





Lecture 10:

Threat Hunting with YARA

Goal of Threat Hunting

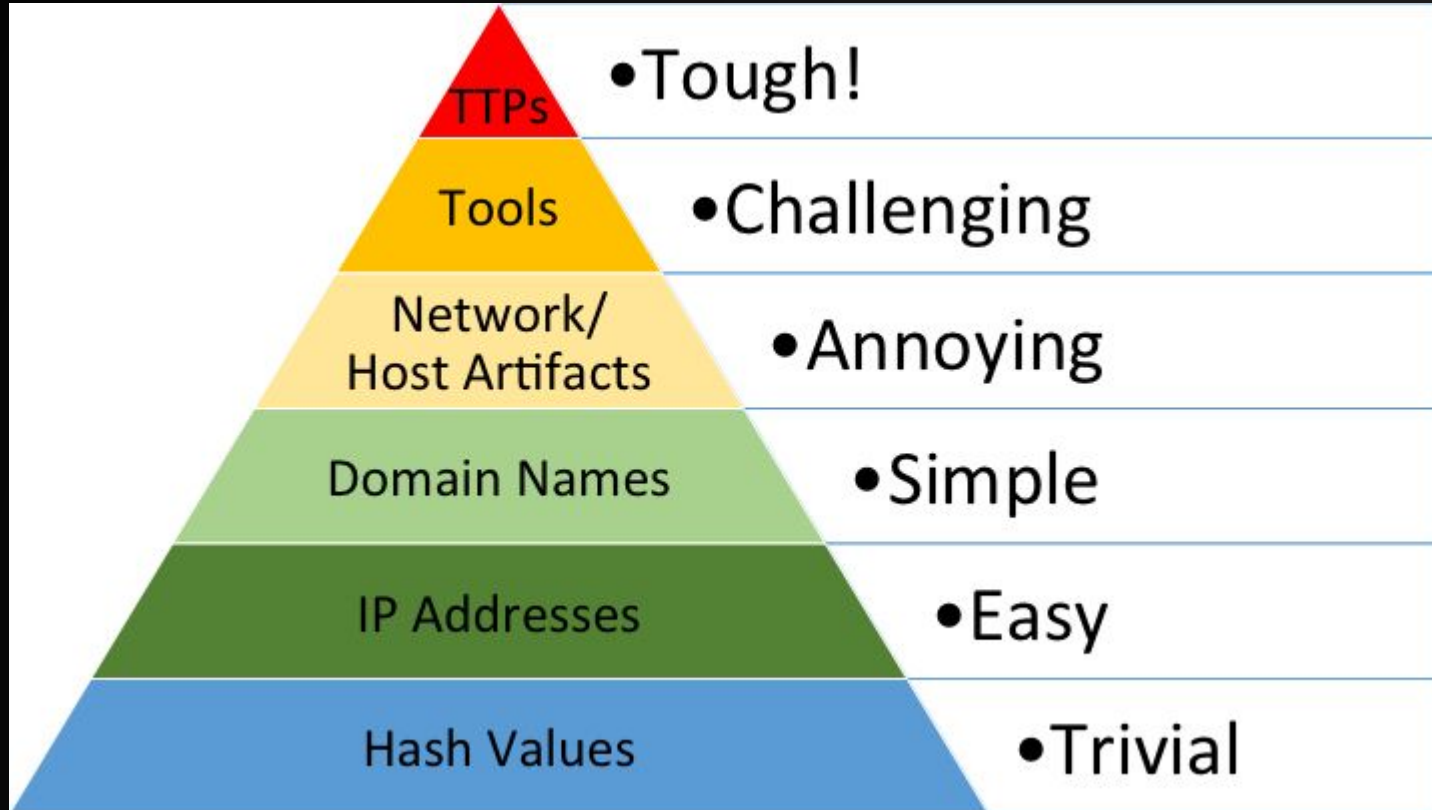
Threat Hunting: Searching environments to detect and isolate malicious activity.

- **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.

