Writing YARA rules

An introduction to YARA for AIL usage

CIRCL
Computer Incident
Response Center
Luxembourg

Alexandre Dulaunoy
alexandre.dulaunoy@circl.lu
Jean-Louis Huynen
jean-louis.huynen@circl.lu

info@circl.lu

November 26, 2021

Links

- AIL project: `https://github.com/ail-project`
- AIL framework: `https://github.com/ail-project/ail-framework`
- Training materials: `https://github.com/ail-project/ail-training`
- YARA doc: `https://yara.readthedocs.io/en/stable/`
- YARA download: `http://virustotal.github.io/yara/`

What's YARA?

- *The pattern matching swiss knife for malware researchers (and everyone else)*;
- It's an improved **grep** to create pattern matching rule to search for **strings**, **binary patterns**, **regular expressions**;
- A YARA rule can be contextualised with metadata and tags describing a specific set of pattern matching rules.
- Easier definiton of conditions compared to regex.

A sample rule - disneyplus.yara

```
1   rule disney_plus : credential_leak
2 {
3       meta:
4           description = "Finding list of credentials for
      Disney Plus"
5           leak = 1
6       strings:
7           $a = "gmail.com:"
8           $b = "DISNEY_PLUS"
9           $c = "Disney Plus"
10      condition:
11          $a and ($b or $c)
12 }
13
```

Calling yara from command line

- Searching a single file

```
1 yara disneyplus.yara /home/adulau/dataset/2021/09/01/
      nv6RsKFm
2
```

- Searching a directory

```
1 yara disneyplus.yara -r /home/adulau/dataset
      /2021/09/01/
2
```

Regular Expressions

- Regular Expressions (Regex) are extremely useful in extracting information from text
- A regex is a sequence of characters that specifies a search pattern
- They can be used to match, locate extract and replace text

Regular Expressions

You can search for simple letters and specifiy repetition or existence

```
String: ‘Cookie‘

// ‘.‘ Any single character
$re1: /Co.kie/

// ‘*‘ Zero or more for the previous sequence
$re2: /Co*kie/

// ‘+‘ One or more for the previous sequence
$re3: /Co+kie/
```

Regular Expressions

```
1 // ‘{2,3}‘ Between 2 and 3
2 $re4: /Co{2,3}kie/
3
4 // ‘[a-zA-Z]‘ Any letter between ‘a‘ and ‘Z‘
5 $re5: /Co[a-zA-Z]kie/
```

Regular Expressions

Usecase: Email addresses

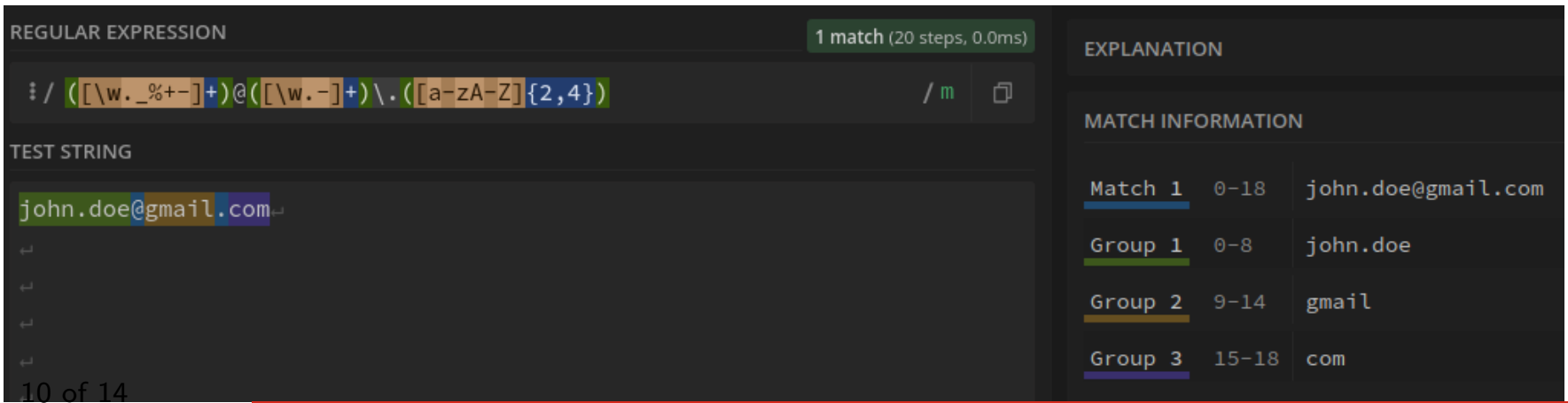
```
$re1: /.+@.+\..+/
// The address `1@.` is valid


$re2: /.+@.+\.[a-zA-Z]{2,4}/
// We enforce a correct TLD (i.e. `.com`)
```

