EFFECTIVELY USING SURICATA

LAB 5

LAB

1. Scenario

The organization you work for is considering deploying [Suricata](https://suricata-ids.org) to enhance its traffic inspection capabilities. The IT Security manager provided you with some Suricata rules to get familiar with their structure. He also provided you with PCAP files containing malicious traffic so that you can experiment with writing Suricata detection rules.

A test instance of Suricata has already been set up and is waiting for you!

2. Learning Objectives

The learning objective of this lab is not only to get familiar with the detection capabilities of Suricata but also to learn effective rule writing of Suricata rules.

Specifically, you will learn how to use Suricata's capabilities in order to:

- Have better visibility over a network
- Respond to incidents timely and effectively
- Proactively hunt for threats

<u>Don't get discouraged, if you are not familiar with regular expressions. Regular expressions will be covered as the course progresses.</u>

3. Introduction To Suricata Rules

What is a Suricata rule

Rules are essentially instructions to the Suricata engine to look for specific markers in network traffic that we want to be notified about if they are seen.

Rules are not always oriented towards detecting malicious traffic. Rules can also be written to provide blue team members with actionable and/or contextual information regarding activity that is occurring on a network.

Rules can be as specific or as broad as we want them to be. Finding the perfect balance is important in order, for example, to be able to detect variations of a given malware but also avoid false positives. They can be seen as a big AND statement (multiple contents are specified, and the rule will trigger an alert if and only if all of these contents are seen in the passing traffic).

Threat intelligence and infosec communities usually offer critical information based on which rules are developed.

Each active rule consumes some of the host's CPU and memory. Specific guidelines exist to assist effective Suricata rule writing.

```
action protocol from_ip port -> to_ip port (msg:"we are under attack
by X"; content:"something"; content:"something else"; sid:10000000;
rev:1;)
```

Header: This rule portion includes what action we want the rule to have along with the protocol Suricata should expect this rule to be found in. It also includes IPs and ports, as well as an arrow indicating directionality.

Rule message & Contents: The rule message is the message we want the analysts (or our self) to be presented with, whereas the contents are the portions of the traffic that we have deemed critical in order to detect activity that we want to be informed of.

Rule metadata: This rule portion mainly helps to keep track of rule modifications. *Sid* is a unique rule identifier.

Let's dive into the Header.

action tells the Suricata engine what to do should the contents are matched. It can be:

- Alert <- Generate alert and log matching packets, but let the traffic through
- Log <- Log traffic (no alert)
- Pass <- Ignore the packet and allow it through
- Drop <- If in IPS mode, drop the packet
- Reject <- IDS will send TCP RST packet(s)

protocol can be: tcp, udp, icmp, ip, http, tls, smb, dns

Then, we need a way to declare the **directionality of the traffic**; this can be done through:

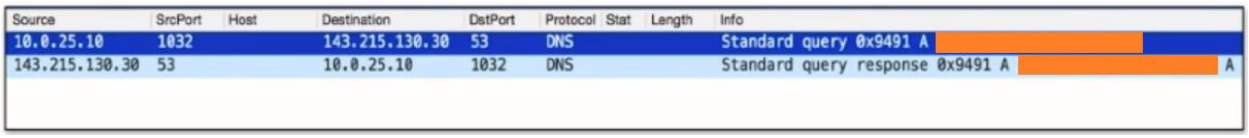
- **Rule Host Variables.** We have seen those variables when analyzing the *suricata.yaml* file. As a reminder, *$HOME_NET* refers to internal networks specified in the *suricata.yaml* file and *$EXTERNAL_NET* usually refers to whatever is not included in the *$HOME_NET* variable.

- **Rule Direction.** Between the 2 IP-Port pairs an arrow exists. This arrow tells the Suricata engine the direction in which the traffic is flowing. Examples:
  - Outbound traffic `$HOME_NET any -> $EXTERNAL_NET any`
  - Inbound traffic `$EXTENRAL_NET any -> $HOME_NET any`
  - Bidirectional traffic `$EXTENRAL_NET any <> $HOME_NET any`

Rule Ports declare the port(s) in which traffic for this rule will be evaluated. Examples:

- `alert tcp $HOME_NET any -> $EXTERNAL_NET 4444`
- `alert tcp $HOME_NET any -> $EXTERNAL_NET $FTP_PORTS`
  - port variables configurable in *suricata.yaml*
- `alert tcp $HOME_NET any -> $EXTERNAL_NET !80`
- `alert tcp $HOME_NET [1024:] -> $EXTERNAL_NET [80,800,6667:6680,!6672]`

Now, let's suppose you were required to create a header based on the PCAP below.



Source	SrcPort	Host	Destination	DstPort	Protocol	Stat	Length	Info
10.0.25.10	1032		143.215.130.30	53	DNS			Standard query 0x9491 A [redacted]
143.215.130.30	53		10.0.25.10	1032	DNS			Standard query response 0x9491 A [redacted] A

The rule header should be `alert dns $HOME_NET any -> any any`

`any` was used instead of `$EXTERNAL_NET` because some networks include internal DNS resolvers and you don't want to rule out that traffic.

Let's now dive into Rule message & Contents.

Rule Message. Arbitrary text that appears when the rule is triggered. For better understanding, it would nice if the rule messages you create contain malware architecture, malware family and malware action.

- **Flow**. Declares the originator and the responder. Remember while developing rules that you want to have the engine looking at "established" tcp sessions. Examples:
    - `flow:<established>,<to_server|to_client>;`
        - `from_server, from_client` can also be used
    - `alert tcp $HOME_NET any -> $EXTERNAL_NET 4444`
        - `flow:established,to_server;`
    - If the protocol is UDP
        - `flow:to_server;`
- **Dsize.** Allows matching using the size of the packet payload (not http, only tcp, and udp). It is based on TCP segment length, NOT total packet length (Wireshark filter: `tcp.len`). Examples:
    - dsize:312;
    - dsize:300<>400;

Rule Content. Values that identify a specific network traffic or activity. Suricata matches this unique content in packets for detection. Note that for certain characters, the use of hex is required. Examples:

- `content:"some thing";`
- `content:"some|20|thing";`
- `content:"User-Agent|3a 20|";`

- **Rule Buffers**. For some protocols, we have many buffers that we can use so that we don't search the entire packet for every content match. Using those buffers we can speed things up and also save resources. Refer to the following for more details: https://suricata.readthedocs.io/en/latest/rules/http-keywords.html

  Consider the below rule content.

  ```
  content:"POST"; content:"/trigger.php"; content:"DetoxCrypto";
  content:"publickey";
  ```

  By using buffers it will be transformed to the below.

  ```
  content:"POST"; http_method; content:"/trigger.php"; http_uri;
  content:"DetoxCrypto"; http_user_agent; content:"publickey";
  http_client_body;
  ```

- **Rule Options.** Additional modifiers to assist detection. They can help the Suricata engine in finding the exact location of contents.

