Centro de
Investigación en
Computación

YARA

Herramienta de búsqueda de patrones

Teoría y
Ejemplos

Course

Análisis y Detección de Malware

Instructor

Acosta Bermejo Raúl

Lecture notes

2024-B
12 de novoembre del 2024

Instituto
Politécnico
Nacional

Table of contents (outline)

Tabla de contenido

Introducción

Definición

Teoría

Clasificación

Referencias

Introduction

Definición

Ver 4.4.= 19/sept/2023
Ver 4.3.1 21/abr/2023
Mucha documentación de la 2.

- Significados

  Yet Another Recursive Acronym o *Yet Another Ridiculous Acronym*.
- Es una herramienta usada principalmente en la detección e investigación de malware.
- Proporciona un método basado en reglas para crear descripciones de familias de malware basadas en patrones binarios y de texto.
- Una descripción es esencialmente un nombre de regla, la cual define un conjunto de cadenas y expresiones booleanas.
- El lenguaje usado tiene características de las expresiones regulares de Perl.
- Por default YARA viene con módulos para procesos de análisis de archivos PE, ELF.
- También ofrece soporte para la plataformas abiertas como el sandbox Cuckoo.

http://virustotal.github.io/yara/, **https**://yararules.com/project/

Introduction

Definición

yara

The pattern matching swiss knife for malware researchers (and everyone else)

{ } YARA in a nutshell

YARA is a tool aimed at (but not limited to) helping malware researchers to identify and classify malware samples. With YARA you can create descriptions of malware families (or whatever you want to describe) based on textual or binary patterns. Each description, a.k.a rule, consists of a set of strings and a boolean expression which determine its logic. Let's see an example:

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        thread_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
```

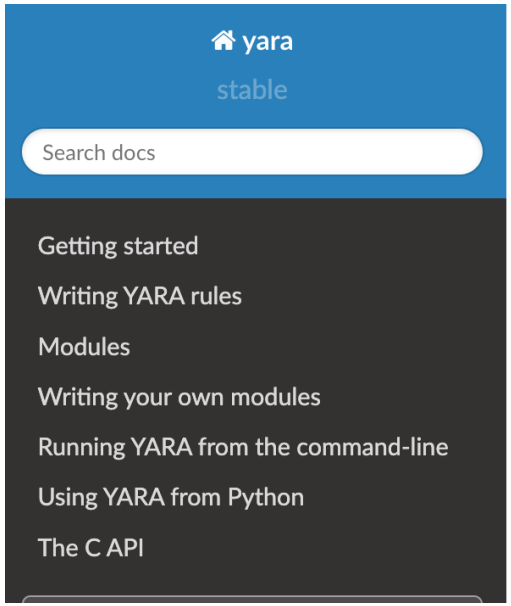
View project in GitHub

Download Latest release

Read the Documentation

Ask for help at YARA's group

Send Bug Report



Welcome to YARA's documentation!

YARA is a tool aimed at (but not limited to) helping malware researchers to identify and classify malware samples. With YARA you can create descriptions of malware families (or whatever you want to describe) based on textual or binary patterns. Each description, a.k.a. rule, consists of a set of strings and a boolean expression which determine its logic. Let's see an example:

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
```

Introduction

Referencias

URL

- Documentación
  - https://virustotal.github.io/yara/
  - https://yara.readthedocs.io/en/stable/
- Código
  - https://github.com/VirusTotal/yara
  - https://github.com/VirusTotal/yara/releases

Pero hay otras fuentes complementarias

- Busquen ustedes las demás ...

Categorias de las reglas

Típicas

Lista

1. Malicious documents
   Example: PDF with JS or Office documents with macros.
2. CVE rules
3. Crypto
4. Exploit Kits
5. Packer
6. Email
7. Anti-debug/Anti-virtualization.
   Techniques used by malware to evade automated analysis.

