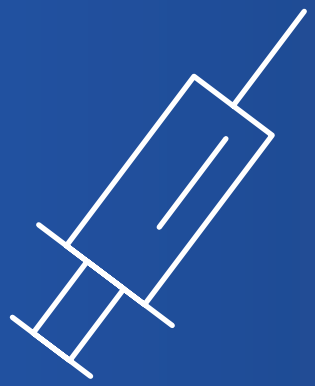


INTEZER

GENERATE ADVANCED YARA RULES BASED ON CODE REUSE

The purpose of this white paper is to define the challenges involved in writing effective YARA rules. It will explain how identifying code reuse between malicious files can be used to automatically produce advanced YARA rules to increase the accuracy of malware detection and classification and improve threat hunting capabilities.

An Introduction to YARA

YARA, short for "yet another recursive acronym", is a tool used in malware detection and classification. Malware researchers leverage YARA to create descriptions of malware families based on textual or binary patterns. Each description, or signature, is a "rule" consisting of a set of strings and a boolean expression to determine its logic. When written effectively YARA rules identify commonalities in malware and classify malicious files to other forms of malware that display similar patterns. More advanced YARA rules can be used to find additional variants of malware and hunt for samples.

While most security vendors use their own signature mechanisms, YARA is an open standard tool which can be used with many platforms and applied to different use cases. For example, YARA enables an organization to create detections of its own, which is relevant for identifying targeted malware that generic or classic signatures often struggle to detect.

This poses a significant advantage for security teams since they do not need to wait for signature updates from their security vendors. Security teams can create personalized, sophisticated YARA signatures even for targeted attacks and customly created malware.

The Challenges in Writing Effective YARA Rules

The vast majority of YARA rules available today are simple-based rules focused on string reuse found in a malware's binary. Strings can include a log message or hard-coded user agent, which are criteria not guaranteed to be unique. Therefore they can result in false positives and can be easily replaced or encrypted by the adversary to avoid detection.

The most effective YARA rules are designed to achieve high detection and classification rates while reducing the number of false positives. Researchers must select the right textual or binary patterns to optimize detection results and accurately classify a file to its respective malware family. This is difficult because files share hundreds and even thousands of strings and it is imperative that rules are not too generic nor too specific.

For example, if a researcher is defining a YARA rule based on a file that contains an embedded library, and the rule is based on the generic library alone – which is not a malicious piece of code in itself – the researcher will receive a hit on every single file that uses this library, in other words increasing the number of false positives generated.

The ideal signature will have the right balance, broad enough to identify many variants of the malware but specific enough to avoid false positives.

Challenge:

If a YARA rule is created based on the file's trusted code, many false positives will be generated.

On the other hand, ensuring a YARA signature is not too specific is also important. Taking a piece of code as it is, for example, and developing a rule that contains all of the code's addresses will make the YARA rule very specific to this particular file only. The rule will not likely generate hits for other or future variants of the malware that contain different values.

Writing Advanced YARA Rules is Difficult to Scale

More complex YARA rules can be created by incorporating features such as wild cards (a type of hexadecimal string), case-insensitive strings and regular expressions. While advanced YARA rules can be powerful instruments for detecting malware, writing them requires a high degree of technical ability and an understanding of YARA, and can be a time consuming process.

Writing advanced YARA rules and malware analysis in general is difficult to scale at high volumes, especially since not every organization has access to a team of highly-skilled researchers or reverse engineers. How can a security team create advanced YARA rules and still achieve automation, especially if the organization is faced with a high volume of alerts? The answer lies in studying patterns in code reuse.

Defining Code Reuse

Almost every software or malware today is comprised of previously written code. Developers of trusted applications will employ code reuse to make their work more efficient and to bring tools to market faster. The same approach applies for malware authors. As attackers write more malware they will establish code patterns. For defenders this provides an opportunity for attribution and identifying threat actor capabilities.