  - **nocase.** Helps rules to not get bypassed through case changes. Example:
    - `content:"DetoxCrypto"; nocase; http_user_agent;`
  - **offset**. Informs the Suricata engine about the position inside the packet where is should start matching. This option is used in conjunction with "*depth*". Examples:
    - `content:"whatever"; offset:5;`
    - `content:"|00 03|"; offset:3; depth:2;`
      - Content of hex |00 03| will be found 3 bytes in and 2 bytes deep
  - **distance**. Informs the Suricata engine to look for the specified content `n` bytes relative to the previous match. This option is used in conjunction with "*within*". Examples:
    - `content:"whatever"; content:"something"; distance:5;`
    - `content:"whatever"; content:"something"; distance:0;`
    - `content:"|00 03|"; offset:3; depth:2; content:"whatever"; distance:0;  nocase; within:30;`

Here are some additional examples to better comprehend what we covered so far.

- `content:"whatever"; nocase; depth:9; content:"some|20|thing"; distance:0;`
    - We are looking for the string "whatever" (with `nocase` enabled) in the first 9 bytes of the packet. An offset of 0 is implied. Then, we are looking for the string "some thing" occurring anywhere after the first match of "whatever".
- `alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Malicious"; flow:established,to_server; dsize:45; content:"suspicious"; offset:33;`
    - We are looking for outbound TCP traffic on any port with a size of 45 bytes. The string "suspicious" should start 33 bytes in the packet. 33+12=45. Essentially, we are looking for the string at the last 12 bytes.

Finally, let's dive into Rule metadata.

- **reference**. This is the first rule metadata field we usually come across. It indicates where the initial information to develop this rule came from. Example:
    - `reference:url,securingtomorrow.mcafee.com/2017-11-20-dridex;`
- **sid.** This is the signature ID number. It is a unique number given by the rule writer. Example:
    - `sid:10000000;`
- **revision.** This field informs about the version of the rule. Example:
    - `rev:5;`

Note: To conclude covering Suricata rules, let's talk about PCRE (Pearl Compatible Regular Expression). PCRE is a very powerful ally while developing rules. One can use PCRE through the `pcre` statement (followed by a regular expression). Note that PCRE must be wrapped in leading and trailing forward slashes and any flags will go after the last slash. Examples:

- `pcre:"/something/flags";`
- `pcre:"/^\/[a-z0-9]{6}\.php$/Ui";`

- We are looking for 6 alphanumeric characters followed by *.php* and nothing after (the lowercase i indicates a case insensitive match, ^ indicates the start and $ indicates the end)

Note that anchors go after and before wrapped slashes and certain characters need to be escaped with a backslash. Additionally, **never write a PCRE-only rule**.

Managing Suricata Rules

- Suricata rules can be downloaded from:
  1. https://rules.emergingthreats.net/open/
  2. https://github.com/EmergingThreats/et-luajit-scripts
  3. https://github.com/ptresearch/AttackDetection
  4. Other Sources

- The best way to manage Suricata rules is by using: https://github.com/OISF/suricata-update (https://suricata.readthedocs.io/en/latest/rule-management/suricata-update.html)

- You can study more on Suricata rules on the official documentation site https://suricata.readthedocs.io/en/latest/rules/index.html

4. Recommended tools

- Suricata
- Wireshark

5. Network Configuration & Credentials

- Incident Responder's Subnet: **172.16.69.0/24**
- Suricata: **172.16.69.100**
- Connection Type: **SSH**
    - Use a Linux or Windows SSH client to connect to Suricata (172.16.69.100), as follows
        - Username: **elsanalyst**
        - Password: **@nalyst**

```
ssh elsanalyst@172.16.69.100  //For Linux-based machines
putty.exe, Session -> Host Name: 172.16.69.100, Port: 22, Connection Type: SSH
// For Windows-based machines
```

All provided PCAPs are inside the */home/elsuser/PCAPs* directory. They can be transferred to the machine from inside, which you connected to this lab as follows.

```
cd PCAPs
sudo python -m SimpleHTTPServer 80
```

Now, you can simply launch a browser of your choice inside the machine from inside which you connected to this lab and browse to http://172.16.69.100 to download the provided PCAP files.

6. Tasks

Task 1: Analyze the provided Suricata rules and describe what they look for

Armed with the knowledge you obtained from section 3. INTRODUCTION TO SURICATA RULES above, analyze the two Suricata rules below and describe what they look for.

Rule 1:

```
alert dns $HOME_NET any -> any any (msg:"TROJAN X Rogue DNS Query
Observed" dns_query; content:"searchcdn.gooogle.com";
isdataat:!1,relative; reference:url,threatintelprovider.com/trojanx;
classtype:trojan-activity; sid:1; rev:1;)
```

Rule 2:

```
alert tls $EXTERNAL_NET any -> $HOME_NET any (msg:"TROJAN Z malicious
SSL Cert"; flow:established,to_client; tls_cert_subject;
content:"CN=uniquestring"; classtype:trojan-activity; sid:1; rev:1;)
```

Task 2: Analyze the provided PCAP files and develop your own rules (Level: Beginner)

Now it's time to develop your own rules. Analyze the *Sofacy.pcap*, *Qadars.pcap*, *7ev3n.pcap* and *Malicious_document.pcap* PCAP files using Wireshark. Then, try to identify how you can instruct a Suricata sensor in order to detect the malicious traffic they contain and ultimately, develop a Suricata rule for each case.

