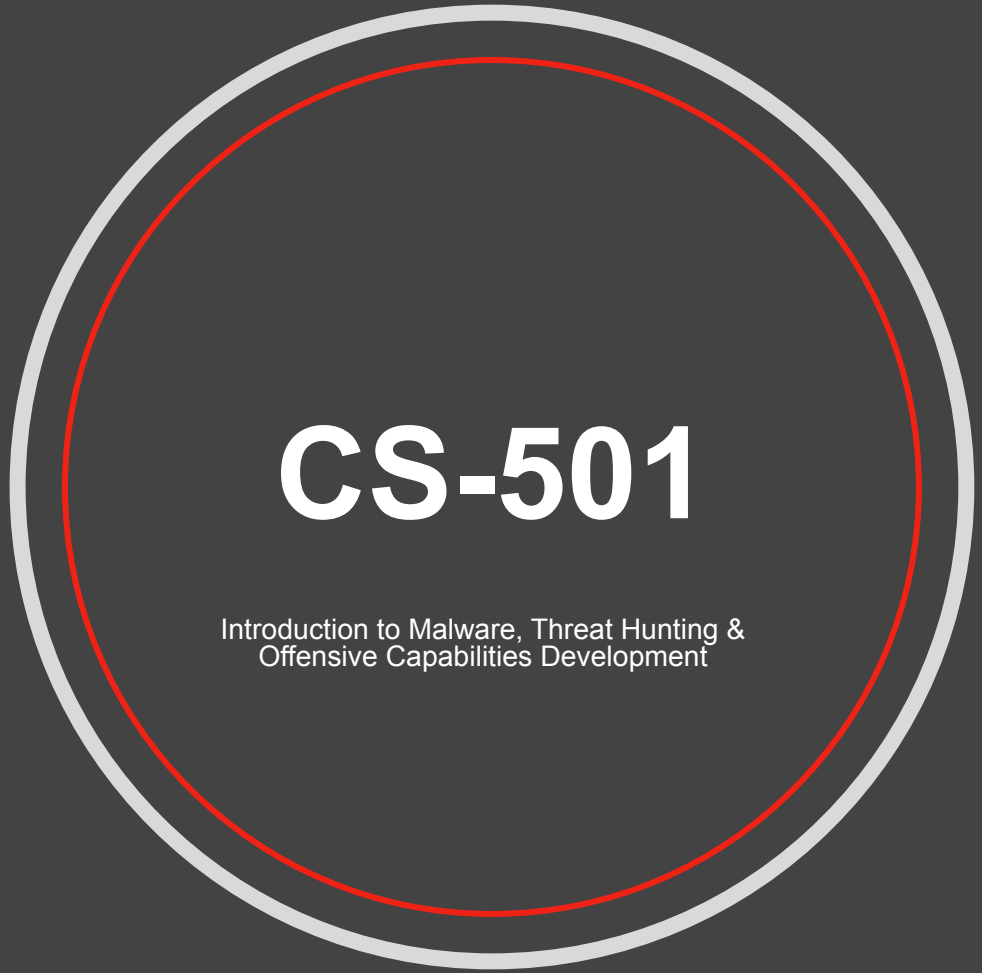


CS-501

Introduction to Malware, Threat Hunting &
Offensive Capabilities Development

Lecture 6: YARA and Threat Hunting

Goal of Threat Hunting

Threat Hunting: Searching environments to detect and isolate malicious activity.

- **Blue Team:** Analysis/Detection of new malicious activity, creating detection rules, attribution.

- **Red Team:** Attribution re: attacker environments, and/or emulation of real malicious activity.

Threat Hunting 101

Step 1: Find IOCs.

Step 2: Pivot from those IOCs to find more IOCs.

Step 3: Look at how the IOCs are similar, or unique.

Step 4: Write detection rules that encompass all the IOCs and are difficult to change, to detect future malicious activity and/or find more activity.

Writing Rules

Detection rules that are high confidence are worth their weight in gold.

People doing this range from Detection and Response teams protecting their own environment, to threat researchers that want to find every instance of a certain malware on the internet.

Threat Hunting 101

Step 1: find IOCs. ✅

Step 2: pivot from those IOCs to find more IOCs.

Threat Hunting 101

Step 1: find IOCs. ✅

Step 2: pivot from those IOCs to find more IOCs.

You only have one malware sample. It’s entirely possible (and very likely) that this isn’t the only sample out there.

Threat Hunting 101

Step 1: find IOCs. ✅

Step 2: pivot from those IOCs to find more IOCs.

What’s a pivot? It’s a shared relationship between two indicators.

Threat Hunting 101

Step 1: find IOCs. ✅

Step 2: pivot from those IOCs to find more IOCs.

What’s a pivot? It’s a shared relationship between two indicators.

Pivot Examples

- Executables with the same importhash
- Executables communicating with domains that resolve to the same IP
- Executables with the same resource/string hash
- WHOIS registration: domains registered using the same email
- YARA rules*
  - Only if they are written well

Note of Caution:

In the real world, it’s easy to pivot out into infinity.

