

[https://digitalguardian.com/blog/what-are-indicators-](https://digitalguardian.com/blog/what-are-indicators-compromise#:~:text=Indicators%20of%20compromise%20(IOCs)%20are,malware%20infections%2C%20or%20other%20threat)

[compromise#:~:text=Indicators%20of%20compromise%20\(IOCs\)%20are,malware%20infections%2C%20or%20other%20threat](https://digitalguardian.com/blog/what-are-indicators-compromise#:~:text=Indicators%20of%20compromise%20(IOCs)%20are,malware%20infections%2C%20or%20other%20threat)

DEFINITION OF INDICATORS OF COMPROMISE

Indicators of compromise (IOCs) are “pieces of forensic data, such as data found in system log entries or files, that identify potentially malicious activity on a system or network.” Indicators of compromise aid information security and IT professionals in detecting data breaches, malware infections, or other threat activity. By monitoring for indicators of compromise, organizations can detect attacks and act quickly to prevent breaches from occurring or limit damages by stopping attacks in earlier stages.

But, IOCs are not always easy to detect; they can be as simple as metadata elements or incredibly complex malicious code and content samples. Analysts often identify various IOCs to look for correlation and piece them together to analyze a potential threat or incident.

INDICATORS OF COMPROMISE VS. INDICATORS OF ATTACK

Indicators of attack are similar to IOCs, but rather than focusing on forensic analysis of a compromise that has already taken place, indicators of attack focus on identifying attacker activity while an attack is in process. Indicators of compromise help answer the question “What happened?” while indicators of attack can help answer questions like “What is happening and why?” A proactive approach to detection uses both IOAs and IOCs to discover security incidents or threats in as close to real time as possible.

KEY IOCs TO GET YOU STARTED:

There are several indicators of compromise that organizations should monitor.

[http://www.petermorin.com/2013/10/top-15-indicators-of-compromise-](http://www.petermorin.com/2013/10/top-15-indicators-of-compromise-ioc/#:~:text=Mobile%20Device%20Profile%20Changes&text=%E2%80%9CIf%20a%20managed%20mobile%20device,and%20CTO%20of%20Marble%20Security.)

[ioc/#:~:text=Mobile%20Device%20Profile%20Changes&text=%E2%80%9CIf%20a%20managed%20mobile%20device,and%20CTO%20of%20Marble%20Security.](http://www.petermorin.com/2013/10/top-15-indicators-of-compromise-ioc/#:~:text=Mobile%20Device%20Profile%20Changes&text=%E2%80%9CIf%20a%20managed%20mobile%20device,and%20CTO%20of%20Marble%20Security.)

1. Unusual Outbound Network Traffic

A common misperception is that traffic inside the network is secure. Look for suspicious traffic leaving the network. It's not just about what comes into your network; it's about outbound traffic as well.

Considering that the chances of keeping an attacker out of a network are difficult in the face of modern attacks, outbound indicators may be much easier to monitor.

So the best approach is to watch for activity within the network and to look for traffic leaving your perimeter. Compromised systems will often call home to command-and-control servers, and this traffic may be visible before any real damage is done.

2. Anomalies In Privileged User Account Activity

Attackers either escalate privileges of accounts they've already compromised or use that compromise to leapfrog into other accounts with higher privileges.

Changes in the behavior of privileged users can indicate that the user account in question is being used by someone else. Watching for changes -- such as time of activity, systems accessed, type or volume of information accessed -- will provide early indication of a breach.

3. Geographical Irregularities

Whether through a privileged account or not, geographical irregularities in log-ins and access patterns can provide good evidence that attackers are pulling strings from far away. Connections to countries that a company would normally not be conducting business which indicates sensitive data could be siphoned to another country.

Similarly, when one account logs in within a short period of time from different IPs around the world, that's a good indication of trouble. More often than not, this is a symptom of an attacker using a compromised set of credentials to log into confidential systems.

4. Swells In Database Read Volume

Once an attacker has made it into the crown jewels and seeks to retrieve information, there will be signs that someone has been mucking about data stores. One of them is a spike in database read volume. Eg : When the attacker attempts to extract the full credit card database, it will generate an enormous amount of read volume, which will be way higher than you would normally see for reads on the credit card tables.