Task 3: Analyze the provided PCAP files and develop your own rules (Level: Intermediate)

Things won't always be straightforward while developing Suricata rules. Complex cases will always come up. Analyze the *DDoSClient.pcap* and *Adobe.pcap* PCAP files using Wireshark. Then, try to identify how you can instruct a Suricata sensor in order to detect the malicious traffic they contain and ultimately, develop a Suricata rule for each case.

SOLUTIONS

Below, you can find solutions for every task of this lab. As a reminder, you can follow your own strategy, which may be different from the one explained in the following lab.

Task 1: Analyze the provided Suricata rules and describe what they look for

Let's start with **Rule 1**.

```
alert dns $HOME_NET any -> any any (msg:"TROJAN X Rogue DNS Query
Observed" dns_query; content:"default27061330-a.akamaihd.net";
isdataat:!1,relative; reference:url,threatintelprovider.com/trojanx;
classtype:trojan-activity; sid:1; rev:1;)
```

- This rule specifies the DNS protocol rather than UDP or something else. *dns* is a Suricata protocol keyword that allows the engine to detect DNS traffic based on the protocol and not the port. This protocol keyword (*dns*) should always be used when inspecting DNS traffic.

- The destination host is set to *any*. As mentioned previously, this is because some networks include internal DNS resolvers and we don't want to rule out that traffic.

- The destination port is set to *any* as well (instead of port 53); this is because we are using the *dns* protocol keyword which will help inspect DNS traffic regardless of the port used.

- Then, there is a typical message that will inform us about the detected activity.

- The main content of the rule starts with the *dns_query* keyword. *dns_query* is essentially a sticky buffer used to inspect the value of a DNS request. Being a sticky buffer results in all content following being included in the buffer. So, for this rule, we first declare the *dns_query* keyword and then the content, which contains the normalized domain that was included in the request. The *dns_query* buffer is normalized, so, one can use a literal period as opposed to the hexadecimal representation in the raw packet. Note that the DNS buffer doesn't include the null byte after the end of the domain. This is where the *isdataat* keyword comes into play. `!1,relative` means that there should be no data 1 byte relative to the previous match, which is effectively the domain name. Basically, this is a way to "lock" the match and avoid matching anything past the *.net* part.

- The remaining part can be easily comprehended.

Let’s continue with **Rule 2**.

```
alert tls $EXTERNAL_NET any -> $HOME_NET any (msg:"TROJAN Z malicious
SSL Cert"; flow:established,to_client; tls_cert_subject;
content:"CN=uniquestring"; classtype:trojan-activity; sid:1; rev:1;)
```

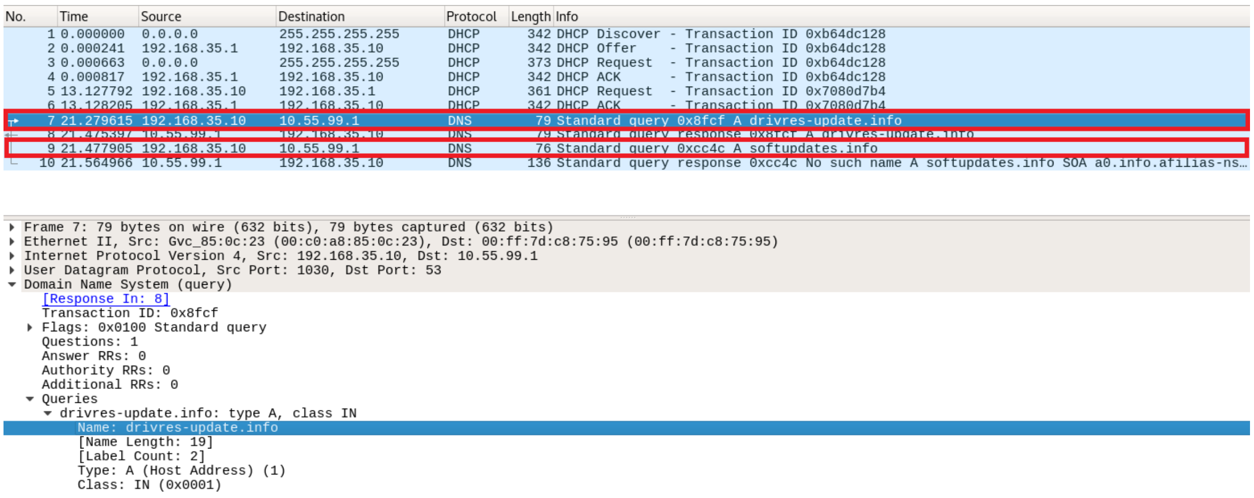
- This rule specifies the TLS protocol. *tls* is a Suricata protocol keyword that allows the engine to detect TLS traffic based on the protocol and not the port. By looking further down the rule, we are obviously checking for the delivery of a TLS certificate from an external server, hence the `$EXTERNAL_NET any -> $HOME_NET any` directionality. Once again, *any* is set for both ports since the *tls* keyword is utilized.
- Then, there is a typical message that will inform us about the detected activity. `flow:established,to_client;` *established* is used due to TCP and *to_client* due to the inbound traffic.
- *tls_cert_subject* is once again a buffer, and the rule content is a unique string included in the TLS certificate’s CN portion.
- The remaining part can be easily comprehended

TASK 2: ANALYZE THE PROVIDED PCAP FILES AND DEVELOP YOUR OWN RULES (LEVEL: BEGINNER)

Let's start with *Sofacy.pcap*

To learn more about Sofacy, refer to the following link https://securelist.com/sofacy-apt-hits-high-profile-targets-with-updated-toolset/72924/. We'll develop a Suricata rule based on our analysis of the *Sofacy.pcap* file, considering that what we see inside the PCAP is malicious.