Note of Caution:

In the real world, it’s easy to pivot out into infinity.

Rules of thumb:

- IOCs found from “2 hops out” [i.e. email -> domain, domain -> IP] is likely a safe bet. Anything further than that is low confidence (not as highly likely to be related to your initial IOC set).

- Always check your assumptions
- You should also probably timebox yourself.

IOC lookup

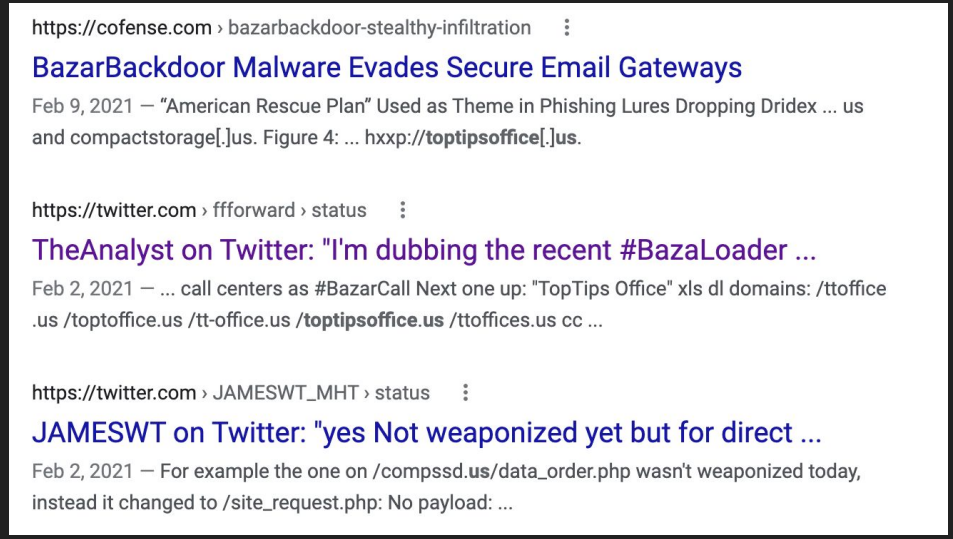
1) Domains/URLs
2) IPs
3) Hashes
4) (and more! i.e. certificates, email addresses, fingerprinted network traffic, etc. Depends on the data you have).

Domains / URLs

1) Google it! [additional context]
2) WHOIS databases [-> IP address]
3) VirusTotal search [-> hashes]
4) Urlscan[.]io -> more context

Example:
https://twitter.com/ffforward/status/1356571665648537601
Ttoffice[.]us, toptipsoffice[.]us

Don't forget to just Google the domains, you might find additional context.

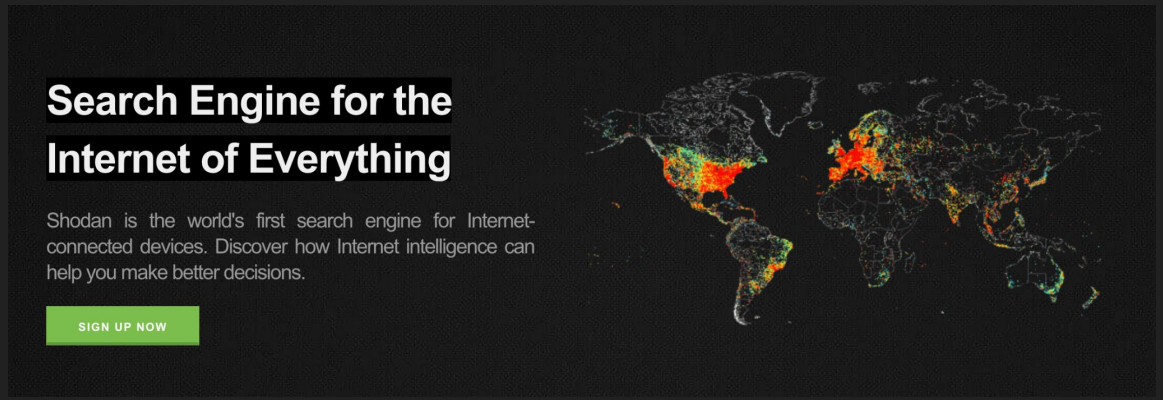


IPs

1) Google it! [additional context]

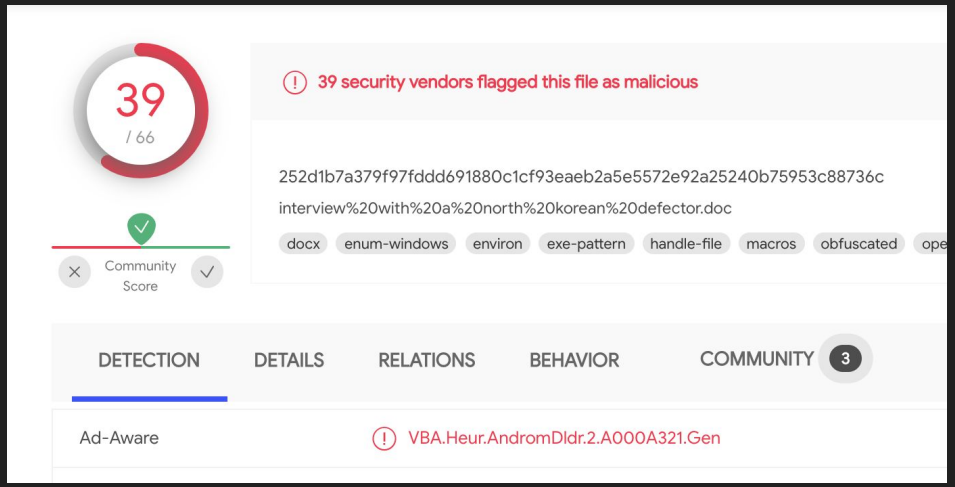
2) VT (“itw” URLs -> “in the wild URLs”)