5. HTML Response Sizes

If attackers use SQL injection to extract data through a Web application, the requests issued by them will usually have a larger HTML response size than a normal request. For example, if the attacker extracts the full credit card database, then a single response for that attacker might be 20 to 50 MB, where a normal response is only 200 KB.

6. Large Numbers Of Requests For The Same File

It takes a lot of trial and error to compromise a site -- attackers have to keep trying different exploits to find ones that stick. And when they find signs that an exploit might be successful, they'll frequently use different permutations to launch it.

So while the URL they are attacking will change on each request, the actual filename portion will probably stay the same. So you might see a single user or IP making 500 requests for 'join.php,' when normally a single IP or user would only request that page a few times max.

7. DNS Request Anomalies

One of the most effective red flags an organization can look for are patterns left by malicious DNS queries.

Command-and-control (C&C) traffic is often the most important traffic to an attacker because it allows their ongoing management of the attack and it needs to be secure so that security professionals can't easily take it over. The unique patterns of this traffic can be recognized and is a very standard approach to identifying a compromise.

"Seeing a large spike in DNS requests from a specific host can serve as a good indicator of potentially suspect activity," he says. "Watching for patterns of DNS requests to external hosts, and implementing appropriate filtering can help mitigate C&C over DNS

8. Unexpected Patching of Systems:

A patch is a set of changes to a computer program or its supporting data designed to update, fix, or improve it. This includes fixing security vulnerabilities and other bugs, with such patches usually being called bugfixes or bug fixes.

Patching is generally a good thing, but if a system is inexplicably patched without reason, that could be the sign that an attacker is locking down a system so that other bad guys can't use it for other criminal activity.

"Most attackers are in the business of making money from your data — they certainly don't want to share the profits with anyone else

9. Bundles Of Data In The Wrong Places

Attackers frequently aggregate data at collection points in a system before attempting exfiltration(Data exfiltration is the unauthorized copying, transfer or retrieval of data from a computer or server.) If you suddenly see large gigabytes of information and data where they should

not exist, particularly compressed in archive formats your company doesn't use, this is a telltale sign of an attack.

In general, files sitting around in unusual locations should be examined because they can point to an impending breach

Files in odd places, like the root folder of the recycle bin, are hard to find looking through Windows, but easy and quick to find with a properly crafted Indicator of Compromise search. Executable files in the temp folder is another one, often used during privilege escalation, which rarely has a legitimate existence outside of attacker activity.

10. Web Traffic with Inhuman Behavior

Web traffic that doesn't match up with normal human behavior shouldn't pass the sniff test.