By opening *Sofacy.pcap* in Wireshark, the first thing we notice is two (2) DNS queries and the respective DNS responses.



```
No.  Time         Source          Destination      Protocol Length Info
   1 0.000000     0.0.0.0         255.255.255.255  DHCP      342 DHCP Discover - Transaction ID 0xb64dc128
   2 0.000241     192.168.35.1    192.168.35.10    DHCP      342 DHCP Offer    - Transaction ID 0xb64dc128
   3 0.000663     0.0.0.0         255.255.255.255  DHCP      373 DHCP Request  - Transaction ID 0xb64dc128
   4 0.000817     192.168.35.1    192.168.35.10    DHCP      342 DHCP ACK      - Transaction ID 0xb64dc128
   5 13.127792    192.168.35.10   192.168.35.1     DHCP      361 DHCP Request  - Transaction ID 0x7080d7b4
   6 13.128205    192.168.35.1    192.168.35.10    DHCP      342 DHCP ACK      - Transaction ID 0x7080d7b4
   7 21.279615    192.168.35.10   10.55.99.1       DNS        79 Standard query 0x8fcf A drivres-update.info
   8 21.475397    10.55.99.1      192.168.35.10    DNS        79 Standard query response 0x8fcf A drivres-update.info
   9 21.477905    192.168.35.10   10.55.99.1       DNS        76 Standard query 0xcc4c A softupdates.info
  10 21.564966    10.55.99.1      192.168.35.10    DNS       136 Standard query response 0xcc4c No such name A softupdates.info SOA a0.info.afilias-ns…

▸ Frame 7: 79 bytes on wire (632 bits), 79 bytes captured (632 bits)
▸ Ethernet II, Src: Gvc_85:0c:23 (00:c0:a8:85:0c:23), Dst: 00:ff:7d:c8:75:95 (00:ff:7d:c8:75:95)
▸ Internet Protocol Version 4, Src: 192.168.35.10, Dst: 10.55.99.1
▸ User Datagram Protocol, Src Port: 1030, Dst Port: 53
▾ Domain Name System (query)
    [Response In: 8]
    Transaction ID: 0x8fcf
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ drivres-update.info: type A, class IN
        Name: drivres-update.info
        [Name Length: 19]
        [Label Count: 2]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
```

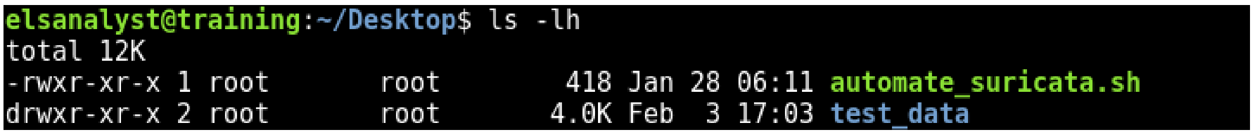
Let's create a Suricata rule to detect those two (2) C2 domains.

```
alert dns $HOME_NET any -> any any (msg:"TROJAN Activity Detected DNS Query
to Known Sofacy Domain 1"; dns_query; content:"drivres-update.info"; nocase;
isdataat:!1,relative; sid:1; rev:1;)


alert dns $HOME_NET any -> any any (msg:"TROJAN Activity Detected DNS Query
to Known Sofacy Domain 2"; dns_query; content:"softupdates.info"; nocase;
isdataat:!1,relative; sid:2; rev:1;)
```

Put these two (2) rules inside the *customsig.rules* directory residing in the home directory of the elsanalyst user. You can do so, by executing `sudo vim customsig.rules`

```
elsanalyst@training:~$ ls
customsig.rules  Desktop  Downloads  PCAPs
```

On the machine's Desktop, you will find a bash script (*automate_suricata.sh*), by OISF's J Williams, that will automate running Suricata with the *customsig.rules* file, logging in a different location (*/tmp/suricata*), echoing the contents of *fast.log* and also cleaning previous logs before each new Suricata execution occurs.

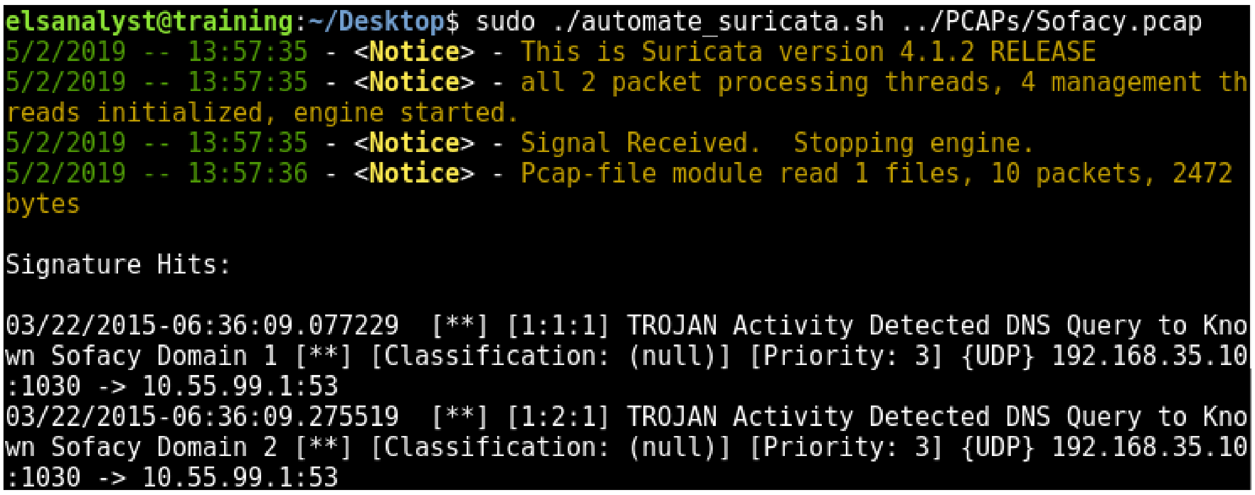


```
elsanalyst@training:~/Desktop$ ls -lh
total 12K
-rwxr-xr-x 1 root       root        418 Jan 28 06:11 automate_suricata.sh
drwxr-xr-x 2 root       root       4.0K Feb  3 17:03 test_data
```

It's time to test the rule above. You can do so, as follows.

```
cd Desktop

sudo ./automate_suricata.sh ../PCAPs/Sofacy.pcap
```

