Using YARA tags to build a heuristic scanner

Jason Reaves
Threat Research, SentinelOne
sysopfb@gmail.com

ABSTRACT

YARA is a tool that has been pretty heavily adopted within the Cyber Security community, it was built to aid malware researchers with identifying and classifying malicious objects[1]. Instead of approaching this problem with a purely good or bad mindset in detecting malicious objects, we can utilize added functionality of YARA, namely tags, to approach the problem of judging how malicious or suspicious an object is by looking at the problem in smaller sets. This concept is commonly used in heuristic engines used by antiviruses and sandboxes where you can give a weight of maliciousness to an object. The aim of this paper is to introduce a method for such an engine to be built by an organization utilizing existing software.

Keywords

Malware; YARA; Heuristics; Scanning

1. Introduction

A common approach with detecting something is to hone in on what makes it unique is this can sometimes involve very large and very specific signatures. This is done because the malware researcher is looking for something specific, whatever that may be they will usually generate a narrow scoped signature and a broad signature this lets you tune your YARA rules in a more manageable fashion but you can end up repeating the process multiple times. Making a more generic signature based on possible suspicious or malicious indicators allows for quicker generation of signatures for detection purposes but also lowers your ability to accurately detect something based on name, instead you are writing detections based on techniques and tactics or other suspicious indicators that can then allow you to assign a score to something. This also allows you to classify something based on its capabilities and is a technique antivirus companies use called heuristic

scanning in order to try and detect new or unknown malware. A normal company however doesn't need to rely on external software in order to perform this type of scanning, a single malware researcher tracking the families targeting a company can easily generate such signatures on a daily basis at various levels, by levels I refer to creating signatures for a standalone scanning engine in this case YARA to scan for example all inbound email attachments for suspicious or malicious indicators, all outbound emails for possible data leaking(DLP), all executable files transferred around the network and can also can be written to target memory scanning from sandbox executions[4,5,6] or forensics[7].

2. Heuristic Engine

For our purposes we are wanting an engine that we can utilize for scanning files, this engine should accept easily creatable rules. The rules for the engine should be able to have meta data in them that can be used for concise descriptions of the hits on a file along with a heuristic score that can applied for each hit, they should also be easily customizable. As it turns out an existing engine already exists geared towards malware identification and classification; YARA. YARA also comes with the ability to have meta data inside its rules, so the research that follows is a case study on how to use YARAs capabilities along with a little development work to get a very easy to use and easy to expand heuristics scanning engine.

This engine will simply be a python script with a YAML configuration and a collection of generated YARA rules that will live in sub directories with the main YARA rules referenced in the YAML configuration file controlling importing the signature files from the subdirectories.

3. YARA Tags

Adding tags to YARA rules is a feature that was added in order to be able to filter the output of running your rules, in order to accomplish our heuristic scanning we use it for a similar but slightly different purpose. We will be using tags to give short and concise descriptions about the rule, combining this with the meta data that can be added into a YARA rule allows us to expand the usage in a variety of ways. Instead of using tags we could have more descriptive data inside the meta data section to be used for a similar purpose but the short nature of tags fit our needs of having a concise description.

4. META data

YARA allows for meta data to be included in its rules, it's intended use is for storing additional information about the rule[]. The additional information is stored in a identifier value method separated by an equal sign, our additional information will be the score value that we want to assign when a file is detected by a rule.

```
rule xor_encoded_FindKernel32 : XOR_FindKernel32
{
meta:
        score = 20
```

This allows us to build a label system that we will be able to collect and output in the event of a file exceeding our score system.

5. Building the engine

The pseudocode for our engine is pretty straightforward:

```
#Engine psuedocode

args = parse_arguments()
conf = load_config(args.conf_file)
scan_dir = args.scan_dir
all_rules = load_all_rules(conf['scans'])

all_files = get_all_files(scan_dir)

for file in all_files:
        scanObj = ScanFile(file)
        scanObj.run(all_rules)
        if scanObj.get_score() > 20:
                print(scanobj)
```

Figure 1 Engine psuedocode

We will have an argument system and a single configuration file, the configuration file will simply contain a list of YARA rules that we will import. An example of this would be common windows API function names that are XOR encoded, such a trait could mean a packed or

obfuscated file which is not a malicious trait by itself but can be suspicious. The level of suspiciousness is completely dependent on each environment that you are scanning in, in certain environments such a trait could be so suspicious that it is deemed indistinguishable from malicious and is then elevated to a malicious trait. These sort of determinations require individuals familiar with the environment to make and could be adjusted continually as time goes on.

```
scans:
- apis
```

In the config is a string ‘apis’ which will line up with a YARA file sitting in predetermined location and named ‘apis.yar’. This allows the scanning engine to simply enumerate everything in the configuration file to load all the scanning rules but also allows flexibility in regards to turning on and off rulesets.

```
include "encoded_apis/getprocaddress.yar"
include "encoded_apis/virtualalloc.yar"
include "encoded_apis/loadlibrarya.yar"
include "encoded_apis/getmodulehandlea.yar"
include "encoded_apis/outputdebugstringa.yar"
```

Figure 2 APIs Yara

Inside the ‘apis.yar’ file is a collection of includes to load all the associated rulesets, this once again makes the entire system more extensible.

```
heur_scan.py
scan_config.yaml
apis.yar
encoded_apis\
- getprocaddress.yar
- virtualalloc.yar
- loadlibrarya.yar
- getmodulehandlea.yar
- outputdebugstringa.yar
```

