Malware Analysis (CS6038)

Week 08.1 Static Analyzers

Scott Nusbaum
nusbausa@ucmail.uc.edu

March 5, 2019

University of CINCINNATI

Overview

- Homework
- Static Analyzers
  - Overview
  - Pros / Cons
  - Free Tools (vscan, ClamAV, Yara)
  - Current Trends and Alternatives

Homework

- Homework 3:
  - Due Feb 28, 2019
  - Graded and uploaded to Blackboard
- Homework 4:
  - Assigned: Feb 28, 2019
  - Due March 14, 2019
  - Covers RE and Network Analysis

Static Analyzers

Signature-based Anti-Virus Systems

- By far, the most popular weapon against cyber attacks is signature-based antivirus software.
- When you are relying upon Symantec's Norton Antivirus, Intel/McAfee's VirusScan, and similar products, you are using a signature-based antivirus solution.
- This may not be the exclusive function of the tools, but it is a significant component.

Pros and Cons to Signatures

- Pros
  - Signatures can be easily written.
    - Based on hashes
    - Strings or ”Unique” byte arrays
  - Open source repo’s exists to cultivate and maintain signatures.

- Cons
  - Easily avoided or bypassed
  - Proprietary databases
  - Not all vendors detect the same way
  - Vendors label or categorize differently the same malware sample
  - Scans can be time intensive
  - Always need to be updated to catch the most resent version of malware

University of CINCINNATI

Free Static Analyzers

- vscan
  - Build around Google's re2 library
- ClamAV
  - Uses binary pattern language for efficiency vs versatility balance
- Yara
  - Contains a more versatile pattern matching engine

* https://zeltser.com/custom-signatures-for-malware-scan/

University of CINCINNATI

vscan

- The tool vscan is a malware scanner that's predominantly focused on searching for the presence of text-based evidence of malware, given a set of files.
- The list of information to search for is managed in a data set named sensors in the config.lua file. The narrow focus of the tool and relatively straightforward configuration make it a useful educational example.
- An example of its input data set is here: https://github.com/mstone/vscan/blob/master/config.lua

- In order to compile I had to install the following:
  - Install libre2-dev
  - Install lua5.1-0-dev
  - Install libarchive-dev
  - Install pandoc
  - Install lynx

ClamAV

- The ClamAV engine has been under active development and production deployment, for arguably the longest duration of these three systems.
- Historically speaking, ClamAV has been prevalent in Linux/UNIX-based mail server deployments for attachment scanning, as well as malware scanning plugin modules for webservers, protecting against malicious uploads.
- ClamAV has its own pattern definition language, and a large library of malware identifiers is maintained by an active support community.
- Using it is simple (below example using evil.pdf generated with Metasploit):

```
bash$ clamscan evil.pdf
evil.pdf: Win.Trojan.MSShellcode-7 FOUND

----------- SCAN SUMMARY -----------
Known viruses: 5725070
Engine version: 0.99.2
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 0.07 MB
Data read: 0.04 MB (ratio 1.64:1)
Time: 7.087 sec (0 m 7 s)
```

ClamAV Signatures

- Allows you to build signature databases out of the hash digest values we discussed earlier.
  - For example: sigtool -md5 evil2.pdf
  - Generate a hash database
    - sigtool –md5 evil2.pdf >> evil_database.hdb
  - Search Hash DB
    - clamscan –d evil_database.hdb evil2.pdf
- More powerfully, however, ClamAV supports a pattern language.
  - This language is also paired with an optional file-structure parsing mechanism that can be used to create powerful, yet efficient, signatures.
  - Full documentation :
- Looking at our evil2.pdf we have the following content:
  - echo –n “please tick the \"Do not” | sigtool –hex-dump
    - 706c656173652074696362074686520222446f206e6f74
  - To match this content in future PDFs, we could generate the following ClamAV signature: PDFmal.metasploit.cs7038;Target:10;0;706c65617365207469636b2074
    - clamscan –d evil_db.ldb evil2.pdf