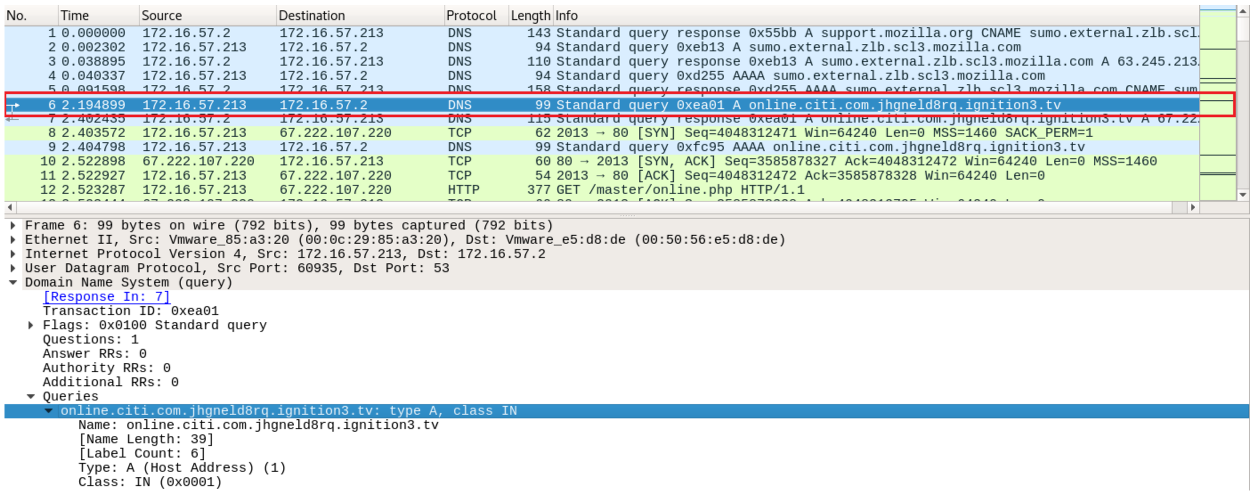
You should see the following.



```
elsanalyst@training:~/Desktop$ sudo ./automate_suricata.sh ../PCAPs/Sofacy.pcap
5/2/2019 -- 13:57:35 - <Notice> - This is Suricata version 4.1.2 RELEASE
5/2/2019 -- 13:57:35 - <Notice> - all 2 packet processing threads, 4 management th
reads initialized, engine started.
5/2/2019 -- 13:57:35 - <Notice> - Signal Received.  Stopping engine.
5/2/2019 -- 13:57:36 - <Notice> - Pcap-file module read 1 files, 10 packets, 2472
bytes

Signature Hits:

03/22/2015-06:36:09.077229  [**] [1:1:1] TROJAN Activity Detected DNS Query to Kno
wn Sofacy Domain 1 [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.35.10
:1030 -> 10.55.99.1:53
03/22/2015-06:36:09.275519  [**] [1:2:1] TROJAN Activity Detected DNS Query to Kno
wn Sofacy Domain 2 [**] [Classification: (null)] [Priority: 3] {UDP} 192.168.35.10
:1030 -> 10.55.99.1:53
```

Let's continue with the *Citi.pcap* file. If you are unfamiliar with Citi, Citi is a global bank.

By opening the *Citi.pcap* in Wireshark, the first thing we notice is a phishy-looking DNS query (note that online.citi.com is a legitimate URL).



```
No.  Time       Source          Destination     Protocol Length Info
   1 0.000000   172.16.57.2     172.16.57.213   DNS      143 Standard query response 0x55bb A support.mozilla.org CNAME sumo.external.zlb.scl
   2 0.002302   172.16.57.213   172.16.57.2     DNS       94 Standard query 0xeb13 A sumo.external.zlb.scl3.mozilla.com
   3 0.038895   172.16.57.2     172.16.57.213   DNS      110 Standard query response 0xeb13 A sumo.external.zlb.scl3.mozilla.com A 63.245.213.
   4 0.040337   172.16.57.213   172.16.57.2     DNS       94 Standard query 0xd255 AAAA sumo.external.zlb.scl3.mozilla.com
   5 0.091598   172.16.57.2     172.16.57.213   DNS      158 Standard query response 0xd255 AAAA sumo.external.zlb.scl3.mozilla.com CNAME sum
   6 2.194899   172.16.57.213   172.16.57.2     DNS       99 Standard query 0xea01 A online.citi.com.jhgneld8rq.ignition3.tv
   7 2.402435   172.16.57.2     172.16.57.213   DNS      115 Standard query response 0xea01 A online.citi.com.jhgneld8rq.ignition3.tv A 67.22.
   8 2.403572   172.16.57.213   67.222.107.220  TCP       62 2013 → 80 [SYN] Seq=4048312471 Win=64240 Len=0 MSS=1460 SACK_PERM=1
   9 2.404798   172.16.57.213   172.16.57.2     DNS       99 Standard query 0xfc95 AAAA online.citi.com.jhgneld8rq.ignition3.tv
  10 2.522898   67.222.107.220  172.16.57.213   TCP       60 80 → 2013 [SYN, ACK] Seq=3585878327 Ack=4048312472 Win=64240 Len=0 MSS=1460
  11 2.522927   172.16.57.213   67.222.107.220  TCP       54 2013 → 80 [ACK] Seq=4048312472 Ack=3585878328 Win=64240 Len=0
  12 2.523287   172.16.57.213   67.222.107.220  HTTP     377 GET /master/online.php HTTP/1.1

▸ Frame 6: 99 bytes on wire (792 bits), 99 bytes captured (792 bits)
▸ Ethernet II, Src: Vmware_85:a3:20 (00:0c:29:85:a3:20), Dst: Vmware_e5:d8:de (00:50:56:e5:d8:de)
▸ Internet Protocol Version 4, Src: 172.16.57.213, Dst: 172.16.57.2
▸ User Datagram Protocol, Src Port: 60935, Dst Port: 53
▾ Domain Name System (query)
    [Response In: 7]
    Transaction ID: 0xea01
  ▸ Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ online.citi.com.jhgneld8rq.ignition3.tv: type A, class IN
        Name: online.citi.com.jhgneld8rq.ignition3.tv
        [Name Length: 39]
        [Label Count: 6]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
```

Let's create a Suricata rule to detect such phishing-related DNS traffic against online.citi.com.

