

# Detecting Malicious Files with YARA Rules as They Traverse the Network

David Bernal Michelena @d4v3c0d3r

Lead Security Researcher, SCILabs

August 2019, Mexico

Black Hat USA 2019



## Abstract

YARA, the pattern matching swiss knife for malware researchers, has been extremely useful at detecting suspicious files on the endpoint. However, few or no information is publicly available on how to leverage this useful tool to scan for files as they are traversing the network. In this paper, I will show how can open source Zeek IDS (formerly bro) and a custom developed script can be used to extract files from the network and to identify attacks on an early stage before it causes more damage. Scanning for YARA files on the network has the benefit of increased performance, as compared to scanning several gigabytes or terabytes on the endpoint, as well as target specific mime types, used for malware delivery. Additionally, Zeek IDS can provide additional context whenever a YARA rule is triggered, that will provide defenders with more information to act more rapidly.

## The Current Problem

Cyber attacks are becoming increasingly sophisticated. As defenses become better and stronger, the attackers adapt and improve their techniques to bypass defenses. In such scenario, defenders must enable as many defense mechanisms in place to increase their defense in depth posture. In this scenario having the capability to detect malicious files with YARA rules in the network is a must, with free and great open source tools such as Zeek and YARA, any organization can enable this detection capability, without the need to spend great amounts of money beforehand.

## Zeek

As explained in the official documentation: “Zeek (formerly bro) is a passive, open-source network traffic analyzer. It is primarily a security monitor that inspects all traffic on a link in depth for signs of suspicious activity. The most immediate benefit that a site gains from deploying Bro is an extensive set of *log files* that record a network’s activity in high-level terms. These logs include not only a comprehensive record of every connection seen on the wire, but also application-layer transcripts such as, e.g., all HTTP sessions with their requested URLs, key headers, MIME types, and server responses; DNS requests with replies; SSL certificates; key content of SMTP sessions; and much more. By default, Bro writes all this information into well-structured tab-separated log files suitable for post-processing with external software.”

## Zeek configuration

Zeek is a very well documented project, therefore the instructions that you will find here will not cover the basic installation, but if you are interested in this you can follow the instructions in the Zeek official documentation available here <https://docs.zeek.org/en/stable/install/install.html#>

## Enable file extraction in Zeek

Zeek by itself is extremely useful on its default configuration, but it can even be further customized for additional tasks. For instance, Zeek can very easily extract all the network files that it observes on the network by simply enabling the proper scripts in the main configuration file

```
vi /usr/local/bro/share/bro/broctl/main.bro
```

Uncomment the following line

```
@load frameworks/files/extract-all-files
```

```
# Load the script to support the "scripts" command.
@load misc/loaded-scripts

# All cluster nodes are inherently controllable with BroControl.
# TODO: This kind of sucks right now though because it always causes the
#       communications framework to hold open a port which can cause
#       high CPU usage on lightly loaded links due to the core packet
#       extraction loop.
@load frameworks/control/controllee
@load frameworks/files/extract-all-files
```

Redeploy bro configuration

```
Broctl deploy
```

```
[root@localhost bro]# broctl deploy
checking configurations ...
installing ...
removing old policies in /home/bro/spool/installed-scripts-do-not-touch/site ...
removing old policies in /home/bro/spool/installed-scripts-do-not-touch/auto ...
creating policy directories ...
installing site policies ...
generating standalone-layout.bro ...
generating local-networks.bro ...
generating broctl-config.bro ...
generating broctl-config.sh ...
stopping ...
stopping bro ...
starting ...
starting bro ...
```

This script extracts all the files to “extract\_files” folder, under bro folder, under spool. In my environment I have configured the spool on “/home/bro/spool/”.