Introduction

Referencias

Evaluación de las reglas

- Teoría de Clasificadores
  - Falsos positivos
  - Verdaderos positivos

Reglas

Gramática y ejecución

Teoría

YARA reglas

Sintaxis

Sintaxis simplificada

```
import name

rule <rule_name> : [tag1 tag2]
{
    meta:
          autor = “name”
    strings:
        $str1 = “”
   condition:
       $str1 or $str1 and $str1
}
```

```
/*
This is a place for comments 1
*/

// Comentario 2
```

YARA reglas

Sintaxis

Ejecución

$ **yara** [OPTION]... RULES_FILE FILE | DIR | PIDDESCRIPTION

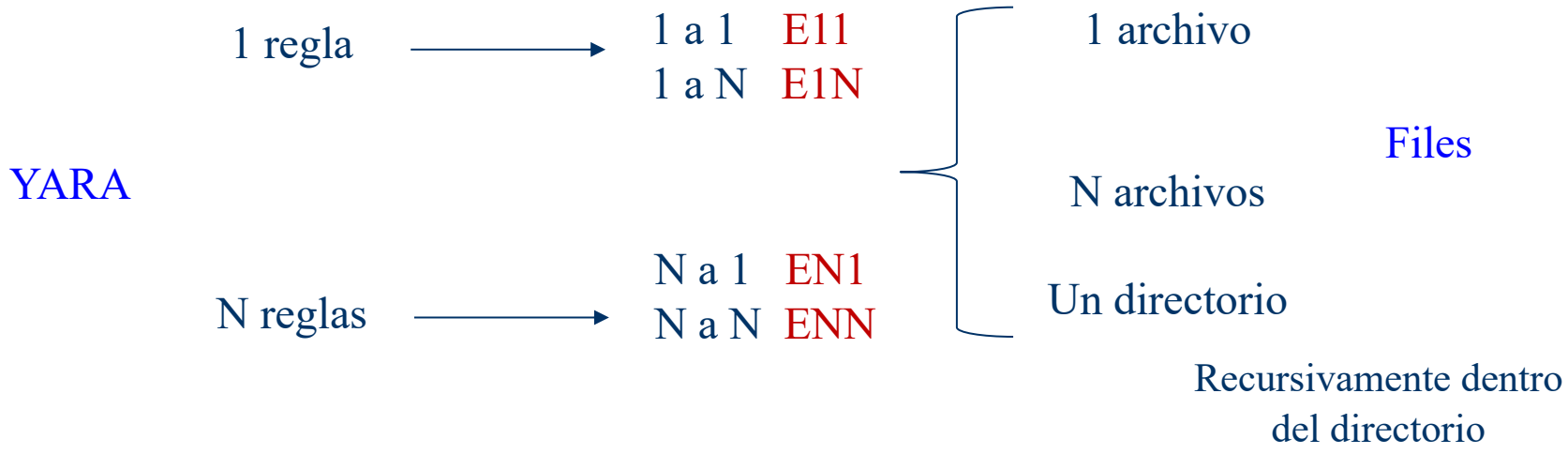
- Scans the given FILE, all files contained in directory DIR, or the process indentified by PID.
- Looking for matches of patterns and rules provided in RULES_FILE.
- The basic command to use is:

$ yara archivo_reglas.yar archivo_a_analizar
$ yara –t etiqueta archivo_reglas.rule archivo_a_analizar
$ yara -C
--compiled-rules

YARA reglas

Sintaxis

Combinaciones de búsquedas



YARA reglas

Sintaxis

Ejemplos

- E11.
  $ yara archivo_reglas.yar archivo

- E1N
  $ yara archivo_reglas.yar *
  $ yara archivo_reglas.yar directorio
  $ yara –r archivo_reglas.yar directorio Busqueda recursiva

- EN1 -w No warnings
  $ find binarypig -name "*.yara" -exec yara -w {} archivo_a_analizar \;

- ENN
  $ yara archivo_reglas.yar archivo_a_analizar

YARA reglas

Sintaxis

Campos de meta

- **Author** – Name, email address, Twitter handle.
- **Date** – Date rule was created.
- **Version** – The version number of the YARA rule for tracking amendments.
- **Reference** – A link to an article or download of the sample, this is used to provide relevant information on the malware sample the rule is designed to detect.
- **Description** – A brief overview of the rule's purpose and malware it aims to detect.
- **Hash** – A list of sample hashes that were used to create the YARA rule.