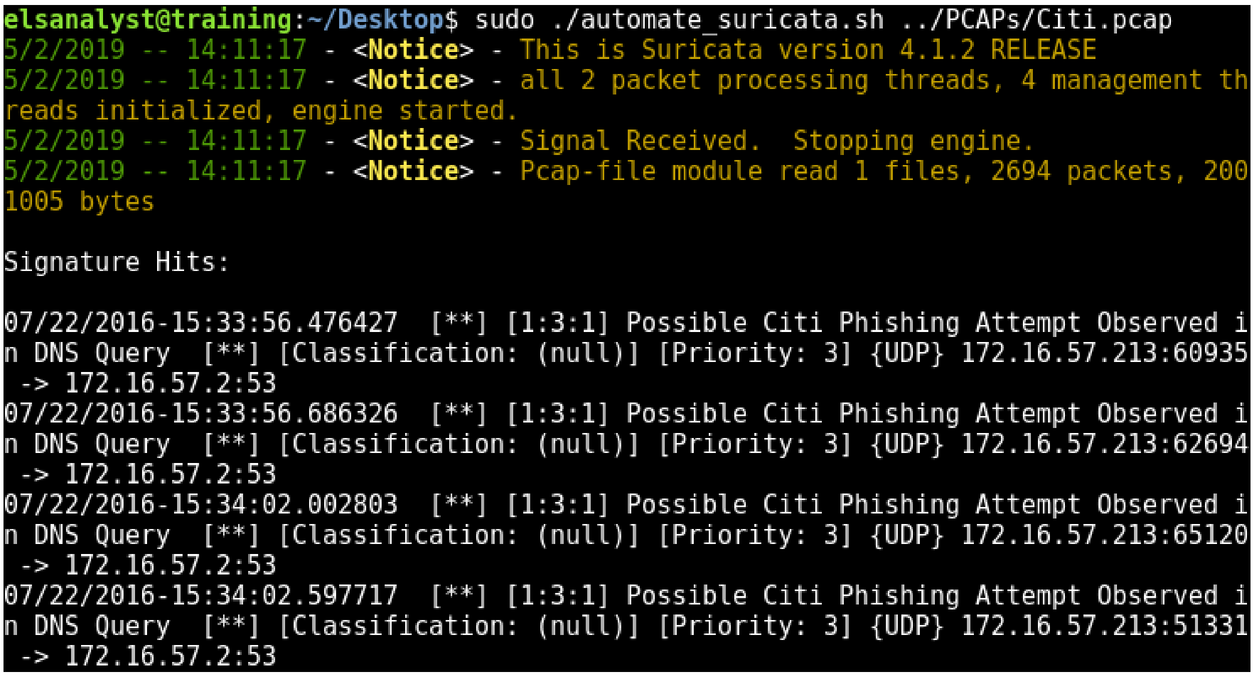
```
alert dns $HOME_NET any -> any any (msg:"Possible Citi Phishing
Attempt Observed in DNS Query "; dns_query; content:"online.citi.com";
nocase; isdataat:1,relative; sid:3; rev:1;)
```

`isdataat:1,relative` will inform us if any data exist after online.citi.com. There shouldn't be any data after it. If data are identified after online.citi.com, then, we are dealing with a phishing URL, similar to the one you can see in the PCAP file above (inside the red rectangle)

It's time to test the rule above. You can do so, as follows.

```
cd Desktop
sudo ./automate_suricata.sh ../PCAPs/Citi.pcap
```

You should see the following.



```
elsanalyst@training:~/Desktop$ sudo ./automate_suricata.sh ../PCAPs/Citi.pcap
5/2/2019 -- 14:11:17 - <Notice> - This is Suricata version 4.1.2 RELEASE
5/2/2019 -- 14:11:17 - <Notice> - all 2 packet processing threads, 4 management th
reads initialized, engine started.
5/2/2019 -- 14:11:17 - <Notice> - Signal Received.  Stopping engine.
5/2/2019 -- 14:11:17 - <Notice> - Pcap-file module read 1 files, 2694 packets, 200
1005 bytes

Signature Hits:

07/22/2016-15:33:56.476427  [**] [1:3:1] Possible Citi Phishing Attempt Observed i
n DNS Query  [**] [Classification: (null)] [Priority: 3] {UDP} 172.16.57.213:60935
 -> 172.16.57.2:53
07/22/2016-15:33:56.686326  [**] [1:3:1] Possible Citi Phishing Attempt Observed i
n DNS Query  [**] [Classification: (null)] [Priority: 3] {UDP} 172.16.57.213:62694
 -> 172.16.57.2:53
07/22/2016-15:34:02.002803  [**] [1:3:1] Possible Citi Phishing Attempt Observed i
n DNS Query  [**] [Classification: (null)] [Priority: 3] {UDP} 172.16.57.213:65120
 -> 172.16.57.2:53
07/22/2016-15:34:02.597717  [**] [1:3:1] Possible Citi Phishing Attempt Observed i
n DNS Query  [**] [Classification: (null)] [Priority: 3] {UDP} 172.16.57.213:51331
 -> 172.16.57.2:53
```

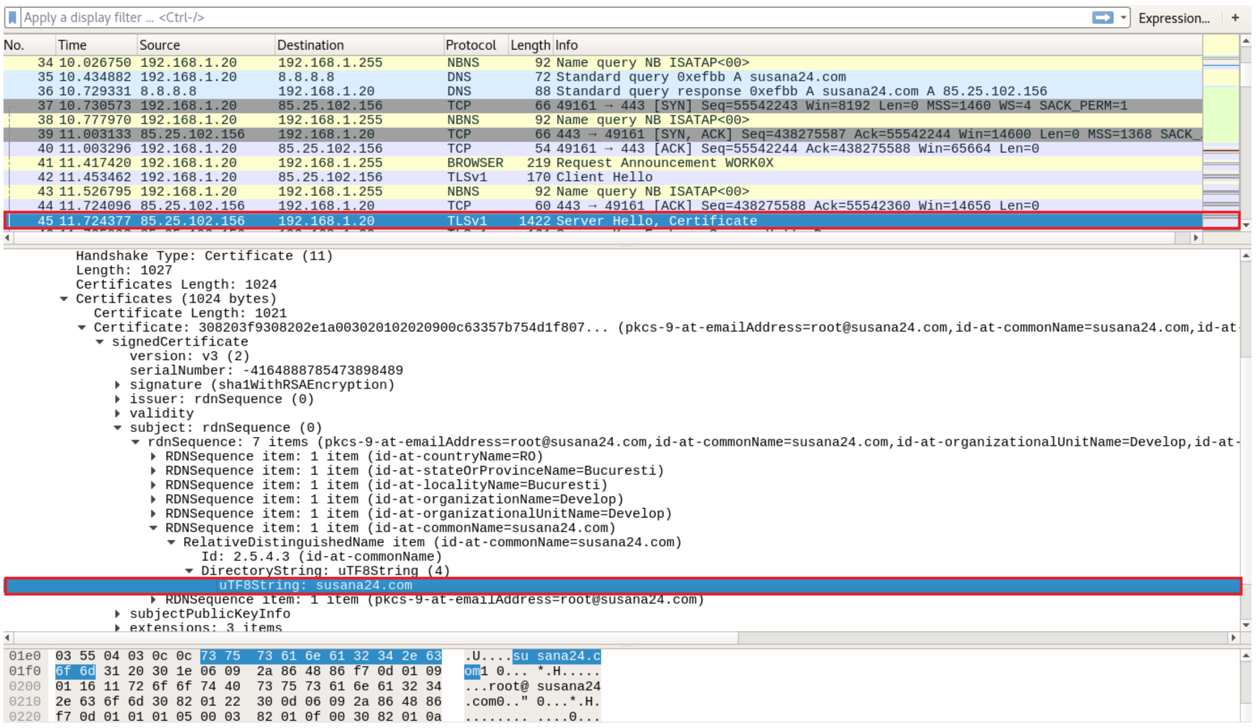
Now, it's time to analyze the *Qadars.pcap* file.