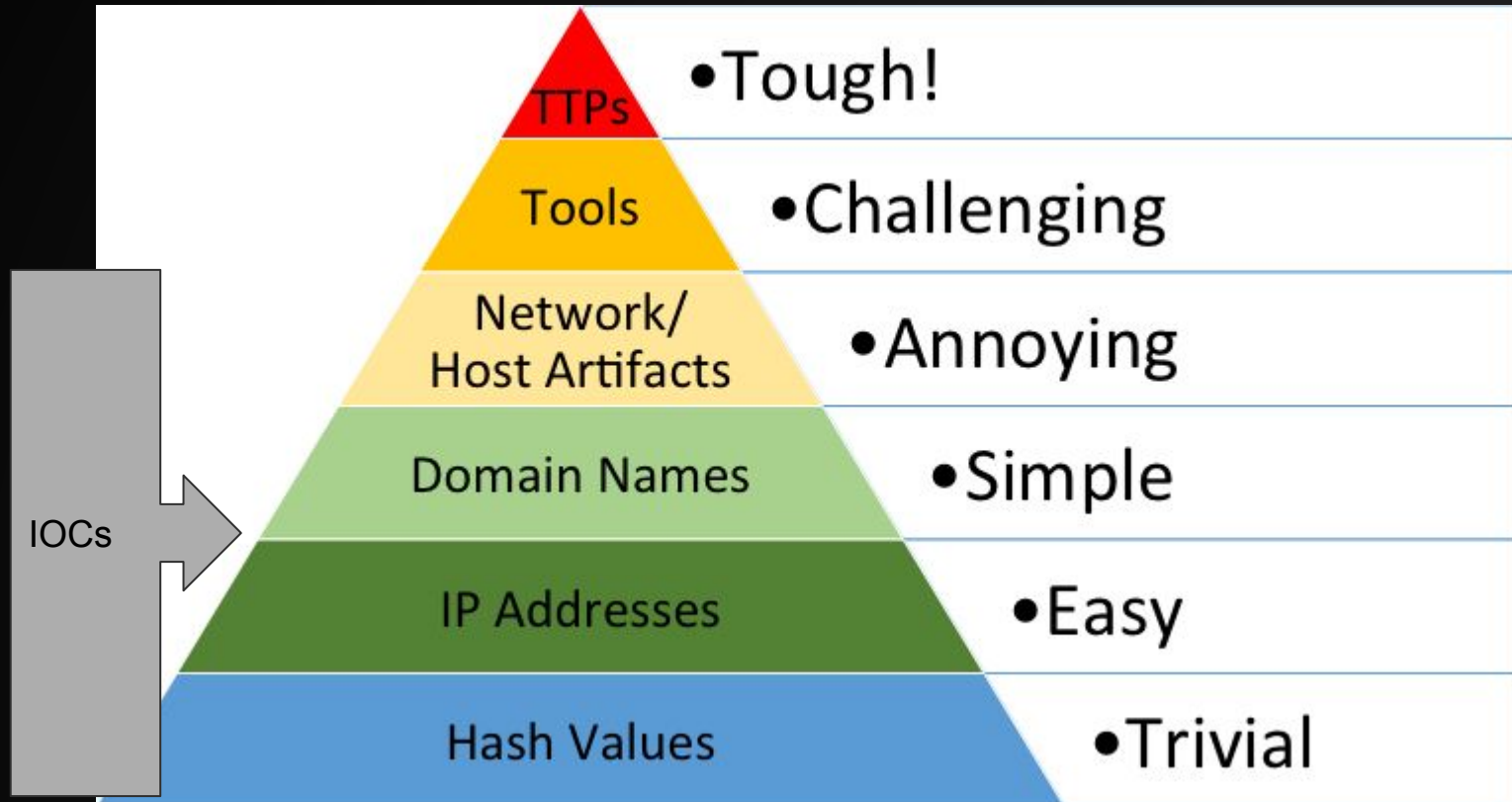
Life of a Blue Teamer/ SOC analyst

- Find Indicators of Compromise
- Move from Indicators of Compromise (IOCs) to -> Tactics, techniques and procedures (TTPs)
- Protecting your Environment from the Tactics / Techniques/ Procedures used.
- Break Links in the “cyber killchain”
- Eventually, threats evolve to combat your countermeasures.
- Go back to step 1 :)