Figure 3 Scanner directory layout

This proposed structure for the demonstration purposes of this paper can be seen in figure 6. A few examples of rules would be to look for XOR encoded API calls such as GetProcAddress and LoadLibraryA or XOR encoded RSA keys.

```
rule xor_encoded_loadlibrarya : XOR_LoaderLibraryA
{
meta:
        score = 10
strings:

        $1 = {4d 6e 60 65 4d 68 63 73 60 73 78 40}
        $2 = {4e 6d 63 66 4e 6b 60 70 63 70 7b 43}
        $3 = {4f 6c 62 67 4f 6a 61 71 62 71 7a 42}
        $4 = {48 6b 65 60 48 6d 66 76 65 76 7d 45}
        $5 = {49 6a 64 61 49 6c 67 77 64 77 7c 44}
        $6 = {4a 69 67 62 4a 6f 64 74 67 74 7f 47}
        $7 = {4b 68 66 63 4b 6e 65 75 66 75 7e 46}
        $8 = {44 67 69 6c 44 61 6a 7a 69 7a 71 49}
        $9 = {45 66 68 6d 45 60 6b 7b 68 7b 70 48}
```

Figure 4 Sniffer of YARA file

```
rule single_xor_RSA1
{
strings:
$rsa1 = "\x06\x02\x00\x00\x00\xa4\x00\x00RSA1\x00\x04\x00\x00\x01\x00\x01\x00" xor
condition:
any of them
}
```

Figure 5 Newer YARA keyword XOR

With the framework lined out it's input over time becomes YARA rules, the flexibility of a scoring system allows the user to create a broad range of rules from specific YARA designed to detect families or packers to broader rules designed to detect suspicious indicators.

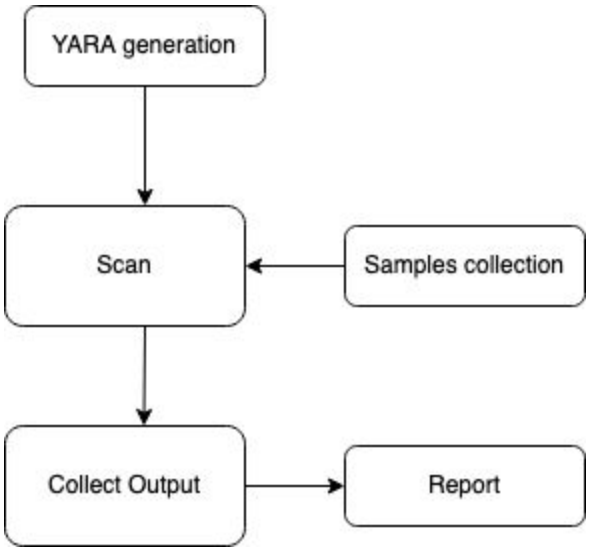

Figure 6 Framework flowchart

Using our scanning engine with a simple print out format:

```
5b1beb5ac97d1ce84ae215dd1ec6b8d7e4522bbe47dbb79e4128abf26d5228891~Rip.exe:
        apis:
                XOR_GetProcAddress 10
                XOR_VirtualAlloc 10
                XOR_LoaderLibraryA 10
                XOR_GetModuleHandleA 10

06apr_doc1~Rip.exe:
        apis:
                XOR_GetProcAddress 10
                XOR_LoaderLibraryA 10
                XOR_GetModuleHandleA 10

06apr_doc.doc:
        apis:
                XOR_GetProcAddress 10
                XOR_LoaderLibraryA 10
                XOR_GetModuleHandleA 10

5b1beb5ac97d1ce84ae215dd1ec6b8d7e4522bbe47dbb79e4128abf26d522889:
        apis:
                XOR_GetProcAddress 10
                XOR_VirtualAlloc 10
                XOR_LoaderLibraryA 10
                XOR_GetModuleHandleA 10

ab2cf2d4-2611-48b5-9799-8d6198e9c936.pcap:
        apis:
                XOR_GetProcAddress 10
                XOR_LoaderLibraryA 10
                XOR_GetModuleHandleA 10
```

Figure 7 Demonstration run

Conclusion and Future Work

In this paper I detailed a proof of concept system for creating a heuristic scanning engine utilizing YARA tags, there are a number of places within the realm of cybersecurity where this concept could be utilized including scanning inbound and outbound objects detecting on a network perimeter. Future work should consist of utilizing this concept for heuristically scanning

inbound objects for example within emails and such a system could built on top of this to consume the scanning reports and then create automated detections and quarantining entire spam campaigns to an enterprise environment based on a heuristic threshold. Scanning outbound objects would also be possible if an enterprise environment were to detect and carve them for temporary scanning you could perform retroactive detection looking for outbound sensitive documents such as intellectual property or other important corporate data being exfiltrated.

References

1: https://virustotal.github.io/yara/
2: https://yara.readthedocs.io/en/v3.4.0/writingrules.html#rule-tags
3: https://yara.readthedocs.io/en/v3.4.0/writingrules.html#metadata
4: Willems, C.: CWSandbox: Automatic Behaviour Analysis of Malware (2006), http://www.cwsandbox.org/
5: https://github.com/kevoreilly/CAPEv2
6: https://github.com/SparkITSolutions/phoenix
7: https://github.com/volatilityfoundation/volatility