Blue Team Level 1 Certification (Standard)

Introduction to BTL1
- ~~Welcome to Blue Team Level 1!~~ 4 Topics
- ~~Lab and Forum Access~~

SECURITY FUNDAMENTALS DOMAIN
- ~~Introduction to Security Fundamentals~~ 1 Topic
- ~~Soft Skills~~ 7 Topics
- ~~Security Controls~~ 5 Topics | 1 Quiz
- ~~Networking 101~~ 6 Topics | 1 Quiz
- ~~Management Principles~~ 4 Topics | 1 Quiz

PHISHING ANALYSIS DOMAIN
- ~~PA1) Introduction to Emails and Phishing~~ 7 Topics | 1 Quiz
- ~~PA2) Types of Phishing Emails~~ 10 Topics | 2 Quizzes
- ~~PA3) Tactics and Techniques Used~~ 12 Topics | 2 Quizzes
- ~~PA4) Investigating a Phishing Email~~ 8 Topics | 2 Quizzes
- ~~PA5) Analysing URLs, Attachments, and Artifacts~~ 8 Topics | 1 Quiz
- PA6) Taking Defensive Actions 12 Topics | 1 Quiz
- PA7) Report Writing 7 Topics | 1 Quiz
- PA8) Phishing Response Challenge 3 Topics | 1 Quiz

THREAT INTELLIGENCE DOMAIN
- TI1) Introduction to Threat Intelligence 7 Topics
- TI2) Threat Actors & APTs 6 Topics | 2 Quizzes
- TI3) Operational Threat Intelligence 7 Topics | 1 Quiz
- TI4) Tactical Threat Intelligence 7 Topics | 1 Quiz
- TI5) Strategic Threat Intelligence 5 Topics | 1 Quiz
- TI6) Malware and Global Campaigns 6 Topics | 1 Quiz

DIGITAL FORENSICS DOMAIN
- DF1) Introduction to Digital Forensics 5 Topics
- DF2) Forensics Fundamentals 10 Topics | 5 Quizzes

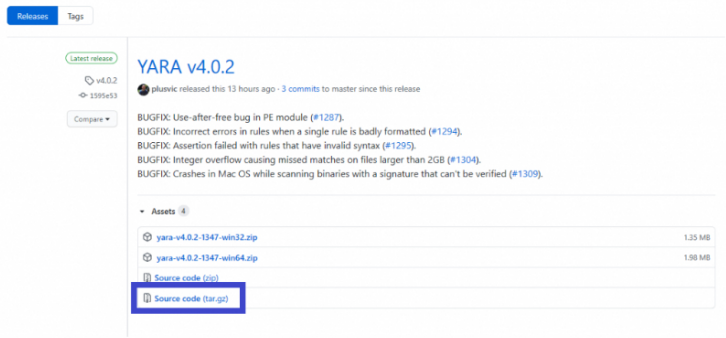
YARA Rules For Detection

YARA rules are a way of identifying specific files by looking at characteristics of a large number of files to see if any of them match the profile, somewhat similar to how antivirus solutions work with virus signatures. YARA was developed by Victor Alvarez who works at VirusTotal and is widely used for security monitoring and detection.

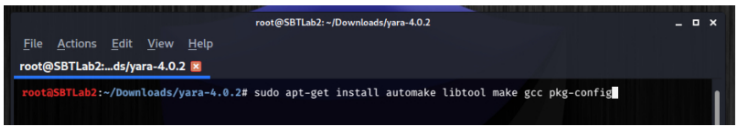
In this lesson we're going to cover how to install YARA on Linux systems (your Kali VM!), how to write rules, how to automatically generate rules from files such as malware, and finally end with a threat-hunting style activity!

INSTALLING YARA ON LINUX

To install YARA on Linux we're going to download the .tar.gz file from the YARA github at the following link: https://github.com/virustotal/yara/releases/tag/v4.0.2. **Make sure you download this file while on your Kali Linux virtual machine.**



Next, there are some dependencies we need to install, other tools and functions that allow YARA to run properly. We can use the command: `sudo apt-get install automake libtool make gcc pkg-config`.



Then we want to open a terminal in the same location as the YARA .tar.gz file, and run the following commands to extract and install yara to our Linux system.