To learn more about Qadars, refer to the following links:

- https://www.countercept.com/blog/decrypting-qadars-banking-trojan-c2-traffic/
- https://sslbl.abuse.ch/ssl-certificates/sha1/1862c777babf298fe5a93406e4dc8456d718abcf/

We'll develop a Suricata rule based on our analysis of the *Qadars.pcap* file, considering that what we see inside the PCAP is malicious.

By opening *Qadars.pcap* in Wireshark, the first thing we notice is a DNS query and a DNS response regarding susana24.com. Then, we notice TLS traffic initiating.



```
Apply a display filter ... <Ctrl-/>                                          Expression...  +
No.  Time      Source         Destination    Protocol Length Info
  34 10.026750 192.168.1.20   192.168.1.255  NBNS       92 Name query NB ISATAP<00>
  35 10.434882 192.168.1.20   8.8.8.8        DNS        72 Standard query 0xefbb A susana24.com
  36 10.729331 8.8.8.8        192.168.1.20   DNS        88 Standard query response 0xefbb A susana24.com A 85.25.102.156
  37 10.730573 192.168.1.20   85.25.102.156  TCP        66 49161 → 443 [SYN] Seq=55542243 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
  38 10.777970 192.168.1.20   192.168.1.255  NBNS       92 Name query NB ISATAP<00>
  39 11.003133 85.25.102.156  192.168.1.20   TCP        66 443 → 49161 [SYN, ACK] Seq=438275587 Ack=55542244 Win=14600 Len=0 MSS=1368 SACK_
  40 11.003296 192.168.1.20   85.25.102.156  TCP        54 49161 → 443 [ACK] Seq=55542244 Ack=438275588 Win=65664 Len=0
  41 11.417420 192.168.1.20   192.168.1.255  BROWSER   219 Request Announcement WORK0X
  42 11.453462 192.168.1.20   85.25.102.156  TLSv1     170 Client Hello
  43 11.526795 192.168.1.20   192.168.1.255  NBNS       92 Name query NB ISATAP<00>
  44 11.724096 85.25.102.156  192.168.1.20   TCP        60 443 → 49161 [ACK] Seq=438275588 Ack=55542360 Win=14656 Len=0
  45 11.724377 85.25.102.156  192.168.1.20   TLSv1    1422 Server Hello, Certificate

        Handshake Type: Certificate (11)
        Length: 1027
        Certificates Length: 1024
      ▾ Certificates (1024 bytes)
          Certificate Length: 1021
        ▾ Certificate: 308203f9308202e1a003020102020900c63357b754d1f807... (pkcs-9-at-emailAddress=root@susana24.com,id-at-commonName=susana24.com,id-at
          ▾ signedCertificate
              version: v3 (2)
              serialNumber: -4164888785473898489
            ▸ signature (sha1WithRSAEncryption)
            ▸ issuer: rdnSequence (0)
            ▸ validity
            ▾ subject: rdnSequence (0)
              ▾ rdnSequence: 7 items (pkcs-9-at-emailAddress=root@susana24.com,id-at-commonName=susana24.com,id-at-organizationalUnitName=Develop,id-at-
                ▸ RDNSequence item: 1 item (id-at-countryName=RO)
                ▸ RDNSequence item: 1 item (id-at-stateOrProvinceName=Bucuresti)
                ▸ RDNSequence item: 1 item (id-at-localityName=Bucuresti)
                ▸ RDNSequence item: 1 item (id-at-organizationName=Develop)
                ▸ RDNSequence item: 1 item (id-at-organizationalUnitName=Develop)
                ▾ RDNSequence item: 1 item (id-at-commonName=susana24.com)
                  ▾ RelativeDistinguishedName item (id-at-commonName=susana24.com)
                      Id: 2.5.4.3 (id-at-commonName)
                    ▾ DirectoryString: uTF8String (4)
                        uTF8String: susana24.com
                ▸ RDNSequence item: 1 item (pkcs-9-at-emailAddress=root@susana24.com)
            ▸ subjectPublicKeyInfo
            ▸ extensions: 3 items

01e0  03 55 04 03 0c 0c 73 75  73 61 6e 61 32 34 2e 63   .U....su sana24.c
01f0  6f 6d 31 20 30 1e 06 09  2a 86 48 86 f7 0d 01 09   om1 0... *.H.....
0200  01 16 11 72 6f 6f 74 40  73 75 73 61 6e 61 32 34   ...root@ susana24
0210  2e 63 6f 6d 30 82 01 22  30 0d 06 09 2a 86 48 86   .com0.." 0...*.H.
0220  f7 0d 01 01 01 05 00 03  82 01 0f 00 30 82 01 0a   ........ ....0...
```