YARA reglas

Sintaxis

Palabras reservadas

all	and	any	ascii	at	condition	contains
entrypoint	false	filesize	fullword	for	global	in
import	include	int8	int16	int32	int8be	int16be
int32be	matches	meta	nocase	not	or	of
private	rule	strings	them	true	uint8	uint16
uint32	uint8be	uint16be	uint32be	wide		

Cadenas

Sintaxis y ejemplos

Teoría

Cadenas

Definiciones

Caracteres

$str1 = “letras”

$str2 = "foobar" nocase

En hexadecimal

$str1 = { E2 34 ?? A? } // Wild-cards

$str1 = { E2 [4-6] A4 } // subcadena de longitud entre 4 y 6 carcateres

[X-Y] debe cumplir la condición 0 <= X <= Y

FE 39 45 [10-] 89 00

FE 39 45 [-] 89 00

Cadenas sin límite por ejemplo [10-infinito], la segunda [0-infinito].

$str2 = { F4 23 (62 B4 | 56) 45 }

YARA reglas

Sintaxis

Significado de algunas palabras

- wide. Used to search for strings encoded with two bytes per character, something typical in many executable binaries.
- ascii. To search for strings in both ASCII and wide form.
- xor. To search for strings with a single byte XOR applied to them.
- base64. To search for strings that have been base64 encoded. A good explanation of the technique is at:
  - https://www.leeholmes.com/searching-for-content-in-base-64-strings/
- fullword. It guarantees that the string will match only if it appears in the file delimited by non-alphanumeric characters.
  - For example, the string domain, if defined as fullword, doesn't match www.mydomain.com but it matches www.my-domain.com and www.domain.com.

YARA reglas

Sintaxis

The number of strings

- all of them // all strings in the rule
- any of them // any string in the rule
- all of ($a*) // all strings whose identifier starts by $a
- any of ($a,$b,$c) // any of $a, $b or $c
- 1 of ($*) // same that "any of them”
- none of ($b*) // zero of the set of strings that start with "$b”

Applying the same condition to many strings

There is another operator very similar to of but even more powerful, the for..of operator. The syntax is:

for expression of string_set : (boolean_expression)

Cadenas

Definiciones

Subconjunto de de secuencias de caracteres “escapables”.

\"	Double quote
\\	Backslash
\t	Horizontal tab
\n	New line
\xdd	Any byte in hexadecimal notation

Otras palabras especiales para manejar cadenas son:

wide, fullword

Cadenas

Definiciones

Expresiones regulares

Se definen en la misma forma que las cadenas pero estan encerradas con caracteres backslaesh (\) en lugar de dobles comillas.

```
$re1 = /md5: [0-9a-zA-Z]{32}/
$re2 = /state: (on|off)/
```

Algunos de los meta-caracteres que se pueden son los siguientes:

Consultar el manual para la lista completa.

\	Quote the next metacharacter
^	Match the beginning of the file
$	Match the end of the file
|	Alternation
()	Grouping
[]	Bracketed character class
*	Match 0 or more times
+	Match 1 or more times
?	Match 0 or 1 times

Cadenas

Definiciones

Endianness is the order or sequence of bytes of a word of digital data in computer memory. Endianness is primarily expressed as:

- **big-endian** It stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest.
- **little-endian**. A little-endian system, in contrast, stores the least-significant byte at the smallest address.

Ejemplos

Archivo de PE, primeros caracteres

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00   MZ..........ÿÿ..
00000010   B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ¸.......@.......
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000030   00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00   ............€...
00000040   0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ..º..´.Í!¸.LÍ!Th
00000050   69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F   is program canno
00000060   74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20   t be run in DOS
00000070   6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00   mode....$.......
```

$mz = { 4d 5A }
condition:
uint16(0) == 0x5A4D

- The file must be a Windows executable, this is because the hex values 4D 5A (MZ letters) are always located at the start of an executable file header.
- This is reversed in YARA due to endianness.

Condiciones

Sintaxis y ejemplos

Teoría

Condiciones

Definiciones

Son expresiones formadas por

- Expresiones booleanas con los operadores: and, or, not.
- Operadores relacionales: >=, <=, <, >, == y !=.
- Operadores aritméticos: +, -, *, \, %.
- Operadores de bits (bitwise): &, |, <<, >>, ~, ^.
- Expresiones numéricas.