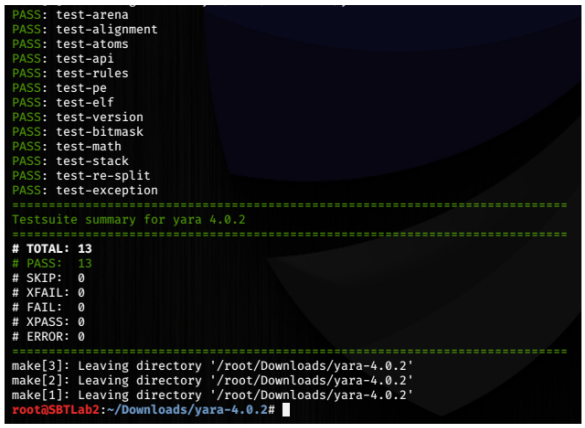
- `tar -zxf yara-4.0.2.tar.gz`
- `cd yara-4.0.2/`
- `./bootstrap.sh`

DF3) Digital Evidence Collection
8 Topics | 1 Quiz
DF4) Windows Investigations
3 Topics | 3 Quizzes
DF5) Linux Investigations
4 Topics | 2 Quizzes
DF6) Volatility
3 Topics | 1 Quiz
DF7) Autopsy
4 Topics | 1 Quiz
SECURITY INFORMATION AND EVENT MANAGEMENT DOMAIN
SI1) Introduction to SIEM
7 Topics | 1 Quiz
SI2) Logging
6 Topics | 2 Quizzes
SI3) Aggregation
2 Topics | 1 Quiz
SI4) Correlation
6 Topics | 1 Quiz
SI5) Using Splunk
5 Topics | 2 Quizzes
INCIDENT RESPONSE DOMAIN
IR1) Introduction to Incident Response
8 Topics | 1 Quiz
IR2) Preparation Phase
10 Topics | 2 Quizzes
IR3) Detection and Analysis Phase
7 Topics | 4 Quizzes
Section Introduction, Detection & Analysis
Common Events & Incidents
Using Baselines & Behavior Profiles
Introduction to Wireshark (GUI)
Introduction to Wireshark (Analysis)
Lab) Network Traffic Analysis
YARA Rules For Detection
Legacy Activity) Threat Hunting With YARA
CMD and PowerShell For Incident Response
Lab) CMD and PowerShell
Activity) End of Section Review, Detection & Analysis
IR4) Containment, Eradication, and Recovery Phase
5 Topics | 1 Quiz
IR5) Lessons Learned and Reporting
7 Topics
IR6) MITRE ATT&CK
13 Topics | 2 Quizzes
BTL1 EXAM
Exam Preparation
Using RDP and SSH
How to Start Your Exam

Then we need to compile and install YARA with the following three commands:

- `./configure`
- `make`
- `sudo make install`

Once these have completed, we need to check if everything has installed correctly using the command: `make check`. Everything should pass, as shown in the screenshot below.



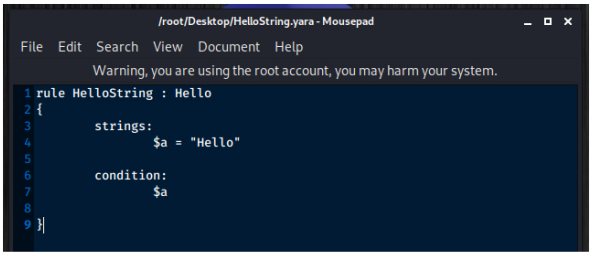
Great, we have YARA installed on our Kali virtual machine! If you run into any issues, read the following resources:

- https://askubuntu.com/questions/265471/autoreconf-not-found-error-during-making-qemu-1-4-0
- https://stackoverflow.com/questions/18978252/error-libtool-library-used-but-libtool-is-undefined
- https://yara.readthedocs.io/en/stable/gettingstarted.html

WRITING YARA RULES

When writing rules, there are three core components that we need to include. The rule name, identification values, and conditions. Below is a screenshot of an extremely simple Yara rule we have made, and we will talk through the components.