Many of the files are from SSL, so we will only see the SSL certificate details in such file, not the actual content, so those are not very useful for our use case of scanning YARA rules.

```
[root@localhost extract_files]# strings extract-1564421018.754694-SSL-F3QM8B2B11APQLwvNd
Greater Manchesterl
Salfordl
COMODO CA Limitedl+0)
'COMODO RSA Certification Authority0
140212000000Z
290211235959Z0
Greater Manchesterl
Salfordl
COMODO CA Limitedl<0:
3COMODO RSA Organization Validation Secure Server CA0
.HeJ
B^~
kxB (
0lj 9
*HH*
E0C0A
http://crl.comodoca.com/COMODORSACertificationAuthority.crl0q
```

Each organization must decide if they prefer to extract all the files or only a subset of files, based on mime type. Extracting all files will give as much visibility as possible, but may not be very efficient, as many of the files would come from SSL or would be of file types that you don't care about. Also, the number of files to extract would be higher so that uses more resources on your sensor.

## Configuring targeted mime-type file extraction

If you want to extract specific mime types that are commonly useful for malware delivery, create the following script and name it “extract-some-files”.

The following script is provided as starting point, you can add or remove more mime types depending on the files that you are interested in. This configuration file includes some of the most commonly used mime extensions for malware delivery. To add more extensions, you will have to add it on the global ext map and on the if code, as you can see below on the script. Update the path where extracted files will be placed on “local fname”, almost at the bottom of the script. These positions have been highlighted in bold.

```
global ext_map: table[string] of string = {
  ["application/x-dosexec"] = "exe",
  ["text/plain"] = "txt",
  ["text/html"] = "html",
  ["application/zip"] = "zip",
  ["application/x-7z-compressed"] = "7z",
  ["application/x-rar"] = "rar",
  ["application/x-rar-compressed"] = "rar",
  ["application/xdmg"] = "dmg",
  ["application/msword"] = "doc",
  ["application/msexcel"] = "xls",
  ["application/mspowerpoint"] = "ppt",
  ["application/vnd.openxmlformats-officedocument.wordprocessingml.document"] = "docx",
  ["application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"] = "xlsx",
  ["application/vnd.openxmlformats-officedocument.presentationml.presentation"]
  = "pptx",
  ["application/pdf"] = "pdf",
  ["text/rtf"] = "rtf",
} &default = "";

event file_sniff(f: fa_file, meta: fa_metadata)
{
  if ( ! meta?$mime_type )
    return;

  if ( ! ( meta$mime_type == "application/x-dosexec" || meta$mime_type == "text/plain" ||
  meta$mime_type == "text/html" || meta$mime_type == "application/xdmg" || meta$mime_type
  == "application/zip" || meta$mime_type == "application/x-7z-compressed" || meta$mime_type
  == "application/x-rar" || meta$mime_type == "application/x-rar-compressed" ||
  meta$mime_type == "application/msword" || meta$mime_type == "application/msexcel" ||
  meta$mime_type == "application/mspowerpoint" || meta$mime_type ==
  "application/vnd.openxmlformats-officedocument.wordprocessingml.document"
  || meta$mime_type == "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"
  || meta$mime_type == "application/vnd.openxmlformats-
  officedocument.presentationml.presentation" || meta$mime_type == "text/rtf" ||
  meta$mime_type == "application/pdf"))
    return;

  local ext = "";

  if ( meta?$mime_type )
    ext = ext_map[meta$mime_type];

  local fname = fmt("/home/bro/extracted/%s-%s.%s", f$source, f$id, ext);
  Files::add_analyzer(f, Files::ANALYZER_EXTRACT, [$extract_filename=fname]);
}
```