```
rule Example
{
  strings:
      $a = "text1"
      $b = "text2”
      $c = "text3"
  condition: ($a or $b) and $c
}
```

Categorias de las reglas

Típicas

Combinando reglas

```
rule WindowsPE
{
    condition:
        uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550
}

rule Ryuk_SequentialComparisons
{
   strings:
      $op1 = { FF 15 [5-26] 83 ?? 4E 75 [2-26] 83 [1-2] 54 75 [2-26] 83 [1-2] 41 75 }
    condition:
       WindowsPE and all of them
}
```

Módulos

Extender la funcionalidad

Teoría

Módulos

Definiciones

- Modules are the method YARA provides for extending its features.
- They allow you to define data structures and functions which can be used in your rules to express more complex conditions.
- List
  - PE
  - ELF
  - Cuckoo
  - Magic
  - Hash
  - Math
  - Dotnet
  - Time
  - Console

Módulos

Definiciones

Examples

```
import "pe"

rule single_section {
   condition: pe.number_of_sections == 1
}

rule control_panel_applet {
   condition: pe.exports("CPlApplet")
}
rule is_dll {
   condition: pe.characteristics & pe.DLL
}
```


Módulos

Definiciones

Otras extensiones

- Extending YARA is not only limited to writing modules.
- There is also another great feature called “External Variables” and they allow you to define rules which depend on values provided from the outside.

Obtener reglas

Creación manual o automáticamente

Teoría

Crear reglas

Examples

Como generarlas?

- YARA uses signatures similar to antivirus solutions, it would make sense to reuse these signatures as a rule database.
  - With the use of the script **clamav_to_yara.py**, you can convert the ClamAV signature database to your own ruleset.
- Some threat intelligence sharing platforms, such as MISP and ThreatConnect, also support YARA. This allows you to build rules based on your own collected threat information.

Crear reglas

Fuentes

Sitios con reglas

- Gratuitas

  GitHub repository Yara Rules.
  - https://github.com/Yara-Rules/rules, Yara versión 3.
  - https://github.com/reversinglabs/reversinglabs-yara-rules
- Comerciales
  - Valhalla: Rich meta data, API (Python module), CLI.

    12,000 quality tested YARA rules in 8 different categories: APT, Hack Tools, Malware, Web Shells, Exploits, Threat Hunting, Anomalies and Third Party. Database grows by 1500 YARA rules every year. A subscription includes the curation of old rules. We've changed and improved over 300 old rules within the last year.
  - https://www.nextron-systems.com/valhalla/.

Crear reglas

Examples

yarGen

- The main principle is the creation of yara rules from strings found in malware files while removing all strings that also appear in goodware files.
- Therefore yarGen includes a big goodware strings and opcode database as ZIP archives that have to be extracted before the first use.
- Version 0.23.0 yarGen has been ported to Python3.
- Example

  $ python yarGen.py -m /path/malware_directory

Links

- https://github.com/Neo23x0/yarGen

Hay que generara una BD
good-strings-office.db
good-opcodes-office.db

Crear reglas

Examples

Ejemplos de creación

- Links
  - https://www.reversinglabs.com/blog/exposing-ryuk-variants-using-yara
  - https://medium.com/@cloud_tips/yara-rule-examples-17414a0536f7

Temas avanzados

Creando módulos y usando otros lenguajes

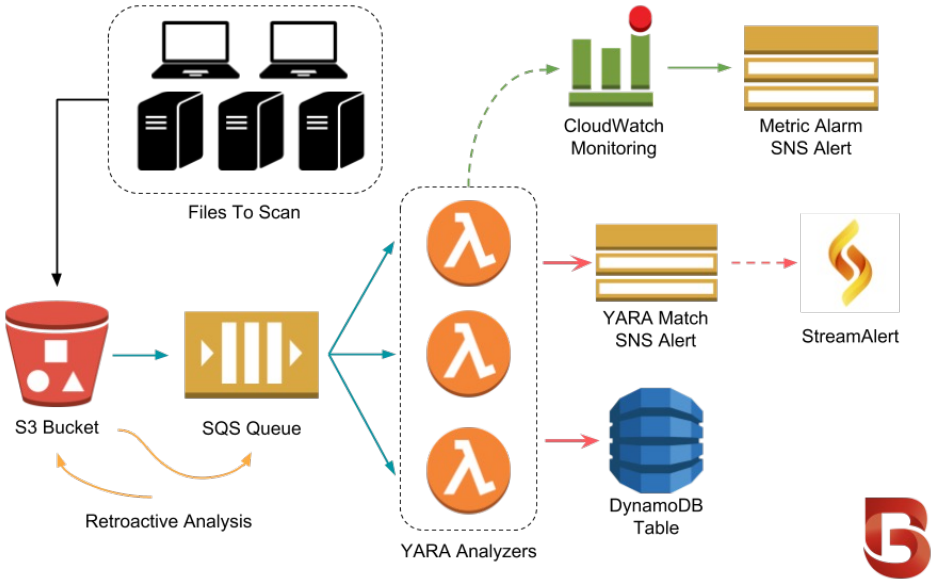
Teoría

Sistemas que usan Yara

Nube

Descripción

- **BinaryAlert** is a serverless, real-time framework for detecting malicious files. BinaryAlert can efficiently analyze millions of files a day with a configurable set of YARA rules and will trigger an alert as soon as anything malicious is discovered.



- Built with Amazon Web Services (AWS).
- The entire infrastructure is described with Terraform configuration files

YARA bindings

Lenguajes

Binding para Lenguaje C

API de YARA: en python

- The first step is importing the YARA library:
  import yara
- Then you will need to compile your YARA rules before applying them to your data, the rules can be compiled from a file path:
  rules = yara.compile(filepath='/foo/bar/myrules')
- Compile returns an instance of the class **yara.Rules** Rules. This class has a save method that can be used to save the compiled rules to a file:
  rules.save('/foo/bar/my_compiled_rules')
- The compiled rules can be loaded later by using the load method:
  rules = yara.load('/foo/bar/my_compiled_rules')
- Instances of Rules also have a match method, which allows you to apply the rules to a file:
  matches = rules.match('/foo/bar/my_file')

Creando módulos

Definiciones

- In YARA 3.0 you can extend its features to express more complex and refined conditions.
- YARA 3.0 does this by employing modules, which you can use to define data structures and functions, which can be later used from within your rules.