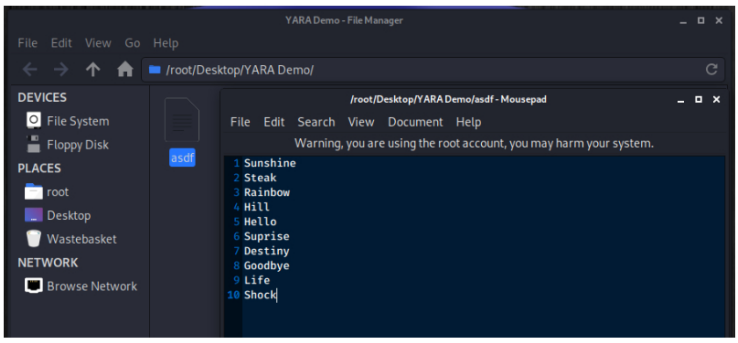
1. **We need to define the name of the rule.**
2. **We need to define the characteristics the rule is looking for in files.**
3. **We need to define the conditions under which a file will be flagged.**



Let's explain what this rule is trying to achieve.

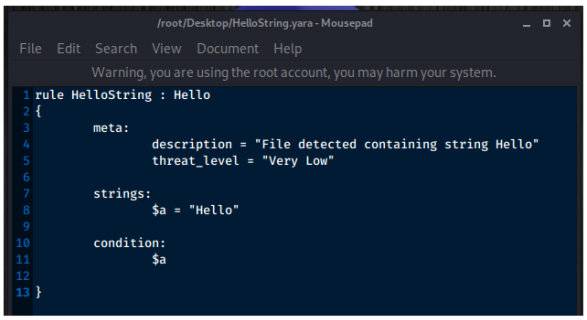
- On the first line we have defined the rule name as "HelloString" and a short-hand name of "Hello".
- On Line 3 we declare a file property, in this case, a text string that should be present inside the file. We use the variable $a to hold the value of "Hello".
- On Line 6 we declare the condition that needs to be met for a file to be flagged. Because we've used $a, this means if any scanned files contain the string "Hello" they will be flagged to us.

Let's show you what we mean with an example. We have a directory (YARA Demo) on our desktop that includes a text file with the word Hello in it (Line 5). We can use this simple YARA rule to look for any files that contain the word Hello.

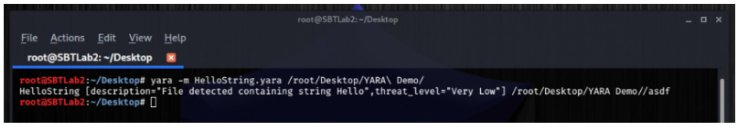


So let's run our YARA rule against the YARA Demo directory and see if we can detect the fact that this file contains the string Hello!

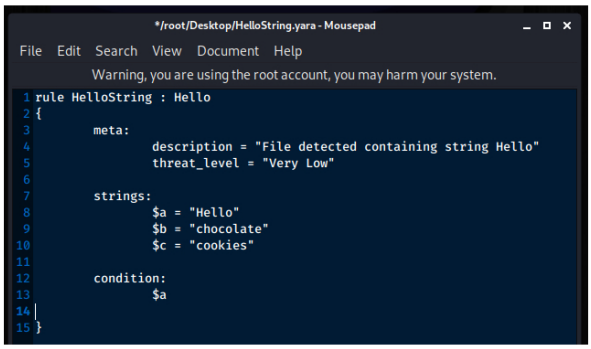
It worked! To develop your understanding of YARA rules, let's continue to build on this one. Next, let's add a human-readable description and our own "threat level" value to the output of our scan results. To do this we can edit the rule file, and add these values under a new heading named "meta". Here's our updated rule:



Now let's run the command again and see what the output looks like now. We need to make one small change, which is including the "-m" flag, telling YARA to print meta information from the alerting rule.



This can be a good place to store basic information, such as who wrote the YARA rule, what version it is, and a description of what it's doing. None of this information is considered during file analysis and is purely for human consumption. Let's build on our rule again by adding in some additional properties and conditions. Instead of just looking for the string, let's also look for the strings "chocolate" and "cookies". We can add these as new variables to our rule.



Because we haven't changed the condition, this rule will continue to only alert on $a, which is "Hello". We have two options here:

- Use OR statements to allow the rule to flag any files that contain any of the strings.
- Use AND statements to allow the rule to flag any files that contain at least two of the strings.