Threat Analysis Pyramid of Pain



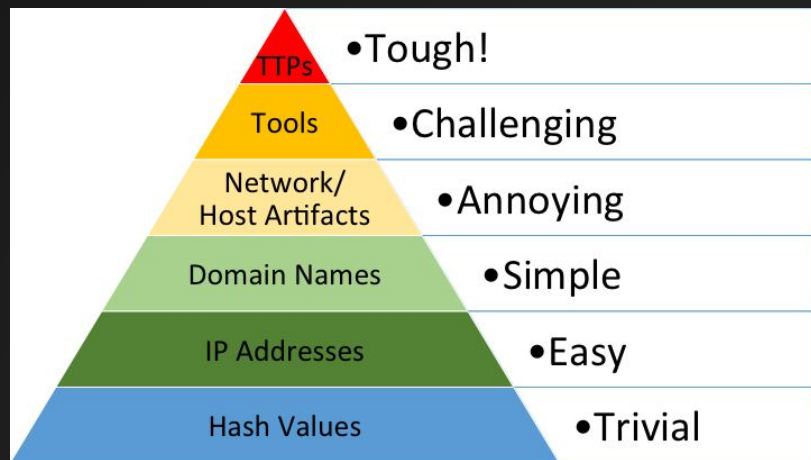
Threat Analysis Pyramid of Pain



Threat Analysis Pyramid of Pain

Getting from IOCs -> TTPs:

- **Hash -> Tools:** What malware family does this hash belong to? How are all these hashes similar? Can I block every instance of the malware?
- **IP/Domain -> Network Artifact:** How does the malware communicate with the C2? Can I look at common patterns in the network traffic and block that behavior?
- **IOC -> TTPs:** Is there a pattern in the way this group conducts operations? Do they drop multiple malware families? Do they look for specific data? How do I block this activity?



Threat Hunting 101

Step 1: Find IOCs.

Step 2: Use IOCs to find more IOCs.

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

Step 4: Write detection rules.

How do we find IOCs?

Depends on who you are!

- AntiVirus vendors detect them using their engines
- Network admins find them either during or after an incident
- Twitter :D

How do we find more IOCs?

Step 1: find IOCs. 

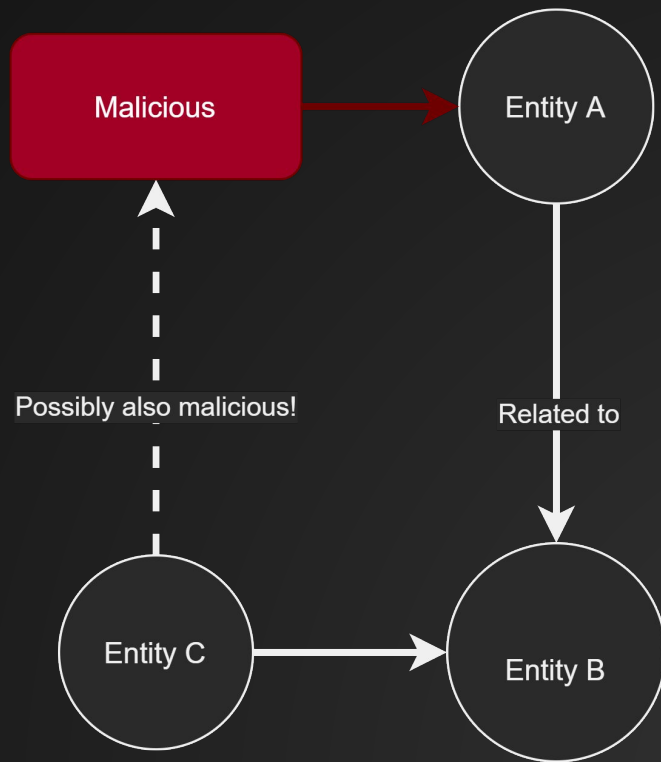
Step 2: Use IOCs to find more IOCs.