Regular Expressions

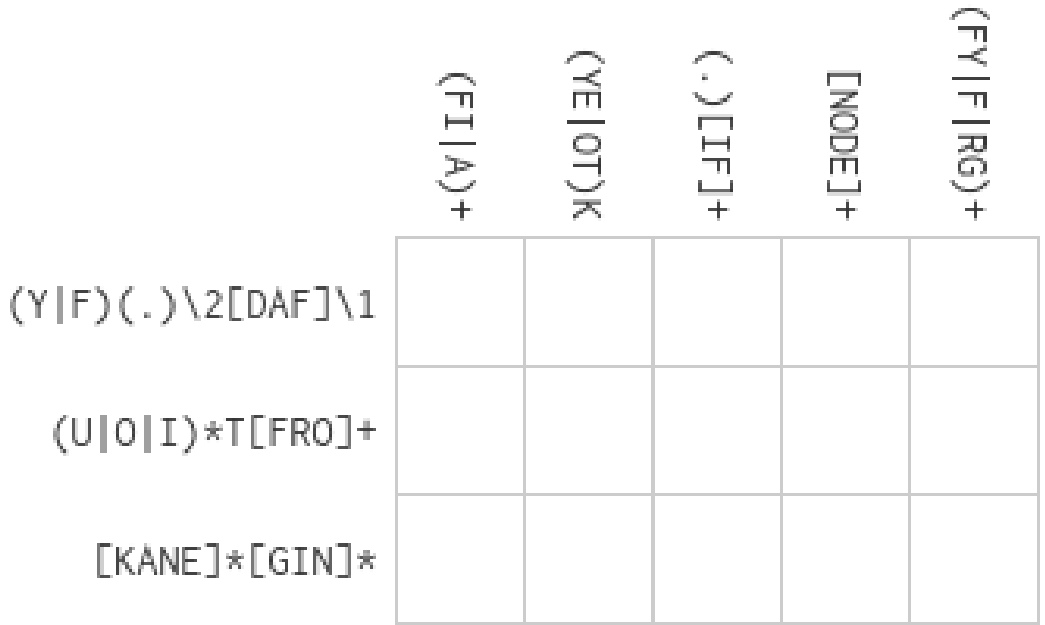
Usecase: Email addresses

```
1 $re2: /.+@[a-zA-Z0-9_.-]+\.[a-zA-Z]{2,4}/
2 // We enforce a correct domain (i.e. ‘gmail‘ or ‘hotmail
      ‘)
3
4 $re3: /[a-zA-Z0-9_.%+-]+@[a-zA-Z0-9_.-]+\.[a-zA-Z]{2,4}/
5 // We enforce a correct email format
6 // ‘!john@doe~@gmail.com‘ is not valid anymore
```