Finally, update Zeek configuration, commenting out extract-all-files script and adding extract-some-files

```
# Load the script to support the "scripts" command.
@load misc/loaded-scripts

# All cluster nodes are inherently controllable with BroControl.
# TODO: This kind of sucks right now though because it always causes the
#       communications framework to hold open a port which can cause
#       high CPU usage on lightly loaded links due to the core packet
#       extraction loop.
@load frameworks/control/controllee
#@load frameworks/files/extract-all-files
@load frameworks/files/extract-some-files
```

```
[root@localhost files]# broctl deploy
checking configurations ...
installing ...
removing old policies in /home/bro/spool/installed-scripts-do-not-touch/site ...
removing old policies in /home/bro/spool/installed-scripts-do-not-touch/auto ...
creating policy directories ...
installing site policies ...
generating standalone-layout.bro ...
generating local-networks.bro ...
generating broctl-config.bro ...
generating broctl-config.sh ...
stopping ...
stopping bro ...
starting ...
starting bro ...
```

As soon as the Zeek configuration is deployed, the files will start to be extracted, it is important to monitor space usage on the sensor to delete old files as needed. The custom script takes care of that, as it deletes all the extracted files after scanning them and saving any files that triggered YARA rules.

# YARA

According to its official documentation: “YARA is an open-source tool designed to help malware researchers identify and classify malware samples. It makes it possible to create descriptions (or rules) for malware families based on textual and/or binary patterns. YARA is multi-platform, running on Linux, Windows and Mac OS X. YARA was created by Victor Álvarez while working in Virus Total. YARA stands for Yet Another Recursive Acronym. Each description, a.k.a rule, consists of a set of strings and a boolean expression which determines its logic.”

Let’s see the example below, taken from YARA official documentation.

```
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}
    $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
    $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
  condition:
    $a or $b or $c
}
```

According to Yara documentation, “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”.

In the following section you will see some YARA rules that I created and that will be used in the demonstration of this solution. YARA is freely available, for instructions on compiling and installing YARA on various platforms, please check the very complete documentation, available on <https://yara.readthedocs.io/en/v3.8.1/gettingstarted.html#compiling-and-installing-yara>

On Windows, it is also possible to download precompiled binaries from the official YARA repositories.

## More YARA rules

I created the following YARA rules to detect malicious Microsoft Word documents. The first one triggers with any Microsoft Word document with macro code and with strings related with code execution.

The second one targets files with macro code and any string that is related with AutoOpen functionality, that is code that is configured to be executed whenever the macro code is enabled, and Word Document is opened or closed.

```
rule Office_doc_Execution{
  meta:
    author = "David Bernal - Scilabs"
    description = "Detects Microsoft Office documents with strings related to
code execution"
    license = "https://creativecommons.org/licenses/by-nc/4.0/"
  strings:
    $run1 = ".Run"
    $run2 = ".ShellExecute"
    $macro1 = "ThisDocument"
    $macro2 = "Project"
  condition:
    uint32(0) == 0xe011cfd0 and uint32(4) == 0xe11ab1a1 and
    all of ($macro*) and 1 of ($run*)
}

rule Office_doc_AutoOpen {
  meta:
    author = "David Bernal - Scilabs"
    description = "Detects Microsoft Office documents with macro code, shell
and function names related to automatic code execution"
    license = "https://creativecommons.org/licenses/by-nc/4.0/"
    revision = "2"
  strings:
    $auto1 = "AutoOpen"
    $auto2 = "AutoClose"
    $auto3 = "Document_Open"
    $macro1 = "ThisDocument"
    $macro2 = "Project"
  condition:
    uint32(0) == 0xe011cfd0 and uint32(4) == 0xe11ab1a1 and
    all of ($macro*) and 1 of ($auto*)
}
```



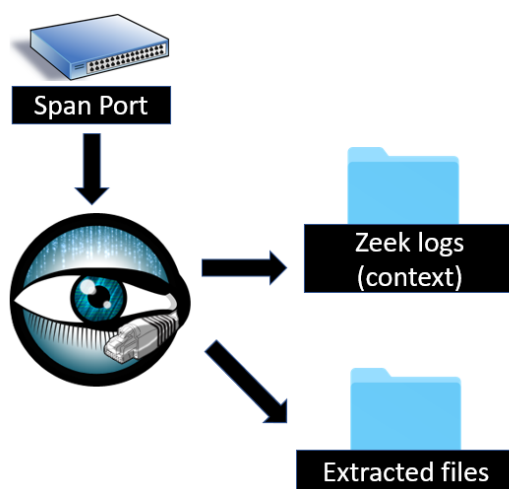
In addition to having and creating your own YARA rules for the threats that you identify, it is advisable to get YARA rules from trusted third parties. The following are some resources where additional YARA rules can be obtained:

