YARA and YARA Rules

YARA is a powerful pattern-matching tool and rule format used for identifying and classifying files based on specific patterns, characteristics, or content. SOC analysts commonly use YARA rules to detect and classify malware samples, suspicious files, or indicators of compromise (IOCs).

YARA rules are typically written in a rule syntax that defines the conditions and patterns to be matched within files. These rules can include various elements, such as strings, regular expressions, and Boolean logic operators, allowing analysts to create complex and precise detection rules. It's important to note that YARA rules can recognize both textual and binary patterns, and they can be applied to memory forensics activities as well.

When applied, YARA scans files or directories and matches them against the defined rules. If a file matches a specific pattern or condition, it can trigger an alert or warrant further examination as a potential security threat.

YARA rules are especially useful for SOC analysts when analyzing malware samples, conducting forensic investigations, or performing threat hunting activities. The flexibility and extensibility of YARA make it a valuable tool in the cybersecurity community.

Usages of Yara

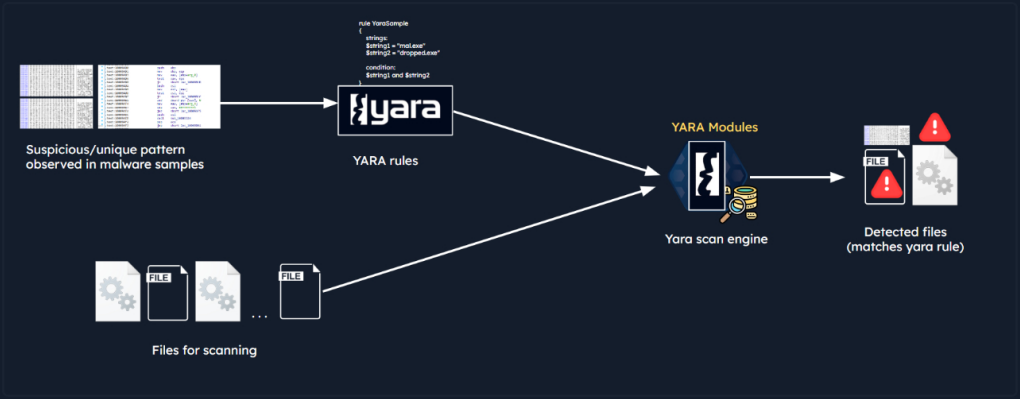
- **Malware Detection and Classification**: YARA is commonly used for detecting and identifying malware samples based on specific patterns, characteristics, or indicators. YARA rules can be created to match against known malware signatures, behaviors, or file properties, aiding in the identification of malicious files and potentially preventing further compromise. Within the scope of digital forensics, YARA can also identify suspicious or malicious patterns in captured memory images.
- **File Analysis and Classification**: YARA is valuable for analyzing and classifying files based on specific patterns or attributes. Analysts can create YARA rules to categorize files by different file formats or file type, version, metadata, packers or other characteristics. This capability is useful in forensic analysis, malware research, or identifying specific file types within large datasets.
- **Indicator of Compromise (IOC) Detection**: YARA can be employed to search for specific indicators of compromise (IOCs) within files or directories. By defining YARA rules that target specific IOC patterns, such as file names, registry keys, or network artifacts, security teams can identify potential security breaches or ongoing attacks.
- **Community-driven Rule Sharing**: With YARA, we have the ability to tap into a community that regularly contributes and shares their detection rules. This ensures that we constantly update and refine our detection mechanisms.
- **Create Custom Security Solutions**: By combining YARA rules with other techniques like static and dynamic analysis, sandboxing, and behavior monitoring, effective security solutions can be created.
- **Custom Yara Signatures/Rules**: YARA allows us to create custom rules tailored to our

organization's specific needs and environment. By deploying these custom rules in our security infrastructure, such as antivirus or endpoint detection and response (EDR) solutions, we can enhance our defense capabilities. Custom YARA rules can help identify unique or targeted threats that are specific to our organization's assets, applications, or industry.