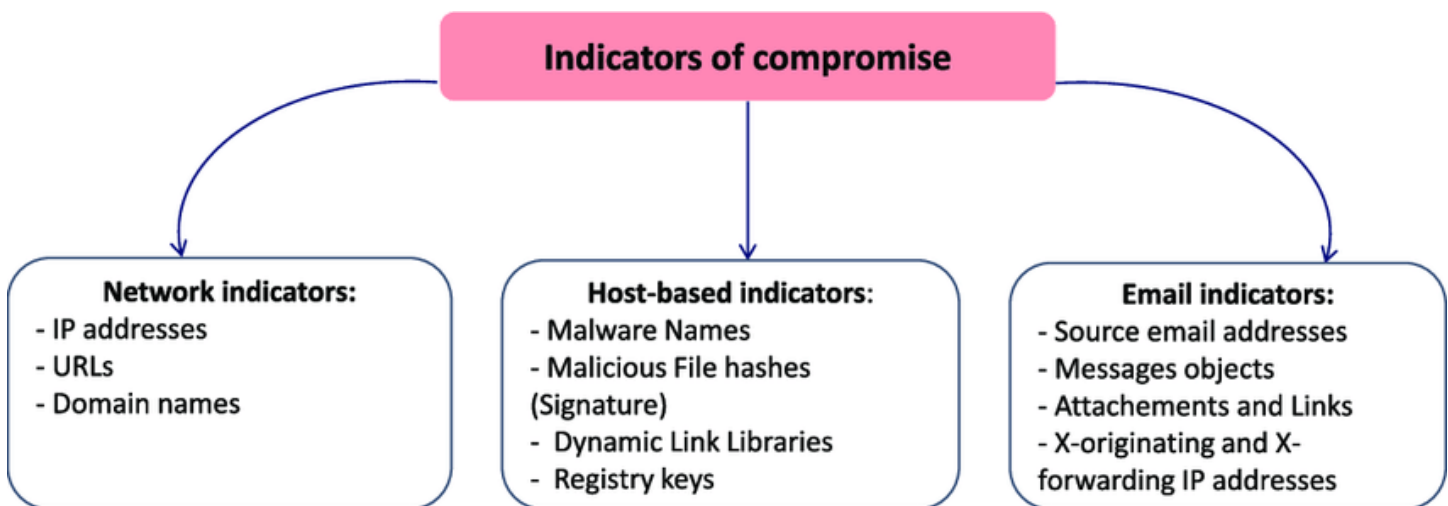
11. Signs Of DDoS Activity

Distributed denial-of-service attacks (DDoS) are frequently used as smokescreens to camouflage other attacks. If an organization experiences signs of DDoS, such as slow network performance, unavailability of websites, firewall failover, or back-end systems working at max capacity for unknown reasons, they shouldn't just worry about those immediate problems.

This presents new opportunities for cybercriminals to plant malware or steal sensitive data. As a result, any DDoS attack should also be reviewed for related data breach activity."

CATEGORIES OF IOCs:

<https://www.info-savvy.com/categories-of-indicators-of-compromise/>



IoCs are divided into four classes as given below:

Network Indicators

Network indicators are helpful for command and management and malware delivery. Examples of network indicators embody URLs, domain names, IP addresses, etc.

Host-Based Indicators

Host-based indicators are found by activity analysis on the infected. Samples of host-based indicators are filenames, file hashes, written record keys, Ds, mutes, etc.

Email Indicators

Attackers typically like email services to send malicious content to the target organization or individual. Such emails are most well-liked because of the benefit of use and comparative obscurity. Samples of email iocs have sender's email address, email subject, attachments or links, etc.

Behavioral Indicators

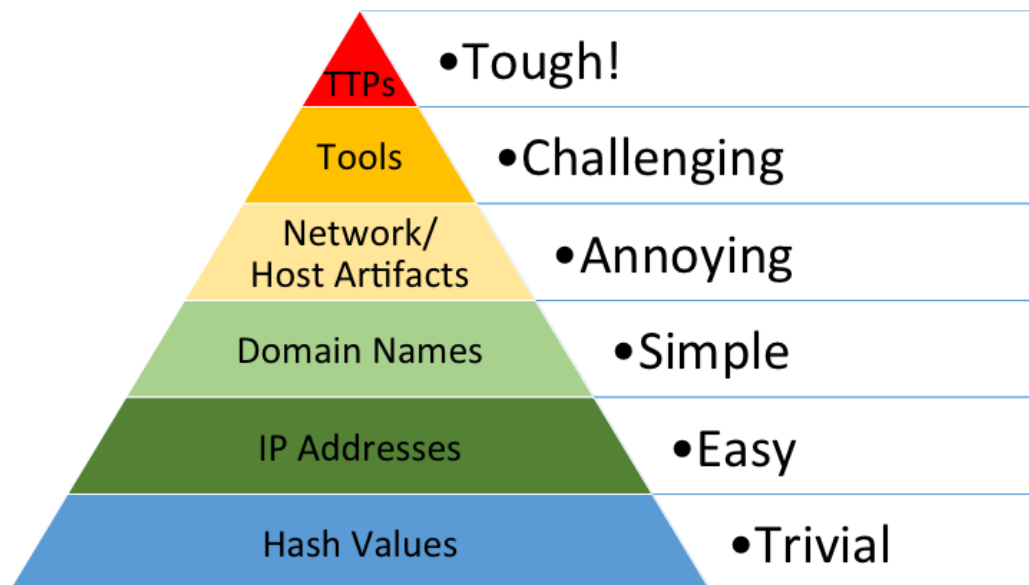
Generally, typical IoCs are helpful for characteristic indications of intrusion like malicious IP address, virus signatures, MOS hash, and name. Behavioral IoCs are used for characteristic specific behavior associated with malicious activities like code injection into the memory or running script of an application. Well-defined behaviors change broad protection to stop all current and future malicious activities. Samples of behavioral indicators embody document capital punishment Power Shell script, remote command execution, etc.