- Florian Roth rule  
<https://github.com/Neo23x0/signature-base/tree/master/yara>
- YARA Rules group GNU-GPLv2  
<https://github.com/Yara-Rules/rules>
- Private research groups: YARA Exchange
- Public malware research papers by various security vendors and CERT teams, the following GitHub includes a long list of YARA repositories:  
<https://github.com/InQuest/awesome-yara>

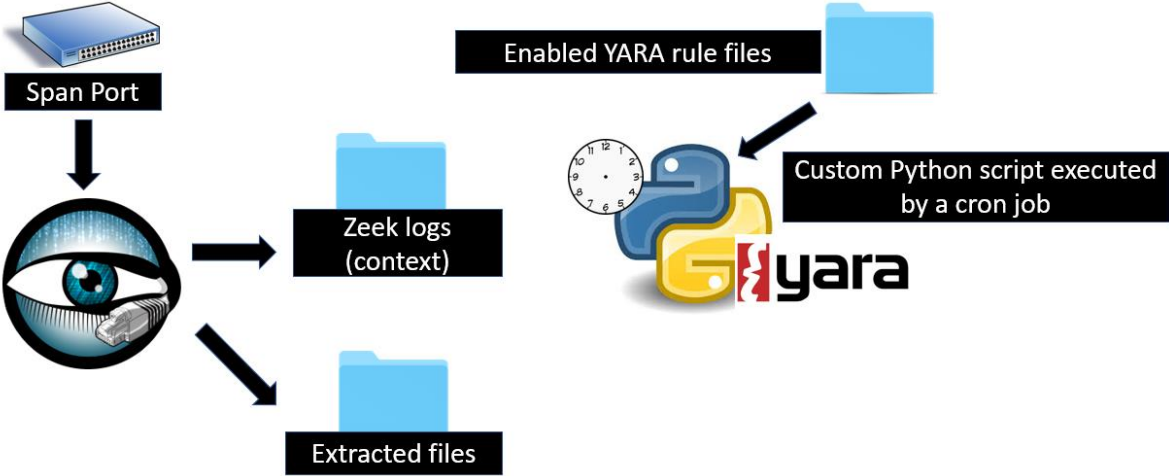
You must validate each YARA rule to determine its quality, license restrictions and decide if you want to use it in your environment. Once a selection has been done, place the YARA rules of your preference in the YARA rules folder. Finally, add the YARA rules path in the custom script that I developed.

## Integrating YARA with Zeek

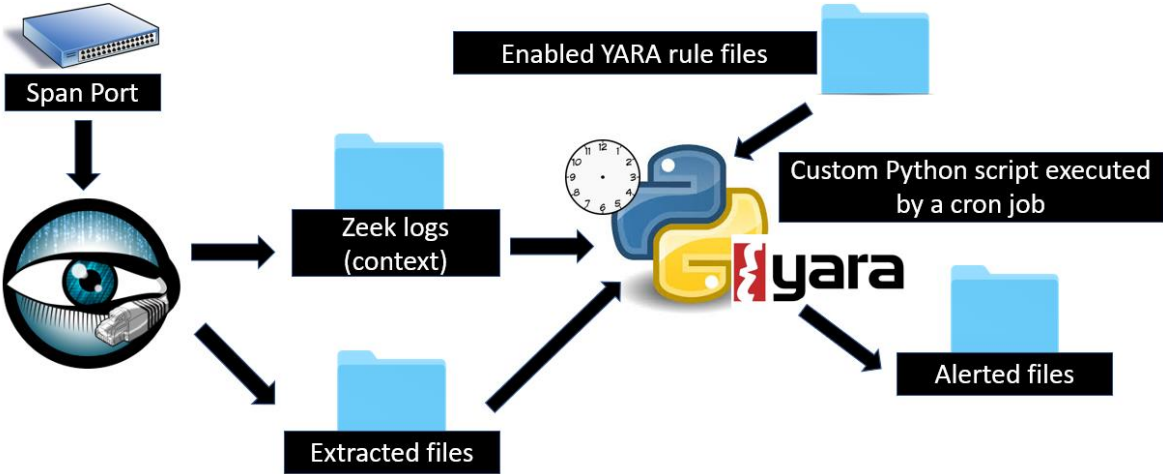
In this section I will explain the various components of the solution and how they interact with each other. The first component is the Zeek sensor, it will receive the traffic for inspection, will generate the network logs and will extract the files to a specific folder, as previously explained.