3) Shodan[.]io (where is it / what’s running)

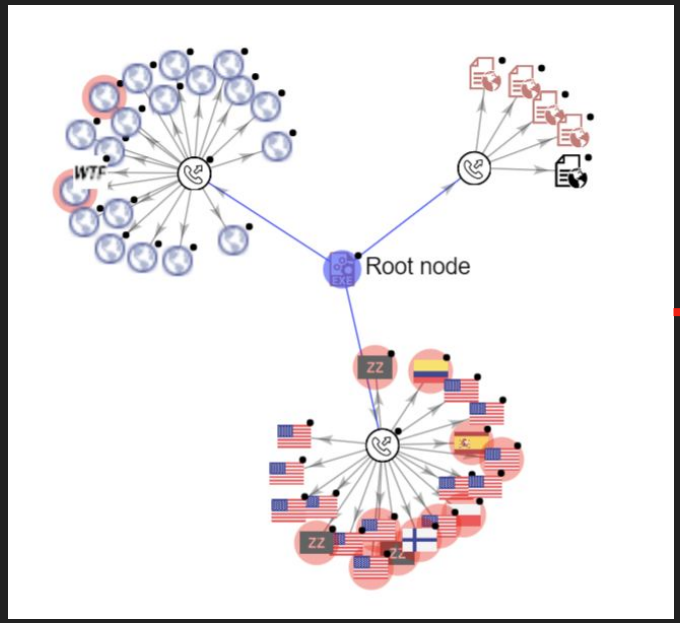


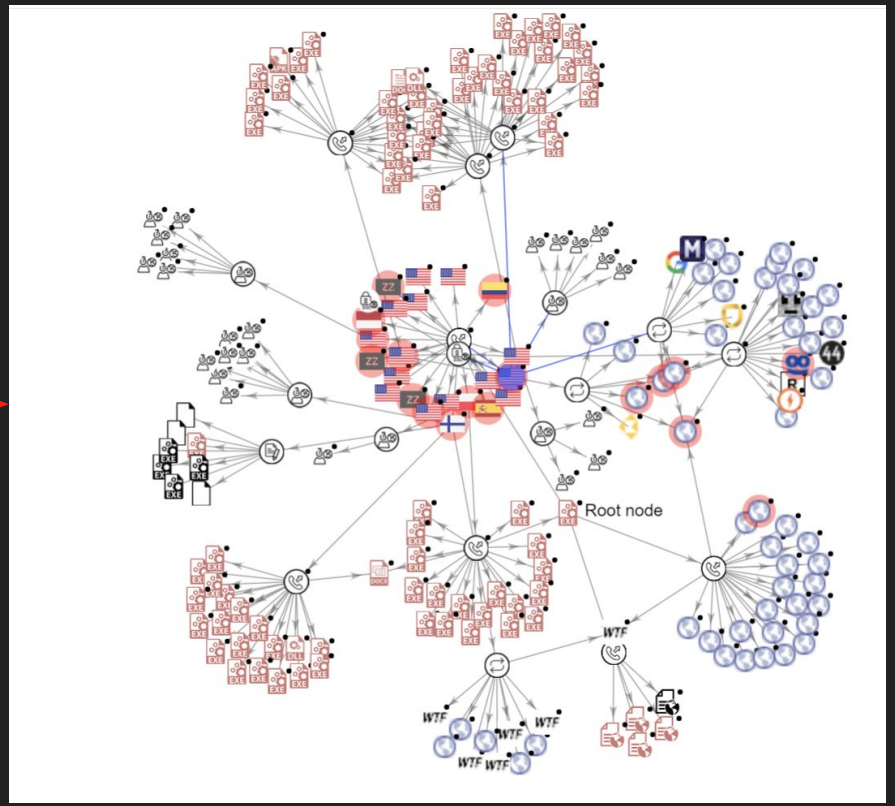
Hashes

1) Google it! [additional context]
2) VirusTotal [hashes -> other IOCs]
3) Antivirus / Sandbox analysis
4) Intezer (function similarity)



Putting it all together





Now to write Detection Rules.

What strings, binary or functions are particularly unique?

Now to write Detection Rules.

What strings, binary or functions look particularly unique?

What similarity is there among your corpus of hashes? Do they all call the same function? Do they all reach out to the same domain?