THE PYRAMID OF PAIN:

<https://www.threathunting.net/files/A%20Framework%20for%20Cyber%20Threat%20Hunting%20Part%201%20The%20Pyramid%20of%20Pain%20-%20Sqrrl.pdf>

The pyramid organizes IoCs in two ways:

1. How difficult (painful) is it to collect and apply the IoC to cyber defenses? Malicious hash values and IP addresses are relatively easy to acquire and integrate into security tools. TTPs are more difficult to identify and apply, as most security tools are not well suited to take advantage of them.
2. How much pain can the IoCs inflict on cyber-attacks? It is relatively easy for an attack to make unclear malware code and change the hash values. IP addresses can be dynamically changed with low cost. TTPs are sticky and expensive for an attack to change. As a result, security tools that leverage TTPs can inflict more pain on an attack.



1. Hash Values: SHA1, MD5 or other similar hashes that correspond to specific suspicious or malicious files. Often used to provide unique references to specific samples of malware or to files involved in an intrusion. It is so easy for hash values to change, and there are so many of them around, that in many cases it may not even be worth tracking them.

2. IP Addresses: IP addresses are quite literally the most fundamental indicator, but if they are using an anonymous proxy service like Tor or something similar, they may change IPs quite frequently and never even notice or care.

3. Domain Names: This could be either a domain name itself (e.g., “evil.net”) or maybe even a sub- or subsub-domain (e.g., “this.is.sooooo.evil.net”). They must be registered, paid for (even if with stolen funds) and hosted somewhere. That said, there are a large number of DNS providers out there with lenient registration standards.

4. Network Artifacts: In practice these are pieces of the activity that might tend to distinguish malicious activity from that of legitimate users. Typical examples might be URI patterns, C2 information embedded in network protocols, etc.

5. Host Artifacts: Observables caused by adversary activities on one or more of your hosts that would distinguish malicious activities from legitimate ones. These can be any distinctive identifier such as be registry keys or values known to be created by specific pieces of malware, files or directories dropped in certain places, etc.

6. Tools: Software used by the adversary to accomplish their mission. Mostly this will be things they bring with them, rather than software or commands that may already be installed on the computer. This would include utilities designed to create malicious documents for spear phishing, backdoors used to establish C2 or password crackers or other host-based utilities they may want to use post-compromise.

7. Tactics, Techniques and Procedures (TTPs): At the apex of the pyramid is how the adversary goes about accomplishing their mission, from exploration all the way through data exfiltration and at

every step in between. When you detect and respond at this level, you are operating directly on adversary behaviors, not against their tools.

USING INDICATORS OF COMPROMISE TO IMPROVE DETECTION AND RESPONSE

Monitoring for indicators of compromise enables organizations to better detect and respond to security compromises. Collecting and correlating IOCs in real time means that organizations can more quickly identify security incidents that may have gone undetected by other tools and provides the necessary resources to perform forensic analysis of incidents. If security teams discover recurrence or patterns of specific IOCs they can update their security tools and policies to protect against future attacks as well.

There is a push for organizations to report these analyses results in a consistent, well-structured manner to help companies and IT professionals automate the processes used in detecting, preventing, and reporting security incidents. Some in the industry argue that documenting IOCs and threats helps organizations and individuals share information among the IT community as well as improve incident response and computer forensics.

A way to observe similarity in malicious code is to leverage analyst insights by identifying files that possess some property in common with a particular file of interest. One way to do this is by using **YARA**. YARA has gained enormous popularity in recent years as a way for malware researchers and network defenders to communicate their knowledge about malicious files, from identifiers for specific families to signatures capturing common tools, techniques, and procedures (TTPs).

<https://virustotal.github.io/yara/>

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"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

The above rule is telling YARA that any file containing one of the three strings must be reported as *silent_banker*. This is just a simple example, more complex and powerful rules can be created by using wild-cards, case-insensitive strings, regular expressions, special operators and many other features that you'll find explained in YARA's documentation.

YARA is multi-platform, running on Windows, Linux and Mac OS X, and can be used through its command-line interface or from your own Python scripts with the yara-python extension.