Code similarity analysis breaks down a given file into thousands of tiny fragments of code, or genes. Identifying code that was used in previous attacks can provide critical insights for security teams, including:

- Accurately determine the intent of the software. For example, is the file trusted or malicious?
- Classify a malicious file to its relevant malware family.
- Uncover the level of sophistication and potential risk of the threat. For example, are you dealing with a common banking trojan, a sophisticated APT or a nation-state sponsored attack? The answer will shape your response.
- Make attribution to the threat actor responsible for creating the malware.
- Not only can studying patterns in code reuse identify the origin of any given file, it can highlight unique, never-before-seen code, which can detect new threats that have been written from scratch.

Generate Automatic, Advanced YARA Rules with Code Reuse

Just as attackers reuse code to deploy new malware, defenders can identify patterns in code reuse to create advanced YARA rules. Here's how:

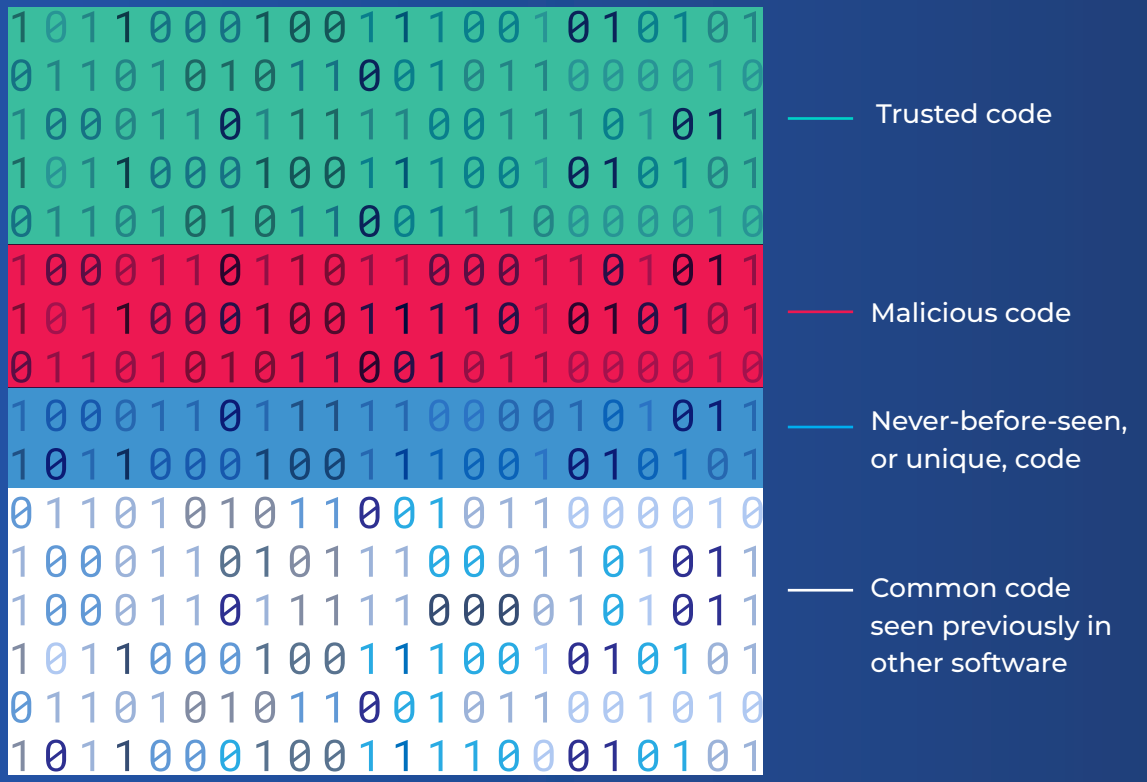
- By identifying a malicious file's unique binary code, a signature can be produced for detecting only those genes. This will enable detection of the exact same malware with high confidence and a low false positive rate.
- Detecting samples of the same variants (the same opcodes with different addresses) can be replaced with wildcards.
- Detecting different variants of the same malware family or campaign (same baseline of code but with different functionalities or capabilities) to look for a partial match.
- Detecting future variants that the attacker may release based on code reuse.