Now to write Detection Rules.

What strings, binary or functions look particularly unique?

What similarity is there among your corpus of hashes? Do they all call the same function? Do they all reach out to the same domain?

Now to write Detection Rules.

What strings, binary or functions look particularly unique?

What similarity is there among your corpus of hashes? Do they all call the same function? Do they all reach out to the same domain?

How likely are these parts of the hash to change? (domains are easier than functions, for example.)

You want a rule that will have a high true - false positive ratio, and that doesn’t become obsolete quickly.

YARA rules

“Yet another Recursive Acronym”

String compare engine for files/hashes.

```
import "pe"

rule EXT_APT32_goopdate_installer {
  meta:
    reference = "https://about.fb.com/news/2020/12/taking-action-against-hackers-in-bangladesh-and-vietnam/"
    author = "Facebook"
    description = "Detects APT32 installer side-loaded with goopdate.dll"
    sample = "69730f2c2bb9668a17f8dfa1f1523e0e1e997ba98f027ce98f5cbaa869347383"
  strings:
    $s0 = { 68 ?? ?? ?? ?? 57 A3 ?? ?? ?? ?? FF D6 33 05 ?? ?? ?? ?? }
    $s1 = "GetProcAddress"
    $s2 = { 8B 4D FC ?? ?? 0F B6 51 0C ?? ?? 8B 4D F0 0F B6 1C 01 33 DA }
    $s3 = "FindNextFileW"
    $s4 = "Process32NextW"

  condition:
    (pe.is_64bit() or pe.is_32bit()) and
    all of them
}
```

Deploying YARA locally

1. https://yara.readthedocs.io/en/v3.4.0/gettingstarted.html
2. Save your rule file as a .yar
3. `yara [OPTIONS] RULES_FILE TARGET_FOLDER`

Example Yara rule

size (bytes)	file-offset	blacklist (11)	hint (69)	group (9)	value (8366)
5	0x000604E6	-	utility	-	tor l
5	0x00060CBB	-	utility	-	set H
48	0x0007B667	-	utility	-	execute once failure in __cxa_get_globals_fast()
6	0x0007D8C7	-	utility	-	delete
39	0x0007523E	-	url-pattern	-	http://evilevilevilevilevil.cf/evil.exe

Example Yara rule

```
rule bin_ClickMe {
   meta:
      description = " - file ClickMe.exe"
      author = "Kai"
      reference = "Ch0nky"
      date = "2021-09-22"
      hash1 = "bd1a5119785edcefd7f96de1ef58a23bae9bcd00c0e00ae422ee363de90bd7ef"
   strings:
      $s1 = "http://evilevilevilevilevil.cf/evil.exe" fullword ascii
      $s3 = "NotMalware.exe" fullword ascii
      $s5 = "C:\\malware\\ch0nky.txt" fullword ascii

   condition:
      uint16(0) == 0x5a4d and filesize < 2000KB and
      all of them
}
```

YARA

- Rules are composed (usually) of two sections: strings and conditions
- The strings are short sequences that we will search the binary for
- The conditions section consists of boolean conditions on the presences of a subset of the strings

yarGen

Tool for creating Yara Rules.

Follow the directions found here: https://github.com/Neo23x0/yarGen

Create a Virtualenv by running `python -m venv my_yara_env`

- source my_yara_env/bin/activate (Linux)
- my_yara_env/Scripts/activate.ps1 (Windows)

pip install -r requirements.txt

Python yarGen.py --update

Read the docs :-)