For example, if we set the condition to $a or $b or $c, YARA would alert us to any files that contained one or more of the text strings. If we set the condition to $a and $b and $c, YARA would alert on files that contain all three of these strings. Let's set the condition to use the OR operator.

Let's also change the meta description value to "File detected containing a defined string". If we run this rule without any additional flags, it will simply highlight files to us that match the condition, but this doesn't tell us which of the strings has been detected. To show this, we can use the flag **-s** to print matched strings. Let's try it out!

From this output we can see the alerting rule, the meta information, the file paths and file names, and the strings that have been matched from our YARA rule. Before we finish with this section, let's quickly cover some important YARA flags:

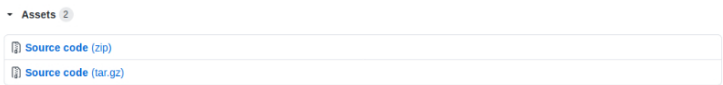
- **-m** // Prints the associated meta information to the terminal after a YARA scan.
- **-s** // Prints the matching strings to the terminal after a YARA scan.
- **-r** // Recursively scan all subfolders within the target location to ensure everything is scanned.

CREATING RULES WITH YARGEN

YarGen is a python tool that allows us to automatically generate rules for malicious files, and can save us a lot of time, and prevent human errors when writing rules. We're going to walk you through installing YarGen, and how to use it. You'll be doing this in the next threat hunting activity, so make sure you have it installed on your Kali Linux virtual machine!

If you run into any issues with yargen or its dependencies, try using the python3 and pip3 commands instead of python and pip.

First, we need to open the following webpage in our Kali machine, go to the releases page, and scroll to the bottom so we can download the .tar.gz file – https://github.com/Neo23x0/yarGen/releases



Then we need to open a terminal in the location of the downloaded .tar.gz file, and the run the following command

to extract `YarGen: tar -zxf yarGen-0.18.0.tar.gz`. Next we need to use pip to download and install some additional dependencies. Try running the command `pip` in a terminal. If it states that the command is not found then you need to install python-pip using the command `sudo apt-get install python-pip`. Now execute the following two commands:

- `sudo pip install pefile cd`
- `sudo pip install scandir lxml naiveBayesClassifier`

Now use cd to move into the extracted YarGen directory, and run the following command to ensure everything is updated: `python yarGen.py --update`. Great, everything is ready. We can now type the command `python yarGen.py --help` to see the available options for generating rules with this tool.

```
  -m M                  Path to scan for malware
  -y min-size           Minimum string length to consider (default=8)
  -z min-score          Minimum score to consider (default=5)
  -x high-scoring       Score required to set string as 'highly specific
                        string' (default: 30)
  -s max-size           Maximum length to consider (default=128)
  -rc maxstrings        Maximum number of strings per rule (default=20,
                        intelligent filtering will be applied)
  --excludegood         Force the exclude all goodware strings

Rule Output:
  -o output_rule_file   Output rule file
  -a author             Author Name
  -r ref                Reference
  -l lic                License
  -p prefix             Prefix for the rule description
  --score               Show the string scores as comments in the rules
  --nosimple            Skip simple rule creation for files included in super
                        rules
  --nomagic             Don't include the magic header condition statement
  --nofilesize          Don't include the filesize condition statement
  -fm FM                Multiplier for the maximum 'filesize' condition value
                        (default: 3)
  --globalrule          Create global rules (improved rule set speed)
  --nosuper             Don't try to create super rules that match against
                        various files

Database Operations:
  --update              Update the local strings and opcodes dbs from the
                        online repository
  -g G                  Path to scan for goodware (dont use the database
                        shipped with yaraGen)
  -u                    Update local standard goodware database with a new
                        analysis result (used with -g)
  -c                    Create new local goodware database (use with -g and
                        optionally -i "identifier")
  -i I                  Specify an identifier for the newly created databases
                        (good-strings-identifier.db, good-opcodes-
                        identifier.db)

General Options:
  --nr                  Do not recursively scan directories
  --oe                  Only scan executable extensions EXE, DLL, ASP, JSP,
                        PHP, BIN, INFECTED
  -fs size-in-MB        Max file size in MB to analyze (default=10)
  --noextras            Don't use extras like Imphash or PE header specifics
  --debug               Debug output

Other Features:
  --opcodes             Do use the OpCode feature (use this if not enough high
                        scoring strings can be found)
  -n opcode-num         Number of opcodes to add if not enough high scoring
                        string could be found (default=3)
```