- **Incident Response**: YARA aids in incident response by enabling analysts to quickly search and analyze files or memory images for specific patterns or indicators. By applying YARA rules during the investigation process, analysts can identify relevant artifacts, determine the scope of an incident, and gather critical information to aid in remediation efforts.
- **Proactive Threat Hunting**: Instead of waiting for an alert, we can leverage YARA to perform proactive searches across our environments, searching for potential threats or remnants of past infections.

How Does YARA Work?

In summary, the YARA scan engine, equipped with YARA modules, scans a set of files by comparing their content against the patterns defined in a set of rules. When a file matches the patterns and conditions specified in a YARA rule, it is considered a detected file. This process allows analysts to efficiently identify files that exhibit specific behaviors or characteristics, aiding in malware detection, IOC identification, and threat hunting. This flow is demonstrated in the diagram below.



In the above diagram, we can see that the YARA scan engine, using YARA modules, matches patterns defined in a set of rules against a set of files, resulting in the detection of files that meet the specified patterns and conditions. This is how YARA rules are helpful to identify threats. Let's understand this in more detail.

- **Set of Rules (containing suspicious patterns)**: First of all, we have one or more YARA rules, which are created by security analysts. These rules define specific patterns, characteristics, or indicators that need to be matched within files. Rules can include strings, regular expressions, byte sequences, and other indicators of interest. The rules are typically stored in a YARA rule file format (e.g., `.yara` or `.yar` file) for easy management and reuse.
- **Set of Files (for scanning)**: A set of files, such as executables, documents, or other binary or text-based files, are provided as input to the YARA scan engine. The files can be stored on a local disk, within a directory, or even within memory images or network traffic captures.
- **YARA Scan Engine**: The YARA scan engine is the core component responsible for performing the actual scanning and matching of files against the defined YARA rules. It utilizes YARA modules, which are sets of algorithms and techniques, to efficiently compare the content of files against the patterns specified in the rules.
- **Scanning and Matching**: The YARA scan engine iterates through each file in the set, one at a time. For each file, it analyzes the content byte by byte, looking for matches against the patterns defined in the YARA rules. The YARA scan engine uses various matching techniques, including string matching, regular expressions, and binary matching, to identify patterns

and indicators within the files.

- **Detection of Files**: When a file matches the patterns and conditions specified in a YARA rule, it is considered a detected file. The YARA scan engine records information about the match, such as the matched rule, the file path, and the offset within the file where the match occurred and provides output indicating the detection, which can be further processed, logged, or used for subsequent actions.

YARA Rule Structure

Let's dive into the structure of a YARA rule. YARA rules consist of several components that define the conditions and patterns to be matched within files. An example of a YARA rule is as follows:

Code: yara

```
rule my_rule {

    meta:
        author = "Author Name"
        description = "example rule"
        hash = ""

    strings:
        $string1 = "test"
        $string2 = "rule"
        $string3 = "htb"

    condition:
        all of them
}
```

Each rule in YARA starts with the keyword `rule` followed by a `rule identifier`. Rule identifiers are case sensitive where the first character cannot be a digit, and cannot exceed 128 characters.

The following keywords are reserved and cannot be used as an identifier:

all	and	any	ascii	at	base64	base64wide	condition
contains	endswith	entrypoint	false	filesize	for	fullword	global
import	icontains	iendswith	iequals	in	include	int16	int16be
int32	int32be	int8	int8be	istartswith	matches	meta	nocase
none	not	of	or	private	rule	startswith	strings
them	true	uint16	uint16be	uint32	uint32be	uint8	uint8be
wide	xor	defined					

Now, let's go over the YARA rule structure using a rule that identifies strings associated with the WannaCry ransomware as an example. The rule below instructs YARA to flag any file containing all three specified strings as `Ransomware_WannaCry`.