Step 2 is achieved by what is called *pivoting*.

Pivoting

A *pivot* is a relationship between two entities.

Usually, when we talk about pivots, we are referring to indicators



Pivot Examples

- Executables with the same hash
- Executables communicating with the same domain or IP
- Executables with the same resource/string
- WHOIS registration: domains registered using the same email
- YARA rules*

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).



Hashes

- Cryptographic hashes:
- Fuzzy Hashes
- Hashes of sections and resources

The screenshot displays a file analysis interface. At the top left, a circular progress indicator shows a score of 39 out of 66. Below this, a green checkmark icon is visible. To the right of the score, a red warning icon and text state: "39 security vendors flagged this file as malicious". Below the score, a green checkmark icon is visible. Below the checkmark, a "Community Score" label is present, flanked by a minus sign and a plus sign. The file's SHA-256 hash is displayed: 252d1b7a379f97fddd691880c1cf93eae2a5e5572e92a25240b75953c88736c. Below the hash, the file name is shown: interview%20with%20a%20north%20korean%20defector.doc. A series of tags are listed below the file name: docx, enum-windows, environ, exe-pattern, handle-file, macros, obfuscated, and open. Below the tags, a navigation bar contains five tabs: DETECTION, DETAILS, RELATIONS, BEHAVIOR, and COMMUNITY. The COMMUNITY tab is selected, indicated by a blue underline and a badge with the number 3. Below the navigation bar, the "Ad-Aware" vendor is listed, with a red warning icon and text: "VBA.Heur.AndromDldr.2.A000A321.Gen".

Domains / URLs / IPs

- 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>
Ttooffice[.]us, toptipsoffice[.]us

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

<https://cofense.com> › bazarbackdoor-stealthy-infiltration ›

BazarBackdoor Malware Evades Secure Email Gateways

Feb 9, 2021 — "American Rescue Plan" Used as Theme in Phishing Lures Dropping Dridex ... us and compactstorage[.]us. Figure 4: ... hxxp://toptipsoffice[.]us.

<https://twitter.com> › ffforward › status ›

TheAnalyst on Twitter: "I'm dubbing the recent #BazaLoader ...

Feb 2, 2021 — ... call centers as #BazarCall Next one up: "TopTips Office" xls dl domains: /ttooffice.us /toptoffice.us /tt-office.us /toptipsoffice.us /ttoffices.us cc ...

<https://twitter.com> › JAMESWT_MHT › status ›

JAMESWT on Twitter: "yes Not weaponized yet but for direct ...

Feb 2, 2021 — For example the one on /compsd.us/data_order.php wasn't weaponized today, instead it changed to /site_request.php: No payload: ...



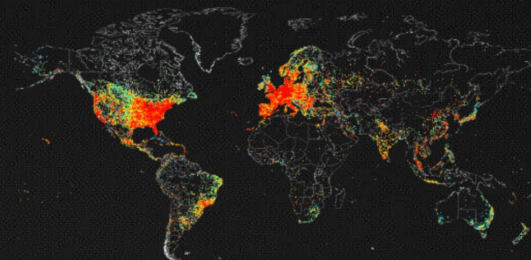
IPs

- 1) Google it! [additional context]
- 2) VT (“itw” URLs -> “in the wild URLs”)
- 3) Shodan[.]io (where is it / what’s running)

Search Engine for the Internet of Everything

Shodan is the world's first search engine for Internet-connected devices. Discover how Internet intelligence can help you make better decisions.