yarGen

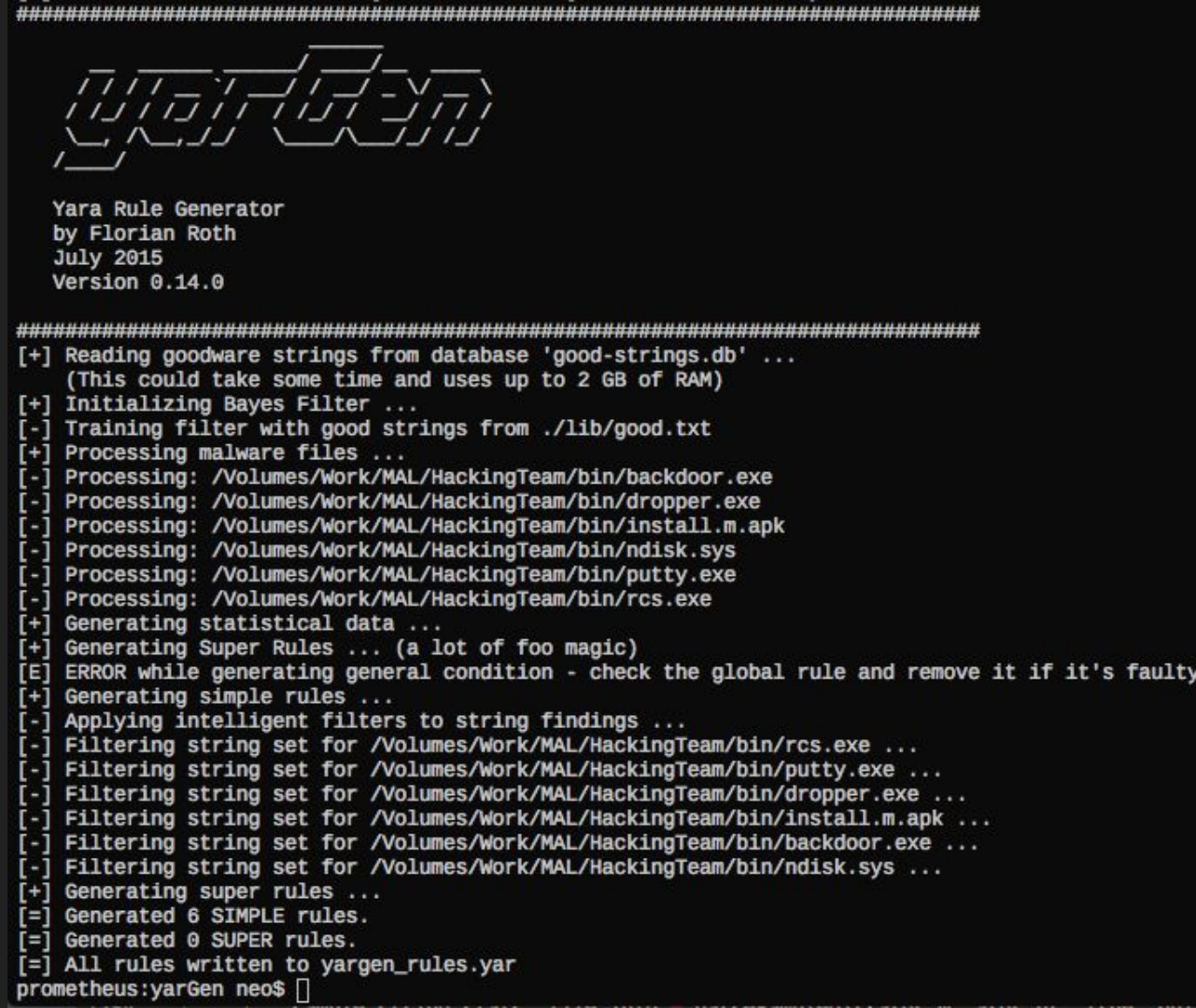
Note that the database is pretty large, and you will likely need a computer with minimum 4GBs of RAM

```
de\Lecture4\bin\ClickMe.exe C
(yara) PS C:\Users\User\Desktop\yarGen-0.23.4\yarGen-0.23.4> python .\yarGen.py --update
--------------------------------------------------------------------------------
                    _____
    __ _____ _____/ ___/__ ___
   / // / _ `/ __/ (_ / -_) _ \
   \_, /\_,_/_/  \___/\__/_//_/
  /___/  Yara Rule Generator
         Florian Roth, July 2020, Version 0.23.3

  Note: Rules have to be post-processed
  See this post for details: https://medium.com/@cyb3rops/121d29322282
--------------------------------------------------------------------------------
Downloading good-opcodes-part1.db from https://www.bsk-consulting.de/yargen/good-opcodes-part1.db ...
Downloading good-opcodes-part2.db from https://www.bsk-consulting.de/yargen/good-opcodes-part2.db ...
Downloading good-opcodes-part3.db from https://www.bsk-consulting.de/yargen/good-opcodes-part3.db ...
Downloading good-opcodes-part4.db from https://www.bsk-consulting.de/yargen/good-opcodes-part4.db ...
```