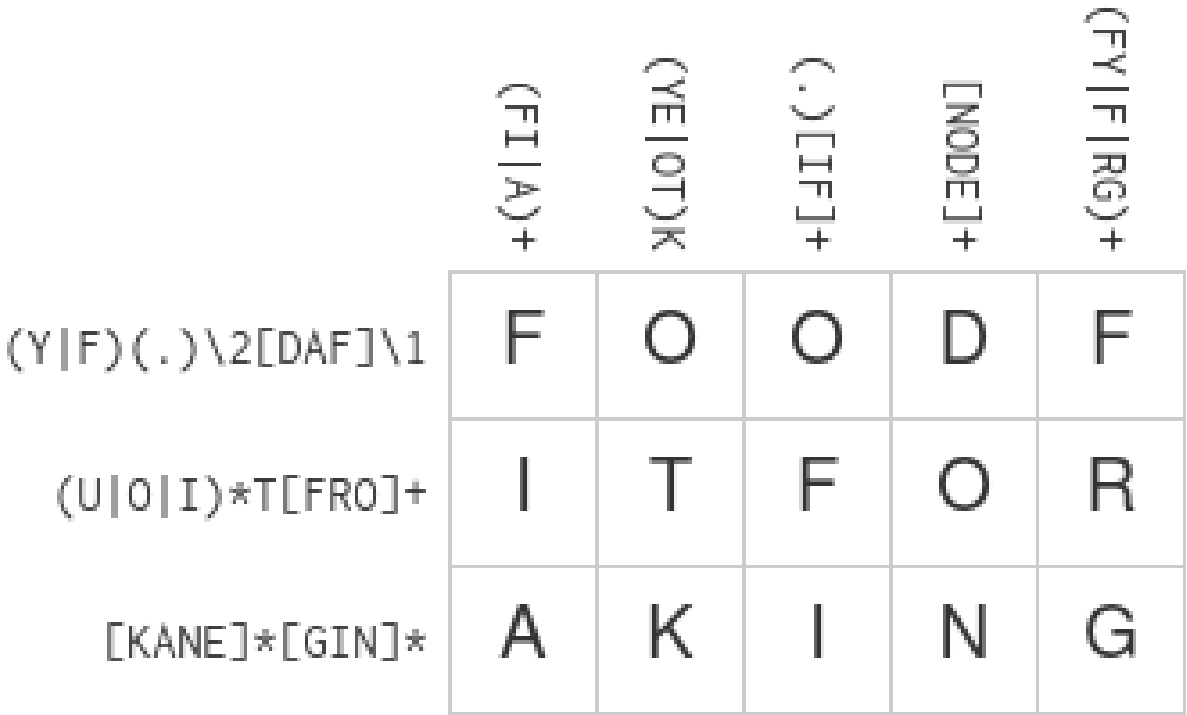
Fun with Regular Expressions

`https://regexcrossword.com/`



Fun with Regular Expressions

`https://regexcrossword.com/`



Searching in binaries

Binaries packed with UPX but made unusable by UPX -d by modifying the magic UPX string:

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0200 3e00 0100 0000 2872 4c00 0000 0000  ..>.....(rL.....
00000020: 4000 0000 0000 0000 0000 0000 0000 0000  @...............
00000030: 0000 0000 4000 3800 0300 4000 0000 0000  ....@.8...@.....
00000040: 0100 0000 0500 0000 0000 0000 0000 0000  ................
00000050: 0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
00000060: 4284 0c00 0000 0000 4284 0c00 0000 0000  B.......B.......
00000070: 0000 2000 0000 0000 0100 0000 0600 0000  .. ............
00000080: 0000 0000 0000 0000 0090 4c00 0000 0000  ..........L.....
00000090: 0090 4c00 0000 0000 0000 0000 0000 0000  ..L.............
000000a0: 7845 4500 0000 0000 0010 0000 0000 0000  xEE.............
000000b0: 51e5 7464 0600 0000 0000 0000 0000 0000  Q.td............
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 1000 0000 0000 0000 deda 8b5f 00ff 9941  ..........._...A
```

```
#!/usr/bin/env python
import sys

def main(srcFilename):
    f = open(srcFilename, 'rb')
    s = open(srcFilename+'_00ff9941', 'wb+')
    header = f.read(0xea)
    s.write(header)
    bindata = f.read()
    f.close()
    bindata = bindata.replace(b'\x00\xff\x99\x41','UPX!')
    s.write(bindata)
    f.close()

if __name__ == '__main__':
    main(sys.argv[1])
```

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0200 3e00 0100 0000 486a 4000 0000 0000  ..>.....Hj@.....
00000020: 4000 0000 0000 0000 0000 0000 0000 0000  @...............
00000030: 0000 0000 4000 3800 0300 4000 0000 0000  ....@.8...@.....
00000040: 0100 0000 0500 0000 0000 0000 0000 0000  ................
00000050: 0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
00000060: 2d7c 0000 0000 0000 2d7c 0000 0000 0000  -|......-|......
00000070: 0000 2000 0000 0000 0100 0000 0600 0000  .. ............
00000080: 0000 0000 0000 0000 0080 4000 0000 0000  ..........@.....
00000090: 0080 4000 0000 0000 0000 0000 0000 0000  ..@.............
000000a0: 7893 2000 0000 0000 0010 0000 0000 0000  x. .............
000000b0: 51e5 7464 0600 0000 0000 0000 0000 0000  Q.td............
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 1000 0000 0000 0000 18b9 39c1 dfdd 3033  ..........9...03
```

```
#!/usr/bin/env python
import sys

def main(srcFilename):
    f = open(srcFilename, 'rb')
    s = open(srcFilename+'_dfdd3033', 'wb+')
    header = f.read(0xea)
    s.write(header)
    bindata = f.read()
    f.close()
    bindata = bindata.replace(b'\xdf\xdd\x30\x33','UPX!')
    s.write(bindata)
    f.close()

if __name__ == '__main__':
    main(sys.argv[1])
```

Searching in binaries

```
1 rule torcryptomining
2 {
3     strings:
4         $upx_erase = {(00 FF 99 41|DF DD 30 33)}
5     condition:
6         $upx_erase at 236
7 }
8
```