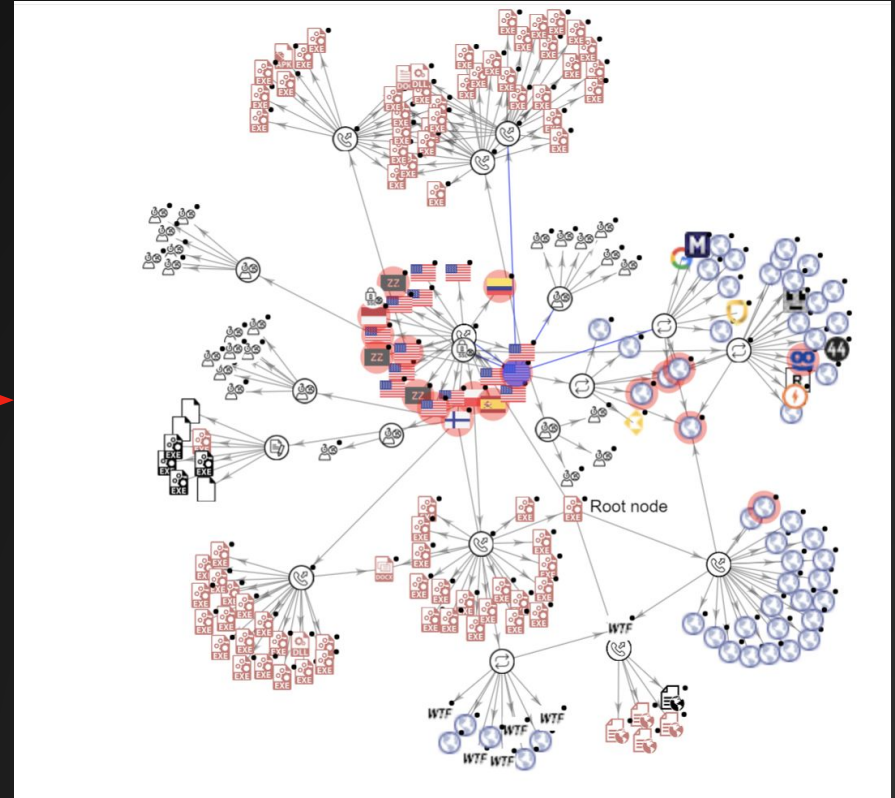
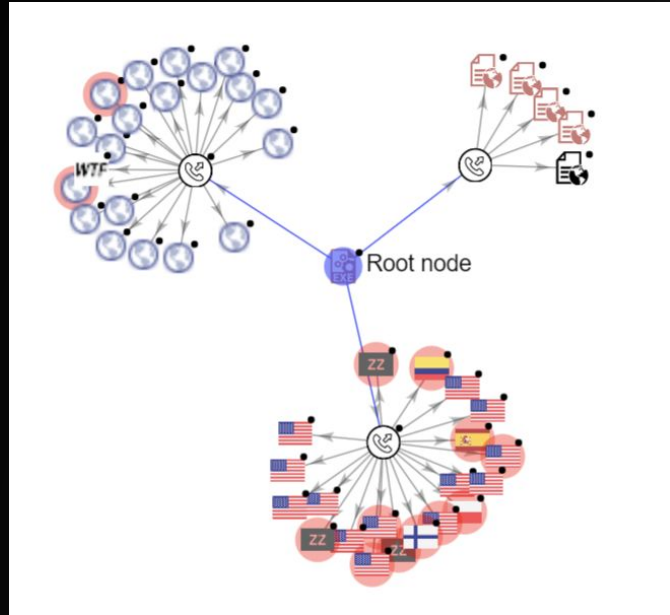
[SIGN UP NOW](#)



Note of Caution:

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

Putting it all together



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.

Common Footguns/confounding factors

- Shared hosting/NAT: two servers/services can share the same IP!
- Hacked servers: compromised servers are very common for payload staging/tier 1 C2s
- Rapid IP rotation: i.e. Kubernetes
- Dynamic DNS: Duckdns.org, chickenkiller.com...etc

Now to write Detection Rules.

What strings, binary or functions are particularly unique?