<https://blog.malwarebytes.com/security-world/technology/2017/09/explained-yara-rules/#:~:text=YARA%20is%20a%20tool%20that,that%20look%20for%20certain%20characteristics.>

BASIC SYNTAX

Each rule has to start with the word *rule*, followed by the name or identifier. The identifier can contain any alphanumeric character and the underscore character, but the first character is not allowed to be a digit. There is a list of [YARA keywords](#) that are not allowed to be used as an

identifier because they have a predefined meaning.

```
rule rogues
{
  meta:
    created   = "09/11/2017 00:00:00"
    modified  = "09/11/2017 17:12:07"
    author    = "Metallica"
    Vendor    = "PC Smart Cleanup"
  strings:
    $textstring1 = "smart"  ascii wide nocase
    $textstring2 = "cleanup" ascii wide nocase
    $textstring3 = "longrun" ascii wide nocase
  condition:
    $textstring1 and @textstring2 and $textstring3
}
```

Condition

Rules are composed of several sections. The *condition* section is the only one that is required. This section specifies when the rule result is *true* for the object (file) that is under investigation. It contains a Boolean expression that determines the result. Conditions are by design Boolean expressions and can contain all the usual logical and relational operators. You can also include another rule as part of your conditions.

Strings

To give the *condition* section a meaning you will also need a *strings* section. The strings section is where you can define the strings that will be looked for in the file. Let's look at an easy example.

```
rule vendor
{
  strings:
    $text_string1 = "Vendor name" wide
    $text_string2 = "Alias name" wide
  condition:
    $text_string1 or $text_string2
}
```

The rule shown above is named *vendor* and looks for the strings "Vendor name" and "Alias name". If either of those strings is found, then the result of the rule is *true*.

There are several types of strings you can look for:

- Hexadecimal, in combination with wild-cards, jumps, and alternatives.
- Text strings, with modifiers: nocase, fullword, wide, and ascii.
- Regular expressions, with the same modifiers as text strings.

There are many more advanced conditions you can use. If you would like to know more you can find it in the <https://yara.readthedocs.io/en/v4.0.1/index.html>

Metadata

Metadata can be added to help identify the files that were picked up by a certain rule. The metadata identifiers are always followed by an equal sign and the set value. The assigned values can be strings, integers, or a Boolean value. Note that identifier/value pairs defined in the metadata section can't be used in the condition section, their only purpose is to store additional information about the rule.

Refer documentation for detailed syntax rules: <https://yara.readthedocs.io/en/v4.0.1/writingrules.html>

Cheat sheet: <https://medium.com/@nidhi.trivedi/yara-cheat-sheet-585eae339e63>

https://insights.sei.cmu.edu/sei_blog/2012/11/writing-effective-yara-signatures-to-identify-malware.html

BENEFITS OF REVERSE ENGINEERING FOR MALWARE ANALYSIS

Reverse engineering is arguably the most expensive form of analysis to apply to malicious files. It is also the process by which the greatest insights can be made against a particular malicious file. Since analysis time is so expensive, however, we constantly seek ways to reduce this cost. When classifying and identifying malware, therefore, it is useful to group related files together to cut down on analysis time and leverage analysis of one file against many files. To express such relationships between files, we use the concept of a "malware family", which is loosely defined as "a set of files related by objective criteria derived from the files themselves." Using this definition, we can apply different criteria to different sets of files to form a family.

For example, Scraze is considered a malware family. In this case, the objective criteria forming the family are the functions that Scraze files have in common. We generally find that it is desirable for objective criteria to have the following properties:

- The criteria should be necessary to the behavior of the malware. Ideal candidates are structural properties (such as a particular section layout or resources needed for the program to run) or behavioral properties (such as function bytes for important malware behavior).
- The criteria should also be sufficient to distinguish the malware family from other families. We often find that malicious files accomplish similar things but perhaps in different ways. For example, many malicious files detect that they are running in a virtual environment, and there have been many published techniques on how to implement this behavior. Since these techniques are published on the Internet, it is trivial for a malware author to incorporate them into his own programs, and we expect this to occur. Thus, using these published examples as criteria indicative of one particular malware family is probably not sufficient to distinguish that family.

APPLYING YARA SIGNATURES EFFECTIVELY TO MALWARE

Analyst experience and intuition can guide the selection of good criteria, as discussed above. The goal of developing these criteria is to use them to identify related files. In practice, we generally find

that good criteria for distinguishing and identifying malware families are excellent targets for creating signatures that identify the families.