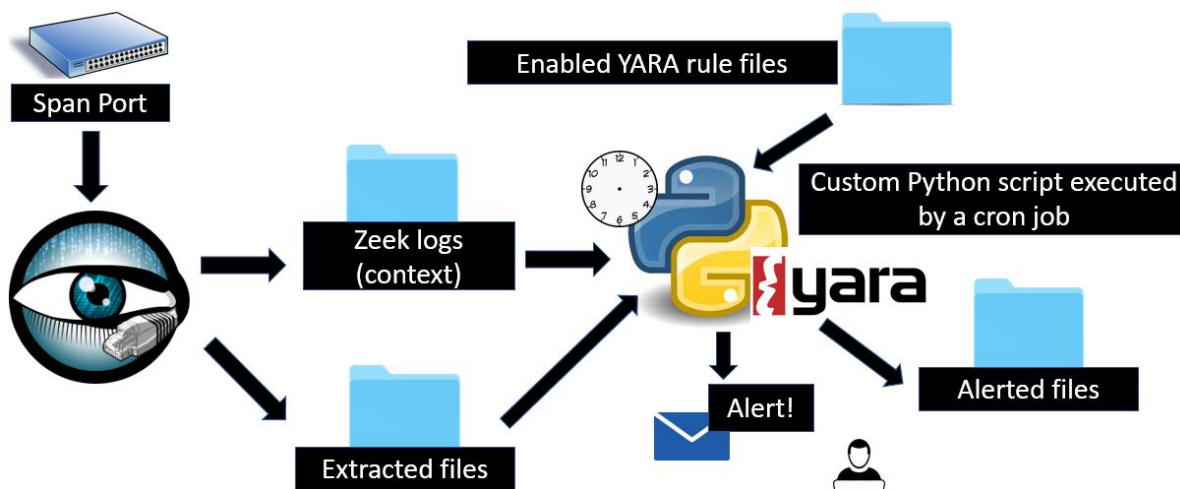
The second component is a cron job that will run a custom python script zeekYaraAlert.py that I developed, and that will take all the YARA rules enabled as input. It will basically concatenate all the YARA rules located in the YARA rule folder into a single file that will be used to scan all the files extracted by Zeek into the extracted files folder. There should not be duplicated YARA rules.