I.e., we want rules that


- encompass IOCs
- are difficult to change
- detect future malicious activity
- Aid in identifying previous activity

Writing Detection 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.

Example:

b80947799ba2f5416a9077fc22ad3d63951d209f7fe67b27f72938612cfa0e9e

Vt/Malware Bazaar

<https://bazaar.abuse.ch/sample/b80947799ba2f5416a9077fc22ad3d63951d209f7fe67b27f72938612cfa0e9e/>

What pivots are available to us?

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.

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”

Tool and language for detecting *static* content in files

YARA will be our primary tool for threat hunting to start

YARA rules

“Yet Another Recursive Acronym”

Basically, it is extra spicy RegEx

```
1  import "pe"
2
3  rule EXT_APT32_goopdate_installer {
4      meta:
5          reference = "https://about.fb.com/news/2020/12/taking-action-against-hackers-in-bangladesh-and-vietnam/"
6          author = "Facebook"
7          description = "Detects APT32 installer side-loaded with goopdate.dll"
8          sample = "69730f2c2bb9668a17f8dfa1f1523e0e1e997ba98f027ce98f5cbaa869347383"
9      strings:
10         $s0 = { 68 ?? ?? ?? ?? 57 A3 ?? ?? ?? ?? FF D6 33 05 ?? ?? ?? ?? }
11         $s1 = "GetProcAddress"
12         $s2 = { 8B 4D FC ?? ?? 0F B6 51 0C ?? ?? 8B 4D F0 0F B6 1C 01 33 DA }
13         $s3 = "FindNextFileW"
14         $s4 = "Process32NextW"
15
16     condition:
17         (pe.is_64bit() or pe.is_32bit()) and
18         all of them
19 }
```


Getting started with YARA

Read the docs.

<https://yara.readthedocs.io/en/v3.4.0/gettingstarted.html>

Seriously, read the docs. Reading documentation is one of the most important skills you can have in information security.

YARA

- Rules are composed (usually) of two sections: strings and conditions
 - Optionally you can include metadata
- The strings are short sequences
- The conditions section consists of boolean conditions on the presences of a subset of the strings

Example Yara rule

```
rule Ch0nkyCheck
{
    meta:
        description = "Common file Ch0nkybear malware searches for"
        author = "Kbsec"
        reference = "https://github.com/kbsec/CS-501-malware-course-public"
        date = "2022-3-01"
    strings:
        // this is a very unique path that seems to show up everywhere ch0nky is deployed!
        $path = "C:\\malware\\ch0nky.txt"

    condition:
        $path
}
```

Using YARA

READ THE DOCS. Simple usage is `yara.exe rule0.yar rule.yar path/to/folder/with/data`

Modules:

Allows for extension of features

Want to implement rules on custom data structures/types?

Want to write rules to express complex conditions and package them for public use?

Want to create “dry code” but for yara rules?

Modules: PE

- Fine grained rules for Portable Executables and DLLs
- PE files contain a lot of information that we can parse out using YARA's PE module
- Imphash : returns the impash of a pe (md5 of the import address table)
- section_index(name) : useful for parsing sections
- exports(function_name): check if the PE exports a function

Modules: Hash

Module containing functionality supporting cryptographic hash functions

This is usually best used with the PE module, to identify and hash sequences of data

What makes a good YARA rule?