One way to encode signatures that identify malware families is by using the open source tool **YARA**. YARA provides a robust language for creating signatures to identify malware. These signatures are encoded as text files, which makes them easy to read and communicate with other malware analysts. We have found three different types of criteria are most suitable for YARA signature development: *strings*, *resources*, and *function bytes*.

The simplest usage of YARA is to encode strings that appear in malicious files. The usefulness of matching strings, however, is highly dependent on which strings are chosen. For example, selecting strings that represent unique configuration items, or commands for a remote access tool, are likely to be indicative of a particular malware family. Conversely, strings in a malicious file that result from the way the file was created are generally poor candidates for YARA signatures.

Here is an example of a YARA signature for the malware family Scraze, based on strings derived from the malware:

```
rule Scraze
{
  strings:
    $strval1 = "C:\Windows\ScreenBlazeUpgrader.bat"
    $strval2 = "\ScreenBlaze.exe "
  condition:
    all of them
}
```

Another effective use of YARA is to encode resources that are stored in malicious files. These resources may include things like distinctive icons, configuration information, or even other files. To encode these resources as YARA signatures, we first extract the resources (using any available tool, for example **Resource Hacker**) and then convert the bytes of the resource to a hexadecimal string that can be represented directly in a YARA signature. Icons in particular make excellent targets for such signatures. However, resources are easily modified in a program. For example, Microsoft Windows allows a program's icon to change by simple copy/paste operations in Windows Explorer. The same care must be taken in selecting program resources as is taken when selecting strings.

Finally, an effective use of YARA signatures is to encode bytes implementing a function called by the malicious program. Functions tend to satisfy our desire for criteria both necessary and sufficient to describe the malware family. The best functions to encode are those that perform some action that indicates the overall character of the malware.

For example, the best function for malware whose primary purpose is to download another file may be one that performs the download or processes downloaded files. Likewise, for remote access tools, it may be a function to encrypt network communications or to process received commands. For viruses, it may be code (which may not even be an actual function) involved with decrypting a payload or re-infecting additional files. We may also identify packers by encoding bytes representing the unpack stub.

Here is an example of a YARA signature encoding part of a function used to check the integrity of an NSIS installer created with NSIS version 2.46.

```
rule NSIS_246
{
  strings:
    $NSIS_246_CheckIntegrity = { 57 53 E8 ?? ?? ?? ?? 85 C0 0F 84 ?? ?? ?? ??
83 3D ?? ?? ?? ?? 00 75 ?? 6A 1C 8D 45 D8 53 50 E8 ?? ?? ?? ?? 8B 45 D8 A9
F0 FF FF FF 75 ?? 81 7D DC EF BE AD DE 75 ?? 81 7D E8 49 6E 73 74 75 ?? 81
7D E4 73 6F 66 74 75 ?? 81 7D E0 4E 75 6C 6C 75 ?? 09 45 08 8B 45 08 8B 0D
?? ?? ?? ?? 83 E0 02 09 05 ?? ?? ?? ?? 8B 45 F0 3B C6 89 0D ?? ?? ?? ?? 0F
8F ?? ?? ?? ?? F6 45 08 08 75 ?? F6 45 08 04 75 ?? }
  condition:
    $NSIS_246_CheckIntegrity
}
```

LIMITATIONS IN CREATING YARA SIGNATURES

Regardless of the criteria used to create YARA signature, there are always limitations, especially for criteria derived from strings. For example, malware with statically coded password lists may have a large number of strings, including those that may seem to unique to a family. Moreover, since strings are easily mutable, a string (such as the filename or path into which a malicious file is installed, or common encoding strings used in HTTP requests) considered as indicative may change at any time. The analyst must assess the significance of the presence or absence of a particular string, rather than delegate responsibility to a YARA signature.

Likewise, when using code instead of data to create YARA signatures, we must also be aware of the limitations. It is important to use functions that are not provided from a common implementation detail, such as common libraries that have been statically linked into the program. It is also important to account for differences in malicious files due to process changes, such as recompilation. One way to sift out these differences is to use algorithms that normalize address references (for example, the PIC algorithm) when selecting function bytes.