Example:

The below example represents a file that has been disassembled into tiny pieces of binary code, or genes.
Creating a YARA rule based on a file's trusted code will result in false positives. At the same time, developing a rule that contains all of the code's addresses will make the rule very specific to this particular file only. The rule will not likely generate hits for other or future variants of the malware that contain different values.

By identifying the malicious and unique code, a researcher can create a YARA rule that will achieve very accurate results.

- Receive hits on different hashes that contain the exact same code.
- Receive hits on different hashes that contain some but not all of the original code. This is useful for identifying new variants that are similar to the malware that the signature is based on.



Sample YARA Rule for TrickBot Malware

The following example demonstrates a code-based YARA rule produced for TrickBot, a common financial trojan. With the code-based YARA signature in place, any future sample or variant that reuses some aspects of TrickBot's code will be detected.

a4dfd173610d318acb4784645cf5e712d552b51d0c8cf10b2c4414d0486af27d.yar

```
rule Intezer_Vaccine_a4dfd173610d318acb4784645cf5e712d552b51d0c8cf10b2c4414d0486af27d
{
    meta:
        copyright = "Intezer Labs"
        description = "Automatic YARA vaccination rule created based on the file's genes"
        author = "Intezer Labs"
        reference = "https://analyze.intezer.com"
        date = "2019-06-02"
        sha256 = "a4dfd173610d318acb4784645cf5e712d552b51d0c8cf10b2c4414d0486af27d"
    strings:
        $4220080_32 = { 33 ?? BF ?? ?? ?? ?? 8D ?? ?? F3 ?? AA 33 ?? C1 ?? ?? 8D ?? ?? ?? ?? ?? 8A ?? ?? 8B ?? 84 ?? 74 }
        $4224961_26 = { 8B ?? ?? ?? 8B ?? 03 ?? ?? 8B ?? 8B ?? 03 ?? ?? 0F B7 ?? ?? 03 ?? ?? 8B ?? EB }
        $4208464_21 = { 5? 8B ?? 8B ?? ?? 83 ?? ?? 5? 8B ?? ?? 5? 3D ?? ?? ?? ?? 0F 87 }
        $4224824_21 = { 8B ?? 0F B7 ?? ?? 03 ?? B8 ?? ?? ?? ?? 66 ?? ?? ?? ?? ?? 0F 85 }
        $4209008_14 = { 0F B7 ?? 66 ?? ?? ?? 83 ?? ?? 66 ?? ?? 75 }

    condition:
        4 of them
}

```

Sample YARA Rule for WannaCry Variant

This is an example of a YARA signature that was produced via code from a previous WannaCry sample. Deployed in May 2017, WannaCry is one of the largest, high profile ransomware attacks in history, infecting over 200,000 computers across 150 countries.