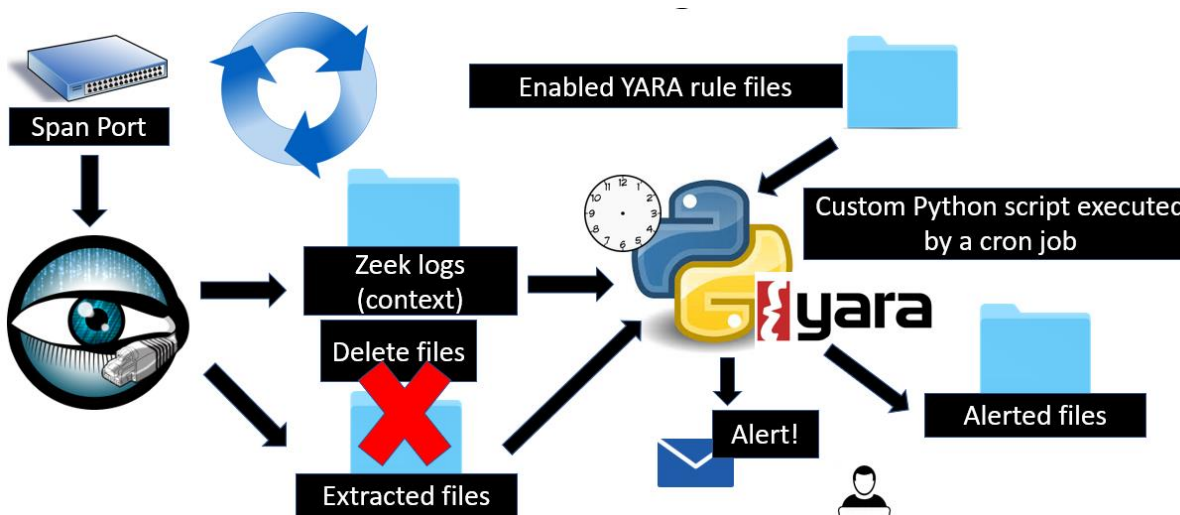
If there is any alerted file, it will take the file id and use it to search in the other Zeek log files to get more context, such as the involved IP address, application level details, like HTTP URL, user agent, HTTP method and it will copy the file that triggered any YARA rule to an Alerted files folder.



The script will send an email alert with the file sample that triggered the YARA rule compressed with the password "infected", in case that it is below the file limit set in the script, that by default is 10 MB.



Finally, the script will delete all the files on the extracted files folder and will run continuously, which is by default set to 1 minute in the script. Since few files are extracted per minute, the scan will be relatively fast, as compared to doing a full scan of a YARA rule on an endpoint hard drive.



## YaraZeekAlert script configuration

To run the script every minute, configure a cron job. If there is any issue when running the script, it's a good idea to have another script that deletes all the files on the extracted folder each hour, to prevent that the hard drive gets filled. Verify that only root has write permissions on both files.

```
*/1 * * * * /home/bro/YARA/yaraZeekAlert.py
```

```
30 * * * * /home/bro/YARA/deleteextracted.py
```

## Demo - Malicious Word Document with Macros

To demonstrate this solution, I will show how to detect an attack that uses a Microsoft Word document with malicious macro code. I decided to use this scenario because this attack vector has been very prevalent, with documented detections from at least 4 years ago and even older, but that is still used by various threat actors<sup>1</sup>. The CnC server shown torcido.cloudapp.net does not belong to a real threat actor, it is used only for proof of concept, such as testing the detection capabilities of this project and other security solutions.

The attack starts with a phishing email message that mentions that the recipient submitted an amount of money and includes a link to see the Invoice.

Hello,

**This email confirms that you submitted this total amount for processing:**

Deposit Date: June 16, 2019

Merchant Id: 4596

Amount: \$3,739.00 USD

BatchDepositID: 13467816589

**For more detailed information, please download the following invoice**

<http://torcido.cloudapp.net/payments/Invoice37481.doc>


Thanks for being our valued customer of EasyGadgets Inc.


Account Executive


---

<sup>1</sup> <https://blog.scilabs.mx/mexico-en-la-mira-cuatro-anos-de-catasia-parte-i/> 2014 Sample  
<https://isc.sans.edu/forums/diary/Malicious+spam+with+Word+document/20225/> 2015 Sample  
<https://blog.didierstevens.com/2017/04/20/malicious-documents-the-matryoshka-edition/> 2017 Sample  
[https://twitter.com/malware\\_traffic/status/1147305038253035522](https://twitter.com/malware_traffic/status/1147305038253035522) June 2019 Sample

