Yara Studies: A Deep Dive into Scanning Performance

TLP White

Botconf 2023

Dominika Regéciová

A Few Notes About Me

- Pronouns: she/her/hers
- Researcher at Gen (Avast)
- Projects with ESA and Czech Police
- My research:
  - Formal models and languages in security
  - Pattern matching
  - Blockchain technology
- Botconf 2021/2022 - Yara: Down the Rabbit Hole Without Slowing Down

Motivation For This Talk

- I work with malware analysts writing Yara rules
- It is easy to write Yara rules but hard to write good Yara rules
- It is hard to create rules with high precision and performance
- I am helping analysts to archive:
  - More precise detection patterns
  - Fewer false positives and a fix for the problem with too many matches
  - A fast scanning speed

Introduction: Yara

Introduction: Yara

```
/* Yara rule example */
rule example_rule {
    meta:
        author = "Dominika Regeciova"
    strings:
        $str = "Hello World!" fullword nocase
        $re = /abcd[x-z]/
        $hex = { 63 62 61 }
    condition:
        $hex at 0 or
        $re or
        $str
}
```

Introduction: Yara

```
/* Yara rule example */
rule example_rule {
    meta:
        author = "Dominika Regeciova"
    strings:
        $str = "Hello World!" fullword nocase
        $re = /abcd[x-z]/
        $hex = { 63 62 61 }
    condition:
        $hex at 0 or
        $re or
        $str
}
```

Gen

Introduction: Yara

```
/* Yara rule example */
rule example_rule {
    meta:
        author = "Dominika Regeciova"
    strings:
        $str = "Hello World!" fullword nocase
        $re = /abcd[x-z]/
        $hex = { 63 62 61 }
    condition:
        $hex at 0 or
        $re or
        $str
}
```

Introduction: Yara

```
/* Yara rule example */
rule example_rule {
    meta:
        author = "Dominika Regeciova"
    strings:
        $str = "Hello World!" fullword nocase
        $re = /abcd[x-z]/
        $hex = { 63 62 61 }
    condition:
        $hex at 0 or
        $re or
        $str
}
```

Introduction: Yara

```
/abcd[x-z]/
```

Introduction: Yara

```
/abcd[x-z]/
```

↓

```
abcd
```

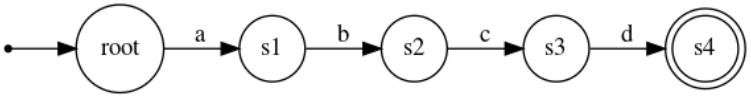
Introduction: Yara

```
/abcd[x-z]/
```

↓

```
abcd
```

↓



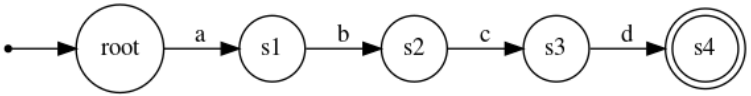
Introduction: Yara

```
/abcd[x-z]/
```

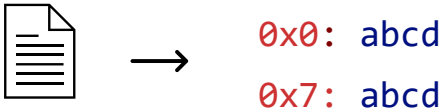
↓

```
abcd
```

↓



↓



```
0x0: abcd
0x7: abcd
```

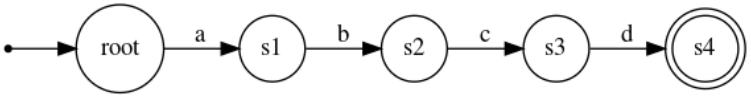

Introduction: Yara

```
/abcd[x-z]/
```

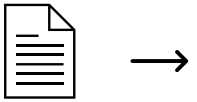
↓

```
abcd
```

↓



↓



```
0x0: abcd
0x7: abcd
```

↓

```
0x0: abcdx => match
0x7: abcdf => not a match
```

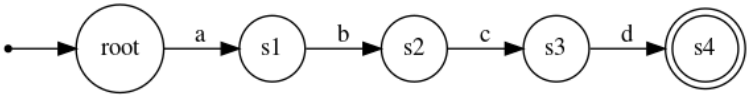
Introduction: Yara

```
/abcd[x-z]/
```

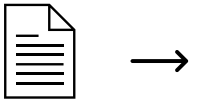
↓

```
abcd
```

↓



↓



```
0x0: abcd
0x7: abcd
```

↓

```
0x0: abcdx => match
0x7: abcdf => not a match
```

↓

```
Condition evaluation → MATCH
```


Introduction: Yara

- Yara in version 4.2.3 was used
- Publicly available datasets were used for scanning
- Together we are scanning around 22 GB of data

```
rule test_00
{
  condition:
    false
}
```

```
$ time ./yara 00.yar -r dataset/
real    0m36.297s
user    0m57.502s
sys     0m18.932s
```


Study I: Strings vs. Condition

Study I: Strings vs. Condition

- Conditions are evaluated **AFTER** the strings definition part
- Yara will be searching for string *$h00* in the whole sample, no matter of filesize and position in the file

```
rule test_01 {
    strings:
        $h00 = { 42 ?? ?? 00 00 61 62 }
    condition:
        filesize < 1KB and
        $h00 at 0
}
```

Study I: Strings vs. Condition

```
rule test_02 {
    condition:
        filesize < 1KB and
        uint8(0) == 0x42 and
        uint32(3) == 0x62610000
}
```

```
$ time ./yara 01.yar -r dataset/
real    0m39.934s
user    1m7.489s
sys     0m19.024s

$ time ./yara 02.yar -r dataset/
real    0m36.565s
user    0m56.980s
sys     0m19.227s
```

```
rule test_01 {
    strings:
        $h00 = { 42 ?? ?? 00 00 61 62 }
    condition:
        filesize < 1KB and
        $h00 at 0
}
```