Malicious code may change over time, so particular functions may come and go. It is therefore better to select a number of functions to encode as signatures and derive the "best" signatures by surveying matching files and refining the analysis as the importance of the signatures is revealed. This method is the essence of family analysis, and is an important application of YARA to malware analysis.

While YARA continues to gain in popularity, there aren't many guidelines on how to use it most effectively. The issue of false positives continues to trouble malware analysts. For example, YARA signatures may match files in which the specified criteria exist and yet do not possess the same semantics as expressed in the original file.

Another issue with using YARA signatures is that they can be fragile in the face of changing codebases. A malware author can and does change his/her code to suit an ever-shifting set of goals, and keeping up with these changes is particularly challenging. While YARA signatures are a powerful means of capturing and communicating analyst insights, care must be taken that they do not drift too far away from the current reality of a particular malware family.

Installation on windows:

Follow the link and download the highlighted file in the screenshot.





<https://github.com/VirusTotal/yara/releases/tag/v4.0.1>

YARA v4.0.1

 plusvic released this 24 days ago · 10 commits to master since this release

- Update sandboxed API (#1276)
- BUGFIX: Fix regression in exports parsing in PE module (2bf67e6)
- BUGFIX: Fix unaligned accesses in ARM (e1654ae)

▼ Assets 4

 yara-v4.0.1-1323-win32.zip	1.35 MB
 yara-v4.0.1-1323-win64.zip	1.98 MB
 Source code (zip)	
 Source code (tar.gz)	

Extract yara64 and yarac64 files to a convenient location (eg: Desktop)

Writing your own rule:

- Open notepad and type the following rule:

```
rule dummy {  
  
    condition: filesize > 500KB  
  
}
```

Save it as filename.yara i.e dummy.yara

This rule only checks if the filesize is more than 500KB and returns true.

- To run:
 1. Open command prompt and cd to yara64 location. (desktop)
 2. General command :

yara64 [options] [yara rule file] [file to be investigated]
 3. To see all options type yara64 --help

```

C:\Users\windows\Desktop>yara64 --help
YARA 4.0.1, the pattern matching swiss army knife.
Usage: yara [OPTION]... [NAMESPACE:]RULES_FILE... FILE | DIR | PID

Mandatory arguments to long options are mandatory for short options too.

  -C, --atom-quality-table=FILE      path to a file with the atom quality table
  -C, --compiled-rules              load compiled rules
  -c, --count                       print only number of matches
  -d, --define=VAR=VALUE            define external variable
  --fail-on-warnings               fail on warnings
  -f, --fast-scan                   fast matching mode
  -h, --help                       show this help and exit
  -i, --identifier=IDENTIFIER       print only rules named IDENTIFIER
  -l, --max-rules=NUMBER            abort scanning after matching a NUMBER of rules
  --max-strings-per-rule=NUMBER     set maximum number of strings per rule (default=10000)
  -x, --module-data=MODULE=FILE     pass FILE's content as extra data to MODULE
  -n, --negate                     print only not satisfied rules (negate)
  -w, --no-warnings                disable warnings
  -m, --print-meta                  print metadata
  -D, --print-module-data           print module data
  -e, --print-namespace             print rules' namespace
  -S, --print-stats                 print rules' statistics
  -s, --print-strings               print matching strings
  -L, --print-string-length         print length of matched strings
  -g, --print-tags                  print tags
  -r, --recursive                   recursively search directories (follows symlinks)
  --scan-list                       scan files listed in FILE, one per line
  -k, --stack-size=SLOTS           set maximum stack size (default=16384)
  -t, --tag=TAG                     print only rules tagged as TAG
  -p, --threads=NUMBER             use the specified NUMBER of threads to scan a directory
  -a, --timeout=SECONDS            abort scanning after the given number of SECONDS
  -v, --version                     show version information

```

We will use `-s` to print the strings that are matched and `-r` for a recursive scan.