bf293bda73c5b4c1ec66561ad20d7e2bc6692d051282d35ce8b7b7020c753467(1).yar

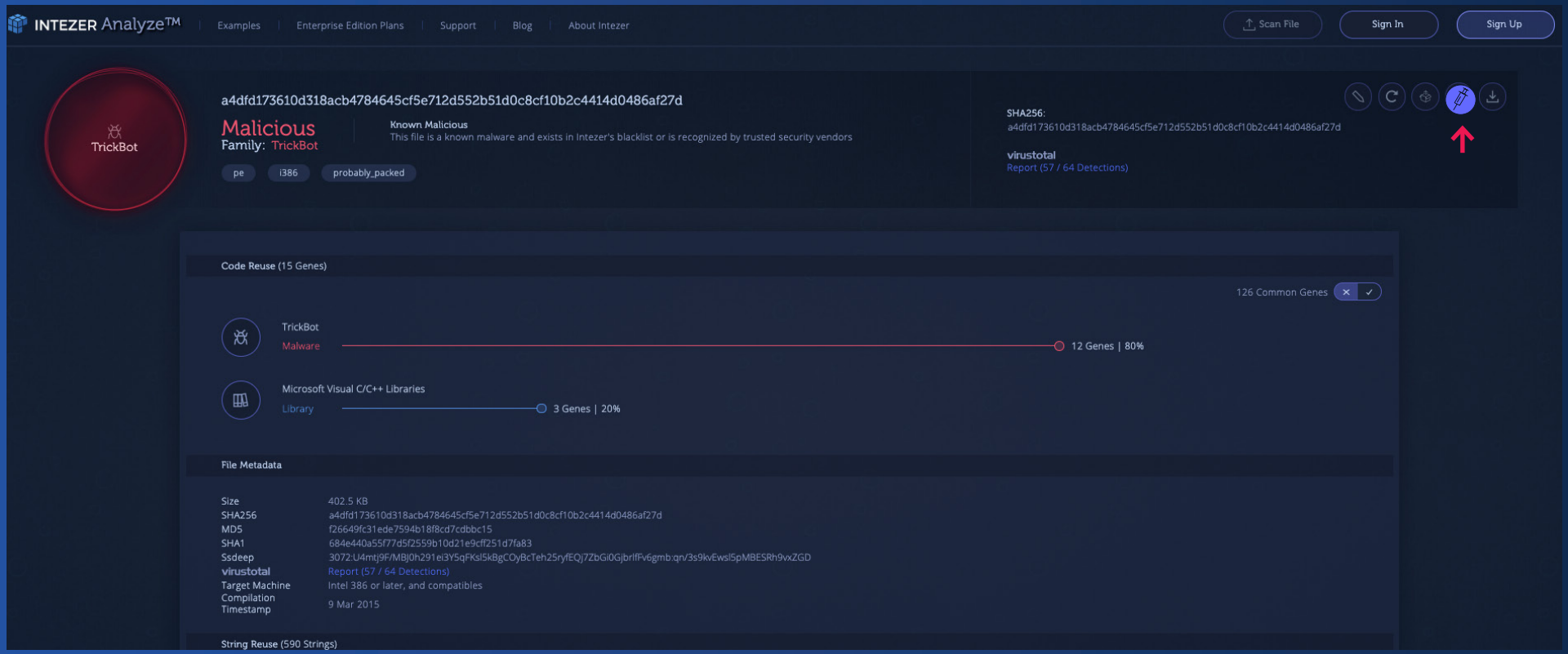
```
rule Intezer_Vaccine_bf293bda73c5b4c1ec66561ad20d7e2bc6692d051282d35ce8b7b7020c753467
{
    meta:
        copyright = "Intezer Labs"
        description = "Automatic YARA vaccination rule created based on the file's genes"
        author = "Intezer Labs"
        reference = "https://analyze.intezer.com"
        date = "2019-06-02"
        sha256 = "bf293bda73c5b4c1ec66561ad20d7e2bc6692d051282d35ce8b7b7020c753467"
    strings:
        $4237064_289 = { 33 ?? 83 ?? ?? 8A ?? 33 ?? 8A ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 33 ?? 8A ?? ?? ?? 33 ?? 8B ?? ?? 8B ??
        ?? ?? ?? ?? ?? 8B ?? 25 ?? ?? ?? ?? 33 ?? 8B ?? ?? ?? ?? ?? ?? 33 ?? 8A ?? ?? ?? 33 ?? 33 ?? 33 ?? 8A ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 8B
        ?? ?? ?? ?? ?? ?? 33 ?? 33 ?? 8A ?? 8B ?? ?? ?? ?? ?? ?? 8B ?? ?? ?? 33 ?? 8B ?? 81 E? ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 33 ?? 8A ?? ??
        ?? 33 ?? 8B ?? ?? 33 ?? 33 ?? 8A ?? 8B ?? ?? ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 33 ?? 33 ?? 8A ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 8B ?? 81 E?
        ?? ?? ?? ?? 33 ?? 8B ?? ?? ?? ?? ?? ?? 8B ?? ?? 33 ?? 33 ?? 33 ?? 8A ?? 33 ?? 8A ?? ?? ?? 89 ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 8B ?? ?? ??
        8B ?? ?? ?? ?? ?? ?? 33 ?? 8A ?? ?? ?? 33 ?? 8B ?? ?? ?? ?? ?? ?? 8B ?? ?? ?? 81 E? ?? ?? ?? ?? 33 ?? 89 ?? ?? ?? 8B ?? ?? ?? ?? ?? ??
        8B ?? 33 ?? 8B ?? ?? 33 ?? 47 89 ?? ?? ?? 89 ?? ?? ?? 89 ?? ?? ?? 0F 85 }
        $4243338_214 = { 8A ?? ?? ?? ?? ?? 5? 5? 88 ?? ?? ?? B9 ?? ?? ?? ?? 33 ?? 8D ?? ?? ?? 8D ?? ?? ?? F3 ?? 66 ?? AA 8D ?? ?? ?? BE ?? ?? ??
        ?? 5? 5? 89 ?? ?? ?? FF 1? ?? ?? ?? ?? 8A ?? ?? ?? ?? ?? 33 ?? 88 ?? ?? ?? ?? ?? ?? B9 ?? ?? ?? ?? 8D ?? ?? ?? ?? ?? ?? 8D ?? ?? ?? F3
        ?? 66 ?? AA 8D ?? ?? ?? ?? ?? ?? 5? 5? C6 ?? ?? ?? ?? 89 ?? ?? ?? FF 1? ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 6A ?? 5?