The flags we're interested in are -m which allows us to define the path to the file or files we want to generate rules from, and -o which allows us to define the name and path of the rules generated by YarGen. Let's test it out on a malicious .exe file. The command we want to execute is: `python yarGen.py -m /root/Desktop/Malware -o ./TestRule.yara`

- **python yarGen.py** – Runs the yarGen python script (we need to be in the same directory as this file, or provide the file path to yarGen.py)
- **-m /root/Desktop/Malware** – Create rules for files inside the Malware folder on the Desktop of our current user, root
- **-o ./TestRule.yara** – Output the generated rule to the current folder of the terminal (./) in our case, the yarGen folder, and name the file TestRule.yara

```
[+] Total: 19764 / Added 183 entries
[+] Loading ./dbs/good-strings-part1.db ...
[+] Total: 10167454 / Added 1046235 entries
[+] Loading ./dbs/good-strings-part3.db ...
[+] Total: 12284943 / Added 2117489 entries
[+] Loading ./dbs/good-exports-part8.db ...
[+] Total: 404321 / Added 3834 entries
[+] Initializing Bayes Filter ...
[-] Training filter with good strings from ./lib/good.txt
[+] Processing malware files ...
[+] Processing /root/Desktop/Malware/wallpaperHD.exe ...
[+] Generating statistical data ...
[+] Generating Super Rules ... (a lot of foo magic)
[+] Generating Simple Rules ...
[-] Applying intelligent filters to string findings ...
[-] Filtering string set for /root/Desktop/Malware/wallpaperHD.exe ...
[+] Generating Super Rules ...
[=] Generated 1 SIMPLE rules.
[=] Generated 0 SUPER rules.
[=] All rules written to ./TestRule.yara
```

It took about 3 minutes for the rule to be generated in our lab. Once we saw the "All rules written" message we used CTRL + C to exit, then used the following command to read the rule from the terminal: `cat TestRule.yara`. Let's take a look at what was generated!

```
import "pe"

rule wallpaperHD {
   meta:
      description = "Auto-generated rule - file wallpaperHD.exe"
      author = "YarGen Rule Generator"
      reference = "not set"
      date = "2020-06-27"
      hash1 = "240387329dee4f03f98a89a2feff9bf30dcba61fcf614cdac24129da54442762"
   strings:
      $x1 = "C:\\Users\\x\\Desktop\\Home\\Code\\Mfix\\Release\\Mfix.pdb" fullword ascii
      $x2 = "svchostx64.exe" fullword ascii
      $s3 = "\\svchostx64.exe" fullword wide
      $s4 = "winupd32.exe" fullword ascii
      $s5 = "wincfg32.exe" fullword ascii
      $s6 = "winupdsrvc.exe" fullword ascii
      $s7 = "wincfg.exe" fullword ascii
      $s8 = "winupd.exe" fullword ascii
      $s9 = "\"/108/99/111/113/29/41/56/31/39/55/18/16/10/54/58/44/47/34/35/63/127/14/65/109/103/\" ; " fullword ascii
      $s10 = "\"/108/99/111/113/29/41/56/31/39/55/18/16/10/54/58/44/47/34/35/63/102/14/65/109/103/\" ;" fullword ascii
      $s11 = "3)3.383E3W3\\3a3n3" fullword ascii
   condition:
      ( uint16(0) == 0x5a4d and
        filesize < 1000KB and
        pe.imphash() == "654ce24415bc7fe02dd3f21cfc4ec6c6" and
        ( 1 of ($x*) or all of ($s*) )
      ) or ( all of them )
}
```

The above rule has selected a number of strings from the binary (wallpaperHD.exe) and the condition also includes the file size and the file hash. Due to how the condition is written, the rule is able to look for a combination of all the indicators of compromise (IOCs) we have pulled from the file, and we could now import this YARA rule into our endpoint detection and response platform (EDR) to use this for detection on a large scale.

Now that you understand how YARA works, and how to use YarGen, it's time for a threat hunting activity!

Quizzes

Legacy Activity) Threat Hunting With YARA

Previous Topic

Back to Lesson

Next Topic

Privacy & Cookies Policy

Privacy - Terms