Interesting strings

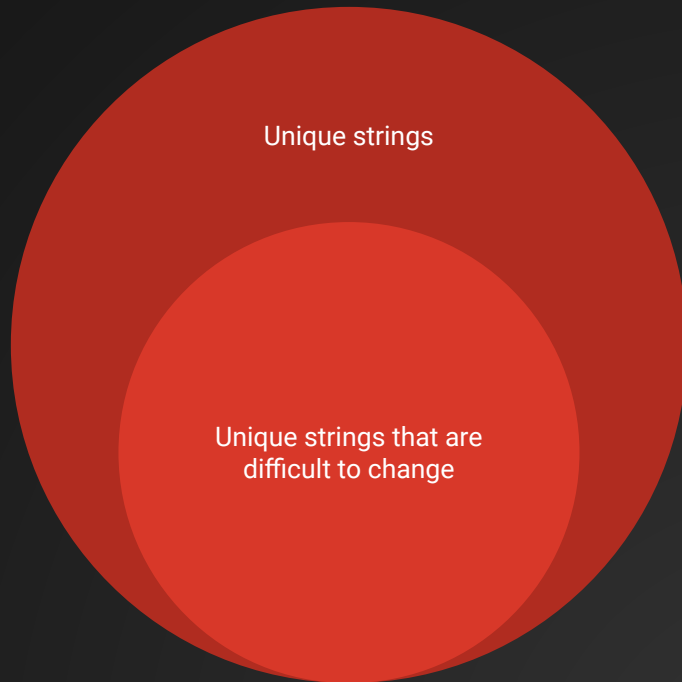
misspellings in comments

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

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)