        8B ?? E8 ?? ?? ?? ?? 8D ?? ?? ?? 8B ?? 5? E8 ?? ?? ?? ?? 8A ?? ?? ?? ?? ?? ?? 8D ?? ?? ?? 6A ?? 57 8B ?? 88 ?? ?? ?? E8 ?? ?? ?? ?? 8D
        ?? ?? ?? ?? ?? ?? 8B ?? 5? E8 ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 5? 33 ?? 5? 64 ?? ?? ?? ?? ?? ?? 81 C? ?? ?? ?? ?? C3 }
        $4236863_189 = { 8B ?? ?? ?? 33 ?? 33 ?? 57 8A ?? 8A ?? ?? C1 ?? ?? 47 8B ?? ?? C1 ?? ?? 0B ?? 47 33 ?? 57 8A ?? 57 0B ?? 47 33 ?? 8B ??
        ?? 8A ?? 0B ?? 47 33 ?? 33 ?? 8A ?? 33 ?? 8B ?? 33 ?? 8A ?? ?? 89 ?? ?? ?? C1 ?? ?? 47 C1 ?? ?? 0B ?? 47 33 ?? 8A ?? 0B ?? 47 33 ?? 8A
        ?? 0B ?? 47 33 ?? 33 ?? 8A ?? 8A ?? ?? C1 ?? ?? 47 89 ?? ?? ?? C1 ?? ?? 0B ?? 47 33 ?? 8A ?? 0B ?? 47 33 ?? 8A ?? 0B ?? 8B ?? ?? 47 33
        ?? 33 ?? 89 ?? ?? ?? 8A ?? 8B ?? 33 ?? 8A ?? ?? C1 ?? ?? 47 C1 ?? ?? 0B ?? 47 33 ?? 8A ?? 8A ?? ?? 8B ?? ?? 0B ?? 33 ?? 8B ?? ?? ?? ??
        ?? 89 ?? ?? ?? 83 ?? ?? 89 ?? ?? ?? 0F 8E }
        $4212672_146 = { 57 8B ?? 6A ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 64 ?? ?? ?? ?? ?? 57 64 ?? ?? ?? ?? ?? ?? 81 E? ?? ?? ?? ?? 57 57 57 8B ??
        89 ?? ?? ?? ?? ?? A1 ?? ?? ?? ?? 89 ?? ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? 89 ?? ?? ?? ?? ?? 8A ?? ?? ?? ?? ?? 88 ?? ?? ?? ?? ?? C6 ?? ?? ??
        ?? ?? ?? C6 ?? ?? ?? ?? ?? ?? B9 ?? ?? ?? ?? 33 ?? 8D ?? ?? ?? ?? ?? F3 ?? 66 ?? AA 8D ?? ?? ?? ?? ?? 83 ?? ?? 33 ?? F2 ?? F7 ?? 47 89
        ?? ?? ?? ?? ?? 8B ?? E8 ?? ?? ?? ?? 85 ?? 0F 84 }
        $4199537_113 = { 57 6A ?? 8D ?? ?? ?? ?? ?? ?? 68 ?? ?? ?? ?? 57 FF D? 57 FF 17 ?? ?? ?? ?? 8A ?? ?? ?? ?? ?? B9 ?? ?? ?? ?? 33 ?? 8D ??
        ?? ?? ?? ?? ?? 88 ?? ?? ?? ?? ?? ?? 83 ?? ?? F3 ?? 66 ?? AA 8B ?? ?? ?? ?? ?? 8D ?? ?? ?? ?? ?? ?? 57 68 ?? ?? ?? ?? 57 FF 17 ?? ?? ??
        ?? 8D ?? ?? ?? ?? ?? ?? 68 ?? ?? ?? ?? 57 FF 17 ?? ?? ?? ?? ?? 8B ?? 83 ?? ?? 85 ?? 75 }
        $4201584_105 = { 81 E? ?? ?? ?? ?? 57 57 57 B9 ?? ?? ?? ?? BE ?? ?? ?? ?? 8D ?? ?? ?? 33 ?? F3 ?? B9 ?? ?? ?? ?? 8D ?? ?? ?? F3 ?? B9