yargen

https://github.com/Neo23x0/yarGen



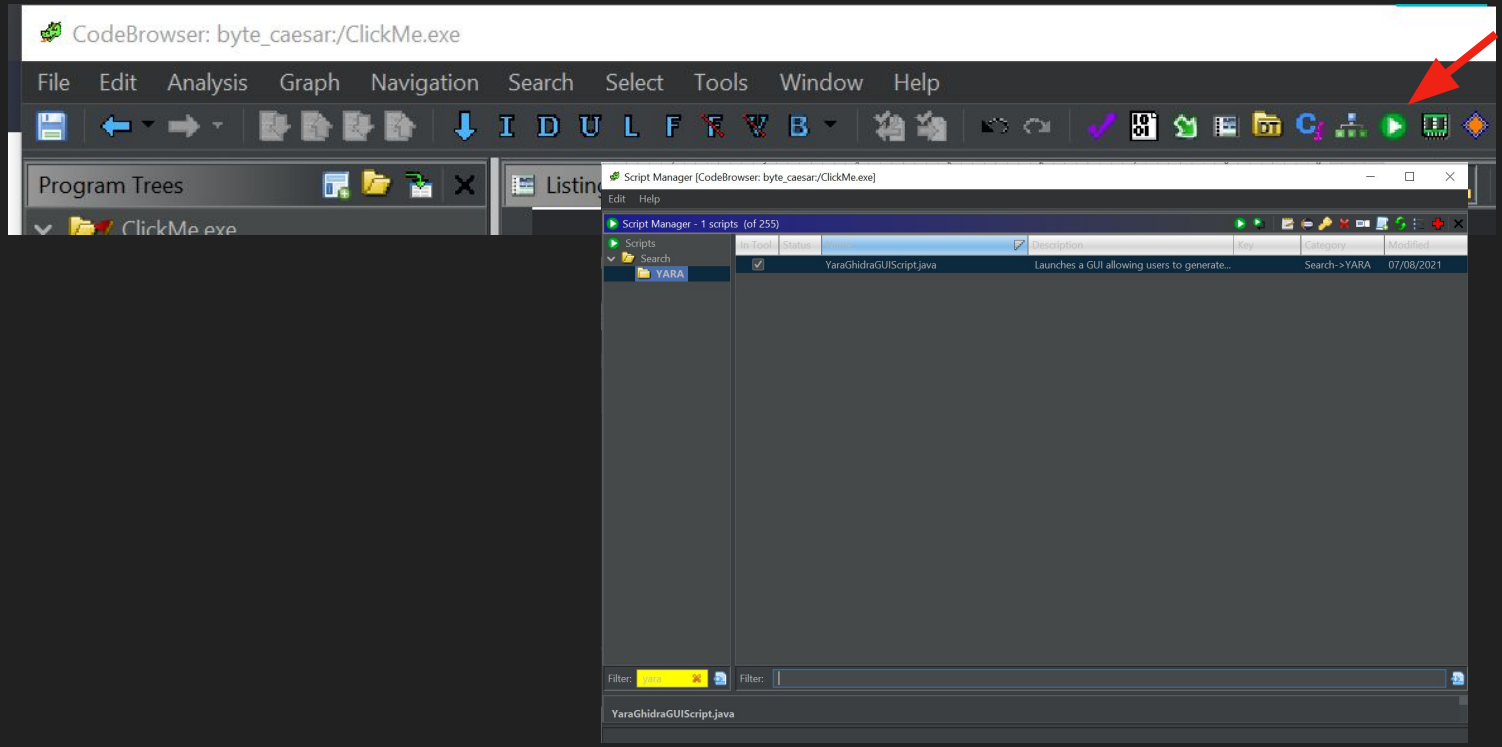
yargen

https://github.com/Neo23x0/yarGen

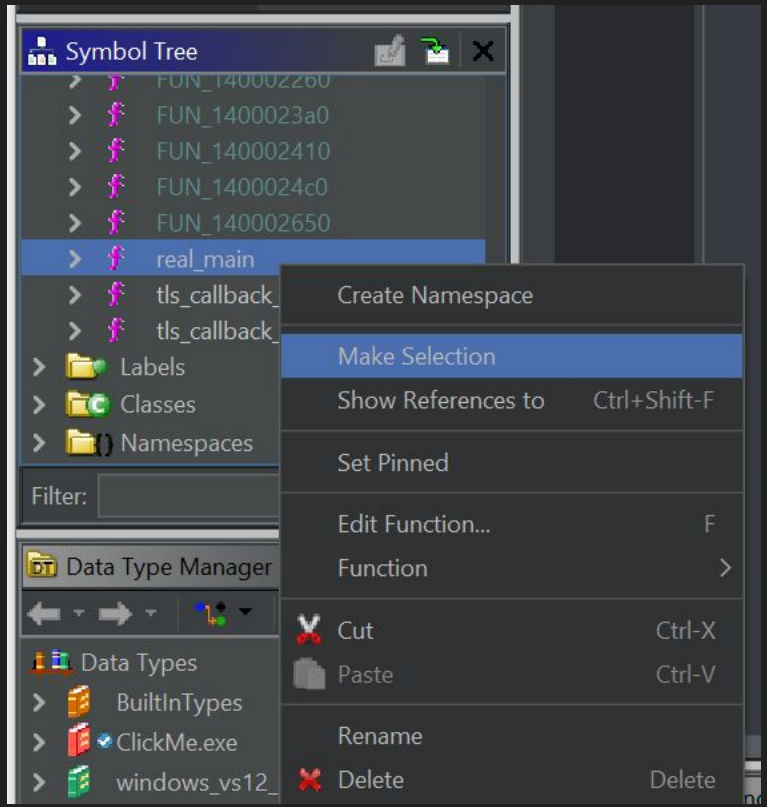
```
python yarGen.py -a [AUTHOR] --score -m
[MALWARE_FOLDER_PATH]

//--score flag shows the scores behind the strings,
aka how malicious/unique they likely are.
```

