


YARA 101

 October 11, 2013 By [Ernesto Corral](#)

What is YARA ?

When speaking about malware detection, there are mainly three ways of determining if a file is malicious: signatures, heuristics and string signatures.

The most widespread in the antivirus detection systems is the **signature based detection**, ie based on the HASH of a file, check it against a signature database and see if this file has previously been detected as malware. This kind of signature is useless for the detection of unknown malware, and to evade this system you just need to recompile the code in a different system or change a single bit.

In order to try to stop these evasion methods, the **heuristic method** is usually the chosen one. This method relies on the behavior of the executable file and, according to the actions that it performs inside the system, it decides if it's dealing with a malicious file. The main issue of this method is that, as many legitimate programs perform suspicious actions, it can generate a large amount of false positives.

Last but not least, there is the method which this article refers to: **string signatures**. This method is based on another kind of signatures, different from the aforementioned kind. Instead of using HASH signatures, it uses text or binary strings that uniquely identify a malware sample. That way, even if the file has been tampered with, if it still contains those string signatures, the analysts will be able to detect and classify the malware sample.

YARA is a tool that was designed by Víctor Manuel Álvarez mainly for string signature based detection and classification of malware. And I say "mainly" because it can be used in different ways – man shall not live by malware alone.

Using relatively simple rules, YARA goes over the files that it receives and looks for the strings defined in the rule and, if they match certain conditions, it tells you what rules match each file. This can also be applied to running processes.

It is very easy to use and, in order to install it, you just need to download the packet for your system from the [project's website](#) and follow [the install instructions](#) if you are Linux (the link is for Ubuntu, but I'm sure you can adapt it to your favorite distro); and for Windows it's even easier: you just have to unzip the YARA executable and, for the YARA library: double click, next, next, finish.

One of the things I like the most about YARA: it has a Python library that allows you to integrate it very easily with your projects!

YARA rules (Yeah! It rulez!)

To learn about YARA rules, let's start with a simple example:

```
rule HelloWorld
{
    strings:
        $a = "Helloworld"

    condition:
```

 [Spanish](#)

PAGES

[About](#)
[authors](#)
[Cookie Policy](#)
[Guest posts](#)
[Legal Notice](#)
[Privacy Policy](#)

SEARCH

AUTHORS

[Manuel Benet](#) (224)
[Antonio Villalon](#) (209)
[Antonio Sanz](#) (102)
[Collaborators](#) (84)
[Maite Moreno](#) (72)
[Robert Loved](#) (60)
[Joseph Rosell](#) (59)
[Josemi Holguin](#) (54)
[Joaquin Moreno](#) (54)
[Jose L. Villalon](#) (50)
[Joan Soriano](#) (43)
[Nelo Belda](#) (43)
[Jose Vila](#) (43)
[Antonio Huerta](#) (38)
[Fernando Dry](#) (38)

[All authors](#)

ARCHIVES

GOAL

[log in](#)
[Entries feed](#)
[Comments feed](#)
[Cookies policy](#)

```
    $a
}
```

Dissecting the rule:

```
rule HelloWorld           // rule -> Keyword all rules start with
                        // HelloWorld -> Rule name
{
    strings : // Strings section -> In this section the strings
              //that will be used afterwards to check the
              //the conditions are defined.
    $a = "Hello world" //variable $a, it contains "Hello world"

    condition : // Condition section -> Conditions are defined.
               $a      //If it matches $a, true
}
```

We create three text files to test the rule:

```
test1.txt:
SecurityArtWork

test2.txt:
hello world

test3.txt:
hello world
```

Analyzing YARA's output. The file rules.yar contains the rule and the option `-s` will show the string that matches the rule:

```
# yara -s rules.yar test1.txt
# yara -s rules.yar test2.txt
HelloWorld test2.txt
0x0:$a: Hello world
# yara -s rules.yar test3.txt
```

We can see a match in the file test2.txt:

```
HelloWorld test2.txt -> Matched rule
0x0:$a: Hello world -> Offset:variable: String matched
```

Even if the documentation is very good, there are some undocumented things, like the ones presented by Julia Wolf ([@foxgrrl](#)) in the [BerlinSides](#) talk entitled *Classifying malware with YARA* . Unfortunately, I didn't manage to find it published, so I guess you will only be able to find those problems while fighting against the (YARA) rules.

YARA with binaries

In order to test YARA with binary files, I will set up a scenario. Let's say we work for an evil corp called The Corp. Inside The Corp, people handle very sensitive patents and in the past it has been the target of some... [APTs](#) ! We, as part of the The Corporation's CSIRT, have to be ready for defending our beloved evil corp from possible attacks. The previous attacks analyzes reveal that the entry point of all the attacks has been email, and in all of them we have found administrative accounts somewhere in the middle of the code.

As any "good" organization, in The Corporation, a strict naming scheme is followed, and all administrative accounts follow one of the following schemes: either admin.<user> or corp.<user>. Furthermore, in the area we want to defend, we only have Linux systems.

In order to defend The Corporation, we set up a sniffer that captures all the email traffic, extracts the attachments and sends them to YARA.

We prepare the following rule to defend The Corporation:

```
rule APT_adm_corp : apt //apt is just a tag, it doesn't
                        //affect the rule.
{
    meta: //Metadata, they don't affect the rule
        author = "xgusix"

    strings:
        $adm = "adm."
        $corp = "corp."
        $elf = { 7f 45 4c 46 } //ELF file's magic numbers

    condition:
        $elf in (0..4) and ($adm or $corp)
        //If $elf in the first 4 bytes and it matches $adm or $corp
}
```

Some fields that did not appear before are explained in the comments. In order to test the rule, we launch it against a malware simulation, which is no less than the following code after being compiled:

```
//testbin.c
#include

intmain()
{
    char *user = "adm.user";
    printf("%s\n",user);
    return 0;
}
```

We run it against our rule with the options: `-s` to watch the strings; `-g` to watch the tags; and `-m` to watch the metadata:

```
# yara -s -m -g rules.yar testbin
APT_adm_corp [apt] [author="xgusix"] testbin
0x0:$elf: 7F 45 4C 46
0x4c0:$adm: adm.
```

The rule has been executed successfully and it has caught our "malware".

There are many ways you can use YARA and what I have just described is simply an example of what can be achieved with this tool. I encourage you all to give it a try and find out what you can do with YARA.



Comments



Executive Coaching says
April 5, 2015 at 1:13 pm

Have you ever thought about publishing an ebook or guest authoring on other sites? I have a blog based uponn on the same information you discuss and would really like to have you share some stories/information. I know my readers would value your work. If you're even remotely interested, feel free to send me an e-mail.