        ?? ?? ?? ?? 8D ?? ?? ?? ?? ?? ?? ?? 88 ?? ?? ?? ?? ?? ?? 68 ?? ?? ?? ?? F3 ?? 66 ?? AA 8D ?? ?? ?? C7 ?? ?? ?? ?? ?? ?? ?? 57 FF 17 ?? ??
        ?? ?? 8B ?? ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? 83 ?? ?? 33 ?? 89 }
        $4244720_98 = { 64 ?? ?? ?? ?? ?? 6A ?? 68 ?? ?? ?? ?? 57 B8 ?? ?? ?? ?? 64 ?? ?? ?? ?? ?? ?? E8 ?? ?? ?? ?? 57 57 68 ?? ?? ?? ?? 8D ??
        ?? ?? E8 ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? ?? 8D ?? ?? ?? 83 ?? ?? 57 6A ?? 57 57 C7 ?? ?? ?? ?? ?? ?? ?? ?? ?? 8B
        ?? E8 ?? ?? ?? ?? 83 ?? ?? 85 ?? 0F 85 }
        $4227264_96 = { 81 E? ?? ?? ?? ?? 57 57 57 8B ?? 57 B9 ?? ?? ?? ?? 33 ?? 8D ?? ?? ?? F3 ?? A0 ?? ?? ?? ?? B9 ?? ?? ?? ?? 88 ?? ?? ?? ??
        ?? ?? 33 ?? 8D ?? ?? ?? ?? ?? ?? 89 ?? ?? ?? F3 ?? 66 ?? 8D ?? ?? ?? ?? ?? ?? C7 ?? ?? ?? ?? ?? ?? ?? 57 68 ?? ?? ?? ?? AA FF 17 ?? ??
        ?? ?? 83 ?? ?? 89 ?? ?? ?? 0F 84 }
        $4200408_88 = { B9 ?? ?? ?? ?? 33 ?? 8D ?? ?? ?? 88 ?? ?? ?? F3 ?? 66 ?? AA B9 ?? ?? ?? ?? 33 ?? 8D ?? ?? ?? ?? ?? ?? 88 ?? ?? ?? ?? ??
        ?? F3 ?? 66 ?? 57 8D ?? ?? ?? 68 ?? ?? ?? ?? 57 AA FF 17 ?? ?? ?? ?? 8D ?? ?? ?? 68 ?? ?? ?? ?? 57 FF 17 ?? ?? ?? ?? 8B ?? 83 ?? ?? 3B
        ?? 0F 84 }
        $4213168_85 = { 57 8B ?? 6A ?? 68 ?? ?? ?? ?? 68 ?? ?? ?? ?? 64 ?? ?? ?? ?? ?? 57 64 ?? ?? ?? ?? ?? ?? 83 ?? ?? 57 57 57 C7 ?? ?? ?? ??
        ?? ?? 33 ?? 89 ?? ?? 89 ?? ?? 89 ?? ?? 57 57 6A ?? 57 6A ?? 68 ?? ?? ?? ?? 8B ?? ?? 57 FF 17 ?? ?? ?? ?? 8B ?? 89 ?? ?? 83 ?? ?? 0F 84 }
        $4241472_84 = { 81 E? ?? ?? ?? ?? A0 ?? ?? ?? ?? 57 57 88 ?? ?? ?? B9 ?? ?? ?? ?? 33 ?? 8D ?? ?? ?? 68 ?? ?? ?? ?? F3 ?? 66 ?? 68 ?? ??
        ?? ?? 68 ?? ?? ?? ?? 8D ?? ?? ?? 68 ?? ?? ?? ?? 57 AA E8 ?? ?? ?? ?? 8B ?? ?? ?? ?? ?? 83 ?? ?? 8D ?? ?? ?? 57 FF D? 83 ?? ?? 0F 85 }
        $4241641_81 = { A0 ?? ?? ?? ?? B9 ?? ?? ?? ?? 88 ?? ?? ?? ?? ?? ?? 33 ?? 8D ?? ?? ?? ?? ?? ?? 68 ?? ?? ?? ?? F3 ?? 66 ?? 68 ?? ?? ?? ??
        68 ?? ?? ?? ?? 8D ?? ?? ?? ?? ?? ?? 68 ?? ?? ?? ?? 57 AA E8 ?? ?? ?? ?? 83 ?? ?? 8D ?? ?? ?? ?? ?? ?? 57 FF D? 83 ?? ?? 75 }
```