When the user clicks on the link to download and see the attachment, it is transferred through HTTP, therefore Zeek extracts it and the custom scripts detects it as malicious as it triggers two YARA rules: Office\_doc\_AutoOpen and Office\_doc\_Execution. Additionally, this file also triggered a rule called "Office\_AutoOpen\_Macro" from Florian Roth researcher, which was developed independently. The fact that a rule from this trusted third party was triggered adds confidence to this detection.

 miércoles 17/07/2019 09:25 p. m.  
bro@scitum.com.mx  
YARA Alert

Para  David Eduardo Bernal Michelena

 HTTP-F5RH0B4Yv40fchollc.doc.zip  
17 KB

alerted rules: ['Office\_doc\_AutoOpen', 'Office\_doc\_Execution', 'Office\_AutoOpen\_Macro']  
filepath: /home/bro/extracted/HTTP-F5RH0B4Yv40fchollc.doc  
md5sum : e4b827195b5ee3ef85f3085852a26012  
sha1sum: bf7668ec23feeac9dad987e1338945e2bb8fa0fb  
sha256sum: 99b2b9973349d796a3658c37bda3d3cad2af46b793536bd3c54a036b44a416d4

The email alert also provides context about the detection, obtained from Zeek sensor, this includes the URL, domain name, source and destination IP addresses involved. This information helps the defender determine if is a false positive or a real threat, it also helps them determine the endpoint that just downloaded the malicious file and contains the attack with other security controls.

saved Zip file: /home/bro/YARA/alertedFiles/HTTP-F5RH0B4Yv40fchollc.doc.zip

```
context: 2019-07-17T21:24:28-0500 F5RH0B4Yv40fchollc 13.94.13.94 172.17.0.1
CUNdaY2zRjiVYQjLnh HTTP 0 MD5,EXTRACT,SHA1 application/msword
- 0.798037 F F 47616 47616 0 0 F -
e4b827195b5ee3ef85f3085852a26012 bf7668ec23feeac9dad987e1338945e2bb8fa0fb
- /home/bro/extracted/HTTP-F5RH0B4Yv40fchollc.doc F -
2019-07-17T21:24:27-0500 CUNdaY2zRjiVYQjLnh 172.17.0.1 36642 13.94.13.94
80 3 GET torcido.cloudapp.net /payments/Invoice37481.doc -
1.1 Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; WOW64; Trident/7.0;
.NET4.0C; .NET4.0E; ms-office; MSOffice 14) 0 47616 200 OK - -
(empty) - - - - - - F5RH0B4Yv40fchollc
- application/msword
```

If the alerted file is smaller than the threshold defined in the custom script, (which by default it is 10 MB) the email alert will include the file sample that triggered the alert, compressed with a custom password, that is by default "infected." This allows them to handle this file securely and maybe move it to an isolated environment or virtual machine where it can be analyzed to understand other aspects of the file sample. If the file is larger, then the analyst must log into the Zeek sensor to collect the file, this path is included on the email as can be seen in the image above.

## Challenges

One of the challenges when implementing this solution is that the YARA rules should have minimal false positives, as those would create noisy email alerts. It is required to invest some time analyzing third party rules to determine which ones to enable.

## Next Steps

- I will create more notification modes, such as CSV file generation, Elasticsearch reporting, SIEM reporting and automatically creating the event in Malware Information Sharing Platform (MISP). This would provide flexibility to the organization where it is implemented to have various ways of receiving YARA alerts.
- Sandbox integration, for automatic detonation of the alerted suspicious files.
- Currently many threats still use clear text protocols, but the use of encrypted protocols is increasing, therefore evaluating Zeek file extraction capabilities with solutions that decrypt network traffic is important to enhance its capabilities.
- Migration to Python 3.

## References

Zeek Project

<https://www.zeek.org/>

<https://docs.zeek.org>

YARA

<https://yara.readthedocs.io/en/v3.8.1/>

<https://support.virustotal.com/hc/en-us/articles/115002178945-YARA>