[SOC] YARA

Introduction

Le **YARA** est un langage pour écrire des règles de détection de malware.

Il s'agit d'un langage simple et descriptif qui est adopté par le grand public.

Il est découpé par section qui ont chacune leur utilité.

Source

- TryHackMe - Yara
- Cuckoosandbox - Sandbox pour tester vos règles Yara
- PEfile - Scan les exécutables Windows PE

Annexes

- Github - Awesome Yara
- Valhalla - Base de règles Yara Opensource

Anatomie d'une règle

Anatomy of a yara Rule

Yara is a tool used to identify file, based on textual or binary pattern.

A rule consists of a set of strings and conditions that determine its logic.

Rules can be compiled with "yarac" to increase the speed of multiple Yara scans.

1 Import Module

Yara modules allow you to extend its functionality. The PE module can be used to match specific data from a PE:

- pe.number_of_exports
- pe.sections[0].name
- pe.imphash()
- pe.imports("kernel32.dll")
- pe.is_dll()

List of modules: pe, elf, hash, math, cuckoo, dotnet, time

2 Rule Name

The rule name identifies your Yara rule. It is recommended to add a meaningful name. There are different types of rules:

- Global rules: applies for all your rules in the file.
- Private rules: can be called in a condition of a rule but not reported.
- Rule tags: used to filter yara's output.

3 Metadata

Rules can also have a metadata section where you can put additional information about your rule.

- Author
- Date
- Description
- Etc...

4 Strings

The field strings is used to define the strings that should match your rule. It exists 3 type of strings:

- Text strings
- Hexadecimal strings
- Regex

5 Condition

Conditions are Boolean expressions used to match the defined pattern.

- Boolean operators:
  - and, or, not
  - <=, >=, ==, <, >, !=
- Arithmetic operators:
  - +, -, *, \, %
- Bitwise operators:
  - &, |, <<, >>, ^, ~
- Counting strings:
  - #string0 == 5
- Strings offset:
  - $string1 at 100

```
import "pe"

rule demo_rule : Tag1 Demo
{
    meta:
        author = "Thomas Roccia"
        description = "demo"
        hash = ""

    strings:
        $string0 = "hello" nocase wide
        $string1 = "world" fullword ascii
        $hex1 = { 01 23 45 ?? 89 ab cd ef }
        $re1 = /md5: [0-9a-zA-Z]{32}/

    condition:
        uint16(0) == 0x5A4D and filesize < 2000KB
        or pe.number_of_sections == 1 and
        any of ($string*) and (not $hex1 or $re1)
}
```

Text Strings

Text strings can be used with modifiers:

- nocase: case insensitive
- wide: encoded strings with 2 bytes per character
- fullword: non alphanumeric
- xor(0x01-0xff): look for xor encryption
- base64: base64 encoding

Hexadecimal

Hex strings can be used to match piece of code:

- Wild-cards: { 00 ?2 A? }
- Jump: { 3B [2-4] B4 }
- Alternatives: { F4 (B4 | 56) }

Regex

Regular expression can also be used and defined as text strings but enclosed in forward slash.

Advanced Condition

- Accessing data at a given position: uint16(0) == 0x5A4D
- Check the size of the file: filesize < 2000KB
- Set of strings: any of ($string0, $hex1)
- Same condition to many strings: for all of them : (# > 3)
- Scan entry point: $value at pe.entry_point
- Match length: `!rel[1] == 32`
- Search within a range of offsets: $value in (0..100)

@FR0GGER_
Thomas Roccia

Manuel

Installation

```
apt install -y yara
```

Meta

Cette section permet de donner des informations complémentaires qui ne seront pas interprêtés comme le ferait un commentaire dans du code.

Par exemple on peut utiliser le mot-clé **desc**, pour donner une description à notre règle afin qu'elle soit plus explicite pour les utilisateurs.

Strings

Cette section permet de détecter des chaînes de caractères présente dans les fichiers.

Voici un exemple d'utilisation :

```
rule helloworld_checker{
	strings:
		$hello_world = "Hello World!"

	condition:
		$hello_world
}
```

On peut aussi détecter des chaînes multiples :

```
rule helloworld_checker{
	strings:
		$hello_world = "Hello World!"
		$hello_world_lowercase = "hello world"
		$hello_world_uppercase = "HELLO WORLD"

	condition:
		any of them
}
```

Opérateurs

Comme dans les langages de programmation traditionnels, on peut utiliser des opérateurs pour nos conditions :

```
rule helloworld_checker{
    strings:
        $hello_world = "Hello World!"

    condition:
        #hello_world <= 10
}
```

Opérateurs	Descriptions
<=	Plus petit ou égal
>=	Plus grand ou égal
!=	Différent de

Combinaisons

On peut utiliser les mot-clés suivants pour combiner nos conditions :

Mot-clés	Descriptions
and	Les deux conditions doivent être valides
or	Au moins l'une des deux conditions doit être valide
not	Inverse la condition (true devient false et false devient true)

Voici un exemple pour vérifier si la chaîne est présente et si la taille du fichier est inférieure à 10KB :

```
rule helloworld_checker{
    strings:
        $hello_world = "Hello World!"

    condition:
        $hello_world and filesize < 10KB
}
```

Lancer le scan

```
yara <RULE>.yar <FILE_TO_SCAN>
```

> Si la règle match, la commande renverra le nom de la règle qui a matchée ainsi que le nom du fichier qui a matché.

Scanners d'IOC basés sur Yara

- Loki
- THOR
- Fenrir
- YAYA

Loki

Mettre à jour la base de signature

```
python loki.py --update
```

Lancer un scan d'un dossier

```
python loki.py -p <DIR>
```

YarGen

Cet outil permet de créer une règle Yara à partir d'un ou plusieurs fichiers connus pour être malveillants.

Il va se baser sur les chaînes de caractères et les informations pour générer la règle qui va détecter le ou les fichiers.

Téléchargement

- Github - YarGen

Mettre à jour l'outil

```
python3 yarGen.py --update
```

> Cela va mettre à jour la base avec les chaînes et les opcodes.

Créer une règle

```
python3 yarGen.py -m <FILE_PATH> --excludegood -o <OUTPUT.yar>
```

> Bien que l'outil soit fonctionnel, il est recommandé d'éditer la règle pour supprimer les chaînes qui pourraient lever des faux-positifs.

Revision #13
Created 23 May 2024 07:24:50 by Elieroc
Updated 23 May 2024 10:34:32 by Elieroc