Intezer Analyze

Leveraging code similarity analysis (or, “Genetic Malware Analysis”) technology, Intezer Analyze can generate automatic YARA signatures based on a file’s code, enabling users to improve their threat hunting capabilities by detecting future variants of the malware. As highlighted above, code-based YARA signatures can effectively reduce false positives and save precious resources in the form of time and analyst efforts. This is particularly valuable for organizations dealing with a large volume of alerts.

This feature also allows for some more advanced usages. It can be used in several scenarios, including:

1. The user can adjust the thresholds of the rule. The default value is 70% of the entire code but it is possible to modify the rule to be more specific, or more flexible.
2. The user can combine or split different YARA signatures to build stronger rules to better fit his or her needs.
3. The user can add to the automated rules with a string reuse feature to make the signature even more powerful for threat hunting.



Key Takeaways

- Incorporating YARA into daily security operations can accelerate incident response time, classify malware, empower threat intelligence and improve detection capabilities by creating custom signatures.
- Traditional methods for writing effective YARA signatures have their challenges, including being too specific or generic with respect to a file's textual or binary patterns.
- In today's YARA landscape many signatures are based on string reuse. The challenge is to identify the "unique" strings, however even if this is achieved, these signatures are much less effective because strings can be easily manipulated, replaced or encrypted by the adversary to avoid detection.
- Automation of YARA signatures is difficult to scale at high volumes and many organizations do not have a dedicated team of malware analysts at hand to manually reverse engineer every file they encounter.
- Identifying malicious code seen in previous threats can be used to generate more accurate YARA signatures for detecting future variants or new malware.
- The Intezer Analyze platform enables users to automatically generate YARA rules based on binary code, rather than simple strings. Code-based YARA rules are the most effective since they are tolerant to modifications and are more equipped to detect variants of the same threat.