	51	59	61	69	71	79
[EDX]+disp8		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		011	43	4B	53	5B	63	6B	73	7B
[--][--]+disp8		100	44	4C	54	5C	64	6C	74	7C
[EBP]+disp8		101	45	4D	55	5D	65	6D	75	7D
[ESI]+disp8		110	46	4E	56	5E	66	6E	76	7E
[EDI]+disp8		111	47	4F	57	5F	67	6F	77	7F
[EAX]+disp32	10	000	80	88	90	98	A0	A8	B0	B8

MORE ADVANCED CASES

The single-byte mov instruction was used to conceal the string

```
00415393    mov     [ebp+SubKey],     'S'   ;C645CC53
00415397    mov     [ebp+SubKey+1],   'o'   ;C645CD6F
0041539B    mov     [ebp+SubKey+2],   'f'   ;C645CE66
0041539F    mov     [ebp+SubKey+3],   't'   ;C645CF74
004153A3    mov     [ebp+SubKey+4],   'w'   ;C645D077
004153A7    mov     [ebp+SubKey+5],   'a'   ;C645D161
004153AB    mov     [ebp+SubKey+6],   'r'   ;C645D272
004153AF    mov     [ebp+SubKey+7],   'e'   ;C645D365
004153B3    mov     [ebp+SubKey+8],   '\'   ;C645D45C
004153B7    mov     [ebp+SubKey+9],   'M'   ;C645D54D
004153BB    mov     [ebp+SubKey+0Ah], 'i'   ;C645D669
004153BF    mov     [ebp+SubKey+0Bh], 'c'   ;C645D763
004153C3    mov     [ebp+SubKey+0Ch], 'r'   ;C645D872
004153C7    mov     [ebp+SubKey+0Dh], 'o'   ;C645D96F
004153CB    mov     [ebp+SubKey+0Eh], 's'   ;C645DA73
004153CF    mov     [ebp+SubKey+0Fh], 'o'   ;C645DB6F
004153D3    mov     [ebp+SubKey+10h], 'f'   ;C645DC66
```

MORE ADVANCED CASES

Signature:

- 0xC6 0x45 is a constant (opcode and r/m8)
- disp8 (index) is variable, but restricted to a single byte
- the character (imm8) is variable, but also restricted to a single byte

Pattern: C6 45 ?? ?? C6 45 ?? ?? C6 45 ...

- Better use jumps: [x]
- [] are used to define strings with chunks of variable content (and length [x-y])

MORE ADVANCED CASES

Identify the pattern we discussed:

```
rule single_byte_mov {

   strings:
      $a = { C6 45 [2] C6 45 [2] C6 45 }

   condition:
      $a
}
```

This only matches a few chars. If you want to match the full string:

```
$a = /(\xc6\x45..){3,}/
```

MORE ADVANCED CASES

Look for constants that are important for an algorithm

The longer the sequence, the better (=> fewer false positives)

Examples:

- static substitution box (s-box) of DES
- MD5 init and transform constants
- polynomial for Cyclic Redundancy Check

Beware of endianness issues

- 0x1234 can be stored as 0x12 0x34 or 0x34 0x12

Consider breaking up long numbers, loading into different registers, optimizations by compiler

MORE ADVANCED CASES

Example: Linear Congruential Generator (LCG)

$$x_{n+1} = (ax_n + c) \ mod \ m$$

Used as a RNG in PoisonIvy:

```
00000DA5 rand_init:
00000DA5       lea    esi, [ebp+base]
00000DAB       rdtsc                 ; seed with CPU tick counter
00000DAD       xchg   eax, edx
00000DAE       xor    ecx, ecx
00000DB0 rand_loop:                  ; LCG x := (x * 2891336453 + 1) mod 2^32
00000DB0       imul   eax, 2891336453
00000DB6       add    eax, 1
00000DB9       mov    [esi+ecx*4+8D9h], eax
00000DC0       add    ecx, 1
00000DC3       cmp    ecx, 34
00000DC6       jb     short rand_loop
```

MORE ADVANCED CASES

Corresponding rule

```
rule has_rng {

   strings:
      $a = { 0F 31 }      // rdtsc
      $b = “2891336453”   // static value of parameter

   condition:
      all of them
}
```

Watch out for false positives!

MORE ADVANCED CASES

Check for imports:

```
import "pe"

rule potential_keylogger {
    strings:
        $autorun = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" wide
ascii

    condition:
        pe.imports("User32.dll", "SetWindowsHookEx")
        and
        pe.imports("User32.dll", “RegisterHotKey”)
        and
        $autorun
}
```

MORE ADVANCED CASES

Check for potentially packed software *(return to this slide in 1-2 weeks 😄)*

```
import "pe"
import "math"
rule possibly_packed {
    meta:
    description = "Rule to detect binaries with a big difference between
                   section size and section vsize (after unpack). Also, it
                   includes a big entropy and executable flags"

    condition:
        for any i in ( 0..pe.number_of_sections - 1):(
            (pe.sections[i].virtual_size > pe.sections[i].raw_data_size*2)
            and
            pe.sections[i].characteristics & SECTION_MEM_EXECUTE
            and
            math.entropy(pe.sections[i].raw_data_offset,
                         pe.section[I].raw_data_size) >= 7
        )
}
```

MORE ADVANCED CASES

Check for clearly packed software

(return to this slide too in 1-2 weeks 😄)

```
import "pe"

rule UPX {

    strings:
        $upx = "UPX"

    condition:
        pe.sections[0].name contains “UPX0”
        and
        pe.sections[1].name contains “UPX1”
        and
        $upx
}
```