Study II: Potentially There

Study II: Potentially There

- The analyst was trying to match the word **powershell**, sometimes spaced with caret symbols: p^owershell, p^o^wershell, …
- For Yara, it is hard to search for a sequence of characters where every other character can be missing

```
rule test_03 {
    strings:
       $re=/p\^?o\^?w\^?e\^?r\^?s\^?h\^?e\^?l\^?l/
    condition:
        $re
}
```

Study II: Potentially There

```
rule test_04 {
    strings:
       $re1=/p\^o\^?w\^?e\^?r\^?s\^?h\^?e\^?l\^?l/
       $re2=/po\^?w\^?e\^?r\^?s\^?h\^?e\^?l\^?l/
    condition:
        $re1 or $re2
}
```

```
rule test_03 {
    strings:
       $re=/p\^?o\^?w\^?e\^?r\^?s\^?h\^?e\^?l\^?l/
    condition:
        $re
}
```

```
$ time ./yara 03.yar -r dataset/
warning: rule "test_03" in
03.yar(3): string "$re" may slow
down scanning
real    0m43.650s
user    1m7.416s
sys     0m21.658s

$ time ./yara 04.yar -r dataset/
real    0m37.430s
user    0m59.172s
sys     0m20.318s
```

Study III: Problematic Alternations

Study III: Problematic Alternations

- For short strings, the notation can influence the effectiveness of them
- Yara can not work with short alternations effectively
- In general, try to create text substrings with a length of 3 or 4 bytes

```
rule test_05 {
    strings:
        $hex = { 44 (03|2E) 33 }
    condition:
        $hex
}
```


Study III: Problematic Alternations

```
rule test_06 {
    strings:
        $hex0 = { 44 03 33 }
        $hex1 = { 44 2E 33 }
    condition:
        $hex0 or $hex1
}
```

```
rule test_05 {
    strings:
        $hex = { 44 (03|2E) 33 }
    condition:
        $hex
}
```

```
$ time ./yara 05.yar -r dataset/
warning: rule "test_05" in
05.yar(3): string "$hex" may
slow down scanning
real    0m46.142s
user    1m12.487s
sys     0m21.040s

$ time ./yara 06.yar -r dataset/
real    0m37.241s
user    0m59.962s
sys     0m17.781s
```

Gen

Study IV: Too General

Study IV: Too General

- The rule can look OK, but the prefix .* part can slow down Yara a lot
- The same applies to regular expressions such as .+ or .{X,}
- Can generate warnings about too many matches based on the input files

```
rule test_07 {
    strings:
        $re = /.*\.exe/ nocase ascii wide
    condition:
        $re
}
```

Study IV: Too General

- The rule can look OK, but the prefix .* part can slow down Yara a lot
- The same applies to regular expressions such as .+ or .{X,}
- Can generate warnings about too many matches
- It can be unintuitive, but less is sometimes more

```
/.*\.exe/
```

↓

```
".exe"
```

Study IV: Too General

```
rule test_08 {
    strings:
        $re = ".exe" nocase ascii wide
    condition:
        $re
}

rule test_07 {
    strings:
        $re = /.*\.exe/ nocase ascii wide
    condition:
        $re
}
```

```
$ time ./yara 07.yar -r dataset/
warning: rule "test_07" in
07.yar(3): $re contains .*, .+
or .{x,} consider using .{,N},
.{1,N} or {x,N} with a
reasonable value for N
real    1m2.497s
user    13m41.575s
sys     1m9.662s

$ time ./yara 08.yar -r dataset/
real    0m37.388s
user    0m59.060s
sys     0m19.649s
```

Study V: Secret Agent 6, IPv6

Study V: Secret Agent 6, IPv6

- The analyst wanted to match the IPv6 addresses
- Their first version of rule cause many false positives

```
rule test_09 {
    strings:
        $ipv6 = /([a-f0-9:]+:+)+[a-f0-9]+/ fullword nocase ascii
    condition:
        $ipv6
}
```

Study V: Secret Agent 6, IPv6

- The analyst wanted to match the IPv6 addresses
- Their first version of rule cause many false positives
- We narrowed it down to only global unicast addresses starting with the prefix 2001

```
/([a-f0-9:]+:+)+[a-f0-9]+/
```

↓

```
/2001:([a-f0-9]{0,4}:){1,6}[a-f0-9]{0,4}/
```

Study V: Secret Agent 6, IPv6

```
rule test_10 {
  strings:
    $ipv6 = /2001:([a-f0-9]{0,4}:){1,6}[a-f0-9]{0,4}/
          fullword nocase ascii
  condition:
    $ipv6
}
```

```
rule test_09 {
    strings:
        $ipv6 = /([a-f0-9:]+:+)+[a-f0-9]+/
             fullword nocase ascii
    condition:
        $ipv6
}
```

```
$ time ./yara 09.yar -r dataset/
warning: rule "test_09" in
09.yar(3): string "$ipv6" may
slow down scanning
real    1m20.158s
user    2m26.389s
sys     0m24.285s

$ time ./yara 10.yar -r dataset/
real    0m39.236s
user    1m0.937s
sys     0m21.562s
```

Conclusion

Conclusion

- The string part is always evaluated before the condition part
- One slow string can bring down a whole rule or even a ruleset
- Be aware of too general patterns
- Help Yara to search patterns effectively

@regeciovad
dominika.regeciova@gendigital.com