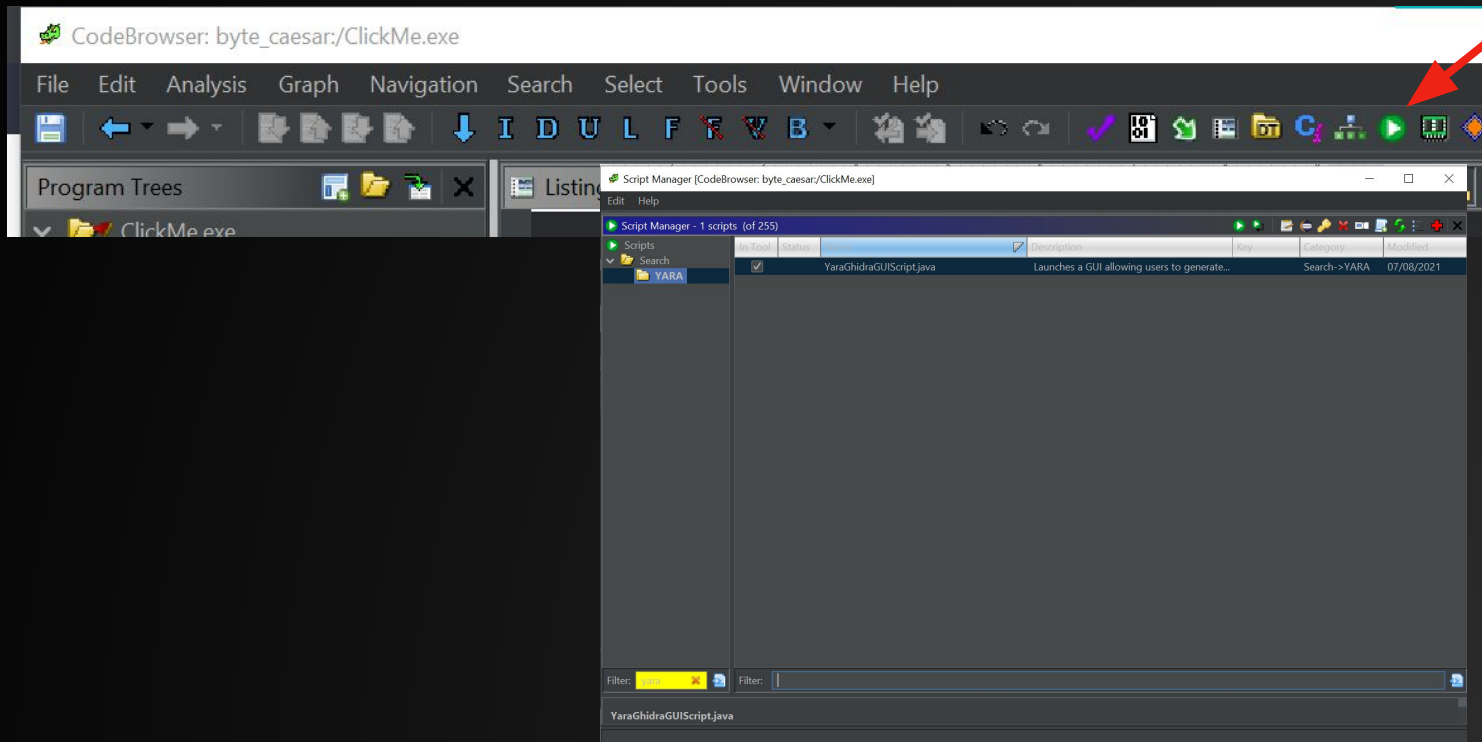
Remark

Detecting content that is obfuscated is difficult

Detecting obfuscation is actually a lot easier!



Yara rules with Ghidra Plugin



Mnemonic	Operand 1	Operand 2
PUSH	RBP	
MOV	RBP	RSP
SUB	RSP	0x30
MOV	qword ptr [RBP + 0x10]	RCX
MOV	qword ptr [RBP + 0x18]	RDX
MOV	qword ptr [RBP + 0x20]	R8

```
rule <insert name>
{
    strings:
        $STR1 = { 55 4? 89 e5 4? 83 ec 30 4? 89 4d 10 4? 89 55 18 4? 89 45 20 4? 8b 45 20 4? 89 c1 e8 fc 5e 00 00 4? 89 45
4? c7 45 f8 00 00 00 eb 30 4? 8b 55 18 4? 8b 45 f8 4? 01 d0 0f b6 08 4? 8b 55 10 4? 8b 45 f8 4? 01 d0 0f b6 10 4? 8b 45 f0 4?
8b 45 f8 4? 01 c0 31 ca 88 10 4? 83 45 f8 01 4? 8b 45 f8 4? 3b 45 20 72 c6 4? 8b 45 f0 4? 83 c4 30 5d c3  }

    condition:
        $STR1 or $STR1
}
```

Using Ghidra to Create Rules

```
rule <insert name>
{
    strings:
        $STR1 = { 55 4? 89 e5 4? 83 ec 30 4? 89 4d 10 4? 89 55 18 4? 89 45 20 4? 8b 45 20 4? 89 c1 e8
fc 5e 00 00 4? 89 45 f0 4? c7 45 f8 00 00 00 00 eb 30 4? 8b 55 18 4? 8b 45 f8 4? 01 d0 0f b6 08 4? 8b 55
10 4? 8b 45 f8 4? 01 d0 0f b6 10 4? 8b 45 f0 4? 8b 45 f8 4? 01 c0 31 ca 88 10 4? 83 45 f8 01 4? 8b 45 f8
4? 3b 45 20 72 c6 4? 8b 45 f0 4? 83 c4 30 5d c3  }

    condition:
        $STR1 or $STR1
}
```

Detecting the XOR cipher

In this way, we detect the static content

```
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> python .\xor.py .\conf.h.base conf.h
OK
[+] Wrote conf.h
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> gcc .\main.c -o b.exe
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> yara64.exe .\ch0nky_ghid.yar .
xor_rule .\b.exe
xor_rule .\a.exe
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> █
```


Remarks about Compilers

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

```
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> zig cc -O3 .\main.c -o d.exe
```

```
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> .\d.exe
```

```
"http://evilevilevil.com"
```

```
"C:\\malware\\ch0nky.txt"
```

```
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> strings.exe .\d.exe | Select-String "http"
```

```
"http://evilevil"C:\\malware\\ch
```

```
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> █
```

Compilers cont

- Different compilers produce different results for the same C code
- Can you think of a way to improve this YARA rule?

```
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> gcc main.c -O3 -o c.exe
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> .\c.exe
"http://evilevilevil.com"

"C:\\malware\\ch0nky.txt"

PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> .\c.exe ^C
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> strings.exe .\c.exe | Select-String ch0nk
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> yara64.exe .\ch0nky_ghid.yar .
xor_rule .\b.exe
xor_rule .\a.exe
PS C:\dev\CS-501-malware-course\LectureCode\lecture_10> █
```