Code: yara

```
rule Ransomware_WannaCry {

    meta:
        author = "Madhukar Raina"
        version = "1.0"
        description = "Simple rule to detect strings from WannaCry ransomware"
        reference = "https://www.virustotal.com/gui/file/ed01ebfbc9eb5bbea545af4d01bf5f107166184048

    strings:
        $wannacry_payload_str1 = "tasksche.exe" fullword ascii
        $wannacry_payload_str2 = "www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com" ascii
```

```
        $wannacry_payload_str3 = "mssecsvc.exe" fullword ascii

    condition:
        all of them
}
```

That's a basic structure of a YARA rule. It starts with a `header` containing `metadata`, followed by `conditions` that define the context of the files to be matched, and a `body` that specifies the patterns or indicators to be found. The use of metadata and `tags` helps in organizing and documenting the rules effectively.

YARA Rule Breakdown:

1. `Rule Header`: The rule header provides metadata and identifies the rule. It typically includes:
    - `Rule name`: A descriptive name for the rule.
    - `Rule tags`: Optional tags or labels to categorize the rule.
    - `Rule metadata`: Additional information such as author, description, and creation date.

    Example:

    Code: yara

    ```
    rule Ransomware_WannaCry {
        meta:
      ...
    }
    ```

2. `Rule Meta`: The rule meta section allows for the definition of additional metadata for the rule. This metadata can include information about the rule's author, references, version, etc.

    Example:

    Code: yara

    ```
    rule Ransomware_WannaCry {
        meta:
            author = "Madhukar Raina"
            version = "1.0"
            description = "Simple rule to detect strings from WannaCry ransomware"
            reference =     "https://www.virustotal.com/gui/file/ed01ebfbc9eb5bbea545af4d01bf5f107
        ...
    }
    ```

3. `Rule Body`: The rule body contains the patterns or indicators to be matched within the files. This is where the actual detection logic is defined.

    Example:

    Code: yara

    ```
    rule Ransomware_WannaCry {

        ...

        strings:
            $wannacry_payload_str1 = "tasksche.exe" fullword ascii
            $wannacry_payload_str2 = "www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com" ascii
            $wannacry_payload_str3 = "mssecsvc.exe" fullword ascii

        ...

    }
    ```

4. `Rule Condition`: Rule conditions define the context and characteristics of the files to be matched. Conditions

4. `Rule Conditions`: Rule conditions define the context or characteristics of the files to be matched. Conditions can be based on file properties, strings, or other indicators. Conditions are specified within the condition section.

   Example:

   Code: yara

   ```
   rule Ransomware_WannaCry {
       ...

       condition:
           all of them
   }
   ```

   In this YARA rule, the condition section simply states `all of them`, which means that all the strings defined in the rule must be present for the rule to trigger a match.

   Let us provide one more example of a condition which specifies that the file size of the analyzed file must be less than `100` kilobytes (KB).

   Code: yara

   ```
       condition:
           filesize < 100KB and (uint16(0) == 0x5A4D or uint16(0) == 0x4D5A)
   ```

   This condition also specifies that the first 2 bytes of the file must be either `0x5A4D` (ASCII `MZ`) or `0x4D5A` (ASCII `ZM`), by using `uint16(0)`:

   Code: yara

   ```
   uint16(offset)
   ```

   Here's how `uint16(0)` works:

   - `uint16`: This indicates the data type to be extracted, which is a `16`-bit unsigned integer (`2` bytes).
   - `(0)`: The value inside the parentheses represents the offset from where the extraction should start. In this case, `0` means the function will extract the 16-bit value starting from the beginning of the data being scanned. The condition uses uint16(0) to compare the first 2 bytes of the file with specific values.

It's important to note that YARA provides a flexible and extensible syntax, allowing for more advanced features and techniques such as modifiers, logical operators, and external modules. These features can enhance the expressiveness and effectiveness of YARA rules for specific detection scenarios.

Remember that YARA rules can be customized to suit our specific use cases and detection needs. Regular practice and experimentation will further enhance our understanding and proficiency with YARA rule creation. YARA documentation contain more details in depth.