```
#include <yara/modules.h>
#define MODULE_NAME demo

begin_declarations;
  declare_string("greeting");
end_declarations;

int module_initialize( YR_MODULE* module) {
  return ERROR_SUCCESS;
}
int module_finalize( YR_MODULE* module) {
  return ERROR_SUCCESS;
}
int module_load( YR_SCAN_CONTEXT* context,
YR_OBJECT* module_object, void* module_data,
size_t module_data_size) {
  set_string("Hello World!", module_object, "greeting");
  return ERROR_SUCCESS;
}
int module_unload( YR_OBJECT* module_object) {
  return ERROR_SUCCESS;
}

#undef MODULE_NAME
```

YARA

Otras herramientas

YARA-CI

- YARA-CI helps you to keep your YARA rules in good shape. It can be integrated into any GitHub repository containing YARA rules, and it will run automated tests every time you make some change. The automated tests include:
  - Rules validation
  - Detection of false positives
  - Detection of false negatives
- **URL**
  - https://yara-ci.cloud.virustotal.com/

- Módulo Radare2 para Yara
  - https://www.securityartwork.es/2017/07/19/modulo-radare2-yara/

Práctica

Tarea de integrar Yara

Descripción

Laboratorio de Malware

Descripción

Objetivo
Detectar características de malware mediante YARA

Que reporte que detecta.
Estatico/dinámico

Agregar la funcionalidad

- Compilar reglas
  - Guardarlas de manera jerárquica usando el nombre del autor.
  - Guardarlas compiladas y su ”código”.
- Crear una BD
  - Con una tabla para guardar la información de las reglas.
  - Con una tabla para guardar información del autor.
  - Con una tabla para guardar la clasificación.
- Crear un programa
  - Que escanea archivos usando las reglas de la BD.
  - Usar python y un ORM, por ejemplo SQLAlchemy.
- Entregar
  - SQL de la BD.
  - El código
  - Las reglas.

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx
http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx
racosta@cic.ipn.mx

57-29-60-00
Ext. 56652