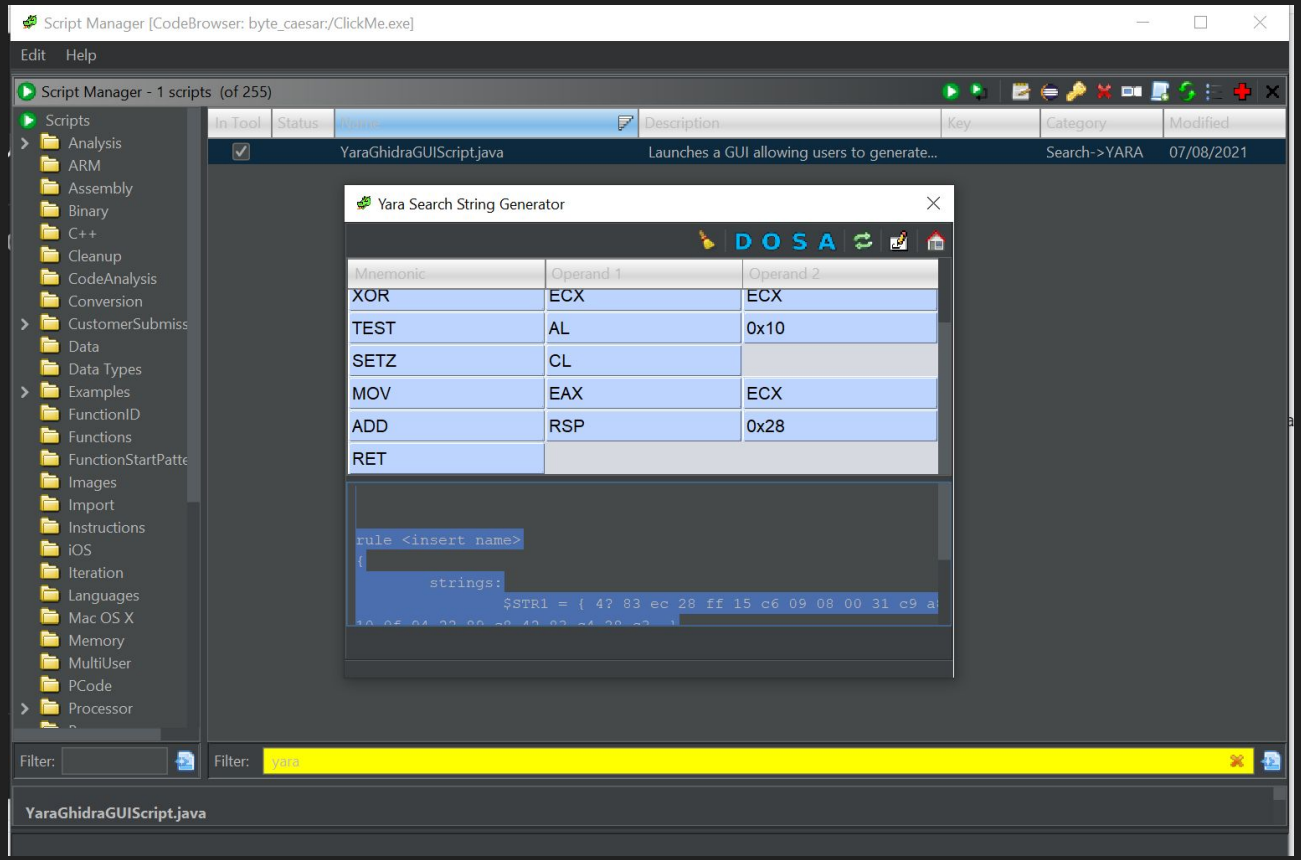
Yara rules with Ghidra Plugin



Yara Rules with Ghidra Plugin



Yara rules with Ghidra Plugin



Resulting rule

```
rule <insert name>
{
    strings:
        $STR1 = { 4? 83 ec 28 ff 15 c6 09 08 00 31 c9 a8 10 0f 94 ?? 89 c8 4? 83 c4 28 c3  }

    condition:
        $STR1 or $STR1
}
```

What makes a good YARA rule?

Interesting strings

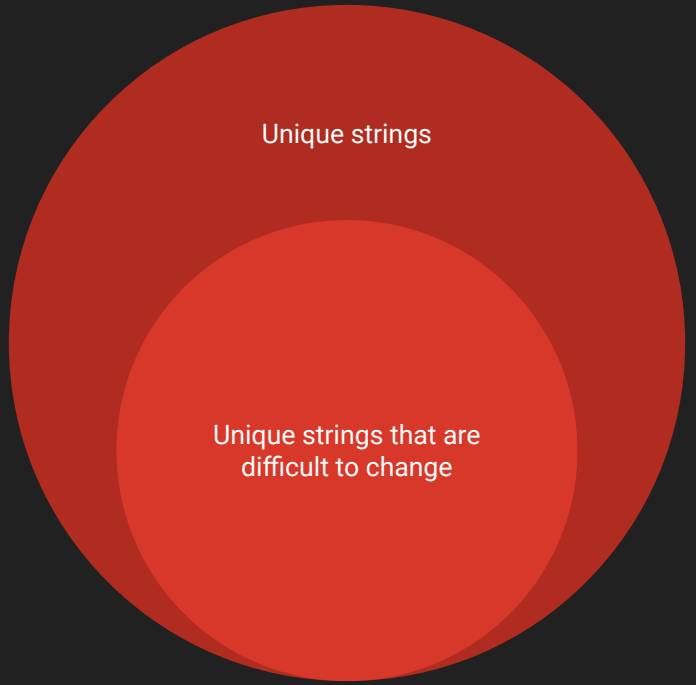
misspellings in comments

C2 domains / IPs

PDB subpaths (/users/AppinSecurity/debug/….)