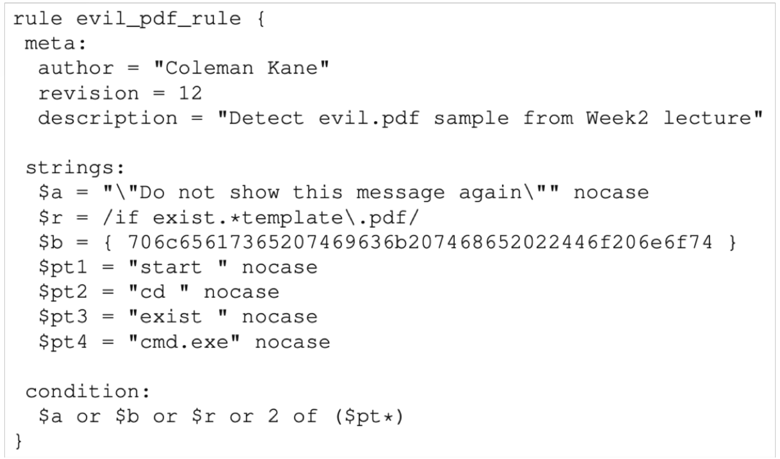
Yara

- The Yara project is a newer system, introduced around 2009-2010, and has a large following of community support.
  - It is similar to ClamAV in this regard.
- However, the community in Yara is focused considerably more on using it for the purposes of malware research and analysis, and less focused on malware detection and alerting.
- However, it works well for both!

University of CINCINNATI

Yara Signatures

- Like ClamAV, Yara also supports a custom signature-building language.
- The syntax provided by Yara is considerably more readable, and arguably more flexible.
  - The side-effect of this functionality is significantly more per rule overhead - leading many users to focus on its use in malware analysis.
- Documentation



```
rule evil_pdf_rule {
 meta:
  author = "Coleman Kane"
  revision = 12
  description = "Detect evil.pdf sample from Week2 lecture"

 strings:
  $a = "\"Do not show this message again\"" nocase
  $r = /if exist.*template\.pdf/
  $b = { 706c6561736520746963 6b20746865202244 6f206e6f74 }
  $pt1 = "start " nocase
  $pt2 = "cd " nocase
  $pt3 = "exist " nocase
  $pt4 = "cmd.exe" nocase

 condition:
  $a or $b or $r or 2 of ($pt*)
}
```

LibYara C API

- In addition to its functionality as a command-line scanning tool like vscan and clamscan, the real power inherent in Yara is that it is a C library at its core, and it can be incorporated to extend other projects.
- Documentation for this C API is here: Link
- At a minimum, initialization consists of the following calls:
  - yr_initialize() - Initialize library
  - yr_compiler_create(YR_COMPILER**) - Create new compiler
  - yr_compiler_add_file(YR_COMPILER*, FILE*, NULL, char*) - read a file and compile the rules it contains (multiple calls possible)
  - yr_compiler_get_rules(YR_COMPILER*, YR_RULES**) - Load a pointer to now-compiled rules into local program scope

University of CINCINNATI

LibYara C API (scanning)

- Scanning involves creating your own custom callback function and then executing one or more of the yr_rules_scan_ * functions, iterating if needed.
- The scanner will run until it has exhaustively searched the buffer or file you provided, and will call your callback function one or more times during the run, providing an opportunity for your custom code to react to the scanner findings.
- When a rule hit occurs, you will receive a pointer to the following data structure in the void *message_data parameter:

```
struct {
  const char *identifier; /* "no_rule" / "yes_rule"  for us */
  const char *tags; /* Rule-defined tags */
  YR_META *metas;   /* Rule-defined metadata key=value pairs */
  YR_STRING *strings; /* strings from rule */
};
```

LibYara Python API

- Additionally, another helpful feature of yara is that there's a really useful Python module for it. In most cases, you can use pip or pip3 to install it:
  - pip install yara-python
- Unlike the C API, the Python context can be instantiated rather quickly:

```
rules = yara.compile(filepath="yara_rules.yar")
```

- And then, scanning is implemented in a manner very similar to the popular regular expressions (re) (https://docs.python.org/3/library/re.html) module:

```
matches = rules.match(data=input_data)
```

University of CINCINNATI

Alternatives

- Cloud based
- Heuristic,
- Behavior Blockers
- HIPS,
- Firewall
- File Reputation,
- Fuzzy hash

University of CINCINNATI