4. Type `yara64 -s -r dummy.yara .` (last dot for the current directory)

```

C:\Users\windows\Desktop>yara64 -s -r dummy.yara .
dummy .\Data Analytics\city_day.csv
dummy .\Data Analytics\Internship Data Analytics.pdf
dummy .\Data Analytics\plots.docx
dummy .\Data Analytics\REPORT_NehaPatil.docx
dummy .\Data Analytics\FINAL.csv
dummy .\DSA Princeton\percolation\lift\algs4.jar
dummy .\resume.pdf
dummy .\Security\first\heapproblem1.docx
dummy .\Security\first\heapproblem2.docx
dummy .\Security\first\NSA_GHIDRA.docx
dummy .\yara64.exe
dummy .\yara64.exe
dummy .\Security\first\BufferOverflowAttacks.docx
dummy .\DSA Princeton\queues\lift\algs4.jar
error scanning .\Security\YARA.docx: could not open file
dummy .\Security\first\vulnerable_program2.docx
dummy .\Security\first\task1-gdb_x86.docx
dummy .\DSA Princeton\queues\tale.txt
dummy .\Security\first\shellcodeexploit.docx
dummy .\Security\first\StackOverflow.docx
dummy .\Security\first\shellprogram.docx
dummy .\Security\first\FINALdoc_Neha.docx

```

These are all the files over 500kb that it matched.

Rule 2:

```
rule csv_dde {  
    strings:  
        $a = "target"  
    condition:  
        $a  
}
```

Scans file to match the string "target".

Similar rule can be used to detect a CSV injection. Excel provides us with the functionality DDE (Dynamic Data Exchange), where we can execute application commands on the Excel window.

=cmd|' /C notepad '! 'A1'to open notepad

So we can scan csv files to match "=cmd" string.

```
C:\Users\windows\Desktop>yara64 -s -r dummy.yara .  
csv_dde .\Data Analytics\city_day.csv  
0x2e:$a: target  
csv_dde .\Data Analytics\FINAL.csv  
0x2e:$a: target  
csv_dde .\Data Analytics\forecast2.csv  
0x16:$a: target  
csv_dde .\Data Analytics\plot39.csv  
0xb:$a: target  
csv_dde .\dummy.yara  
0x23:$a: target  
csv_dde .\yara64.exe  
0x198b83:$a: target  
0x199660:$a: target  
csv_dde .\yara64.exe  
0x18fe93:$a: target  
0x190970:$a: target
```

Rule 3:

<https://seanthegeek.net/257/install-yara-write-yara-rules/amp/>

It would be very useful to check the attachments of suspected phishing emails reported to you by your users. PDF attachments with a link to a phishing site have become a common tactic, because many email gateways still do not check URLs in attached files. This rule checks for links in PDFs:

```
rule dummy {  
  
meta:  
  
    category = "informational"  
  
    confidence = "high"  
  
    description = "A PDFv1.7 that contains a link or external content"  
  
strings:  
  
    $pdf_magic = {25 50 44 46}  
  
    $s_anchor_tag = "<a " ascii wide nocase  
  
    $s_uri = /\(http.+\) / ascii wide nocase  
  
condition:  
  
    $pdf_magic at 0 and any of ($s*)  
  
}
```

The first part of the condition checks if the first few bytes of the file match the [magic numbers \(list here\)](#) for a PDF, allowing the rule to quickly disregard anything that isn't a PDF.

25 50 44 46 2d

%PDF-

Filters like these can greatly increase speed when scanning a large amount of data. The second part of the condition strings whose variable begins with \$s.

\$s_uri is a regular expression that matches any URI/URL in parenthesis, which will match the PDF standard for URI actions and URLs in forms.

\$s_anchor_tag matches any HTML anchor tag, which some PDF converters may leave in a document converted from HTML.

The `ascii`, `wide`, and `nocase` keywords tell YARA to search for ASCII and wide strings, and to be case-insensitive. By default, it will only search for ASCII strings, including substrings, and it will be case-sensitive. There are many more keywords for matching other kinds of strings.

```
C:\Users\windows\Desktop>yara64 -s -r dummy.yara C:\Users\windows\Desktop\resume.pdf
dummy.yara(14): warning in rule "dummy": $s_uri contains .* or .+, consider using .{,N} o
dummy C:\Users\windows\Desktop\resume.pdf
0x0:$pdf_magic: 25 50 44 46
0xce446:$s_uri: (https://zety.com/terms-of-service)
0xce501:$s_uri: (https://zety.com/privacy-policy)
0xce5ba:$s_uri: (https://zety.com/contact)
```