Opcodes of unique functions (like a custom encryption routine). Make sure to modify the rule if there is a hardcoded key

Import hashes, filetype, (sometimes filesize)



Putting on our Attacker Hat

Suppose we wanted to slow down the defenders.

- Encrypting/obfuscating strings
- Using templates to make compile time changes to functions
- Adding garbage assembly instructions to make opcode signatures harder

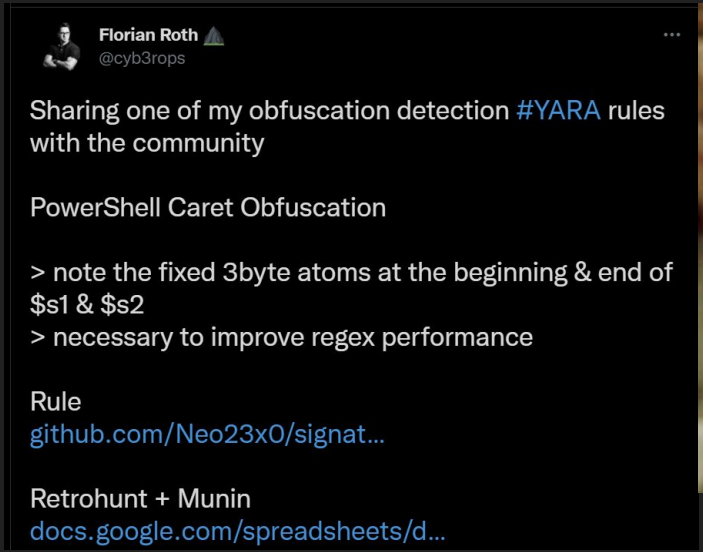
How to Encrypt/Obfuscate Strings at Compile time

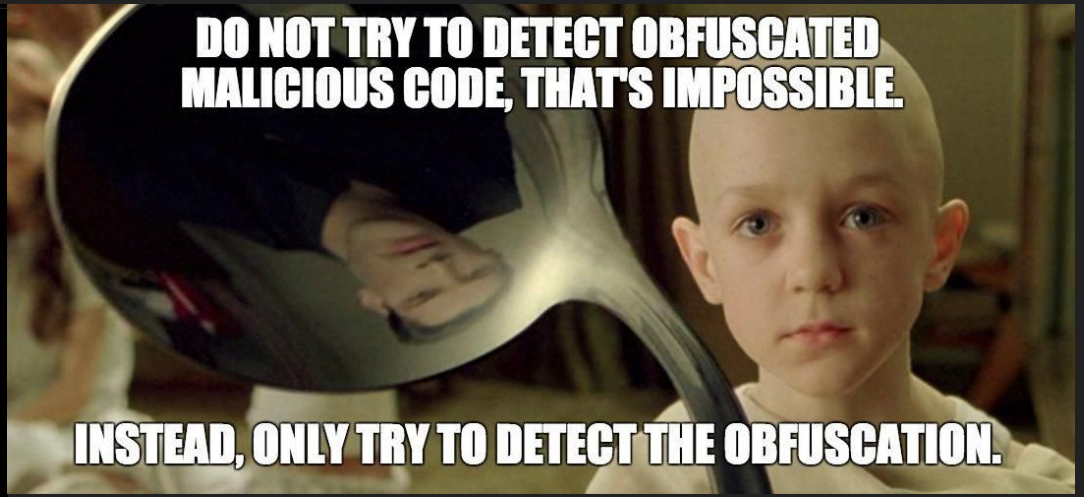
- Templates, constexpr, and seeding a PRNG to store keys in the binary
- Use a scripting language like python to modify source code files with encrypted data
- Store encrypted data as a resource in the binary that can be decrypted at runtime

Simple XOR cipher

- Example: ASCII characters, (utf-8 encoded) with null terminated c strings
  - Remember, that c strings are actually just char pointers that are null terminated.
- For every string literal in our code, we can simple randomly generate bytes equal to the length of the total space in memory the string takes up (this includes the null byte)
- Then we compute the exclusive or of the generated data with the string literal
- We then store the obfuscated data with the key at compile time
- At runtime, the string is recovered by computer XOR(obf_data, rand_data)

Remark


Florian Roth
@cyb3rops
Sharing one of my obfuscation detection #YARA rules with the community
PowerShell Caret Obfuscation
> note the fixed 3byte atoms at the beginning & end of $s1 & $s2
> necessary to improve regex performance
Rule
github.com/Neo23x0/signat...
Retrohunt + Munin
docs.google.com/spreadsheets/d...


DO NOT TRY TO DETECT OBFUSCATED MALICIOUS CODE, THAT'S IMPOSSIBLE.
INSTEAD, ONLY TRY TO DETECT THE OBFUSCATION.

Remarks about Compilers

- Compilers (to me anyway) are black magic
- If you tell it to optimize the your code, it will
- In fact, it will sometimes undo your string encryption if you pass -O3!
- For this reason, you should double check your builds