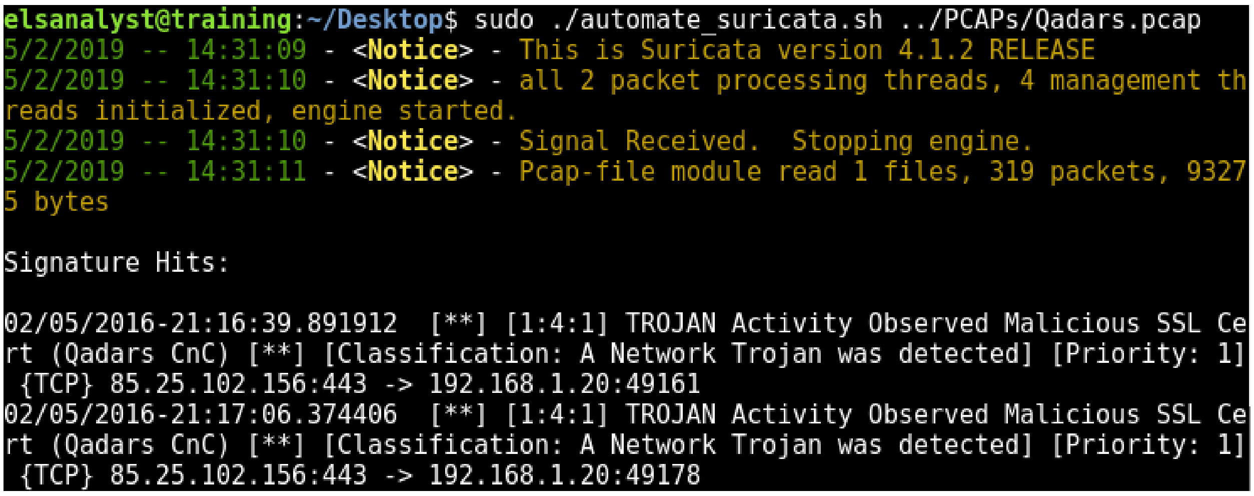
Let's focus on the TLS certificate this time and try to create a rule based on the CN portion of it (susana24.com).

```
alert tls $EXTERNAL_NET any -> $HOME_NET any (msg:"TROJAN Activity
Observed Malicious SSL Cert (Qadars CnC)"; flow:established,to_client;
tls_cert_subject; content:"CN=susana24.com"; classtype:trojan-
activity; sid:4; rev:1;)
```

It's time to test the rule above. You can do so, as follows.

```
cd Desktop
sudo ./automate_suricata.sh ../PCAPs/Qadars.pcap
```

You should see the following.



```
elsanalyst@training:~/Desktop$ sudo ./automate_suricata.sh ../PCAPs/Qadars.pcap
5/2/2019 -- 14:31:09 - <Notice> - This is Suricata version 4.1.2 RELEASE
5/2/2019 -- 14:31:10 - <Notice> - all 2 packet processing threads, 4 management th
reads initialized, engine started.
5/2/2019 -- 14:31:10 - <Notice> - Signal Received.  Stopping engine.
5/2/2019 -- 14:31:11 - <Notice> - Pcap-file module read 1 files, 319 packets, 9327
5 bytes

Signature Hits:

02/05/2016-21:16:39.891912  [**] [1:4:1] TROJAN Activity Observed Malicious SSL Ce
rt (Qadars CnC) [**] [Classification: A Network Trojan was detected] [Priority: 1]
 {TCP} 85.25.102.156:443 -> 192.168.1.20:49161
02/05/2016-21:17:06.374406  [**] [1:4:1] TROJAN Activity Observed Malicious SSL Ce
rt (Qadars CnC) [**] [Classification: A Network Trojan was detected] [Priority: 1]
 {TCP} 85.25.102.156:443 -> 192.168.1.20:49178
```

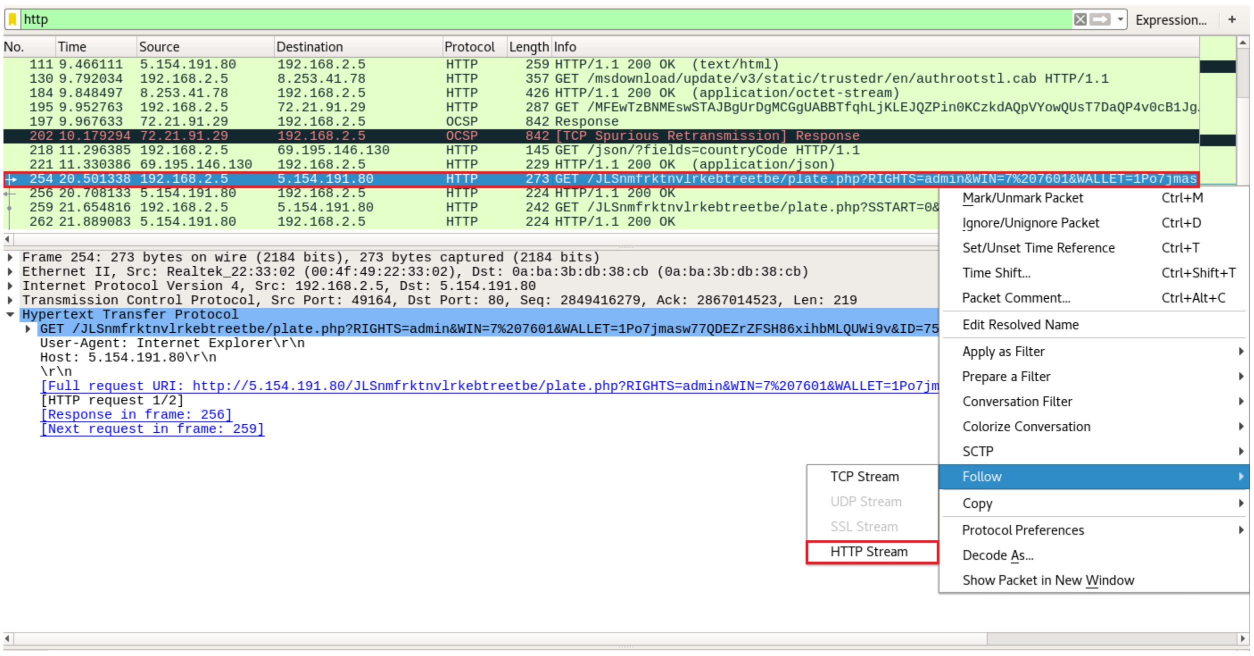
Now, it's time to analyze the *7ev3n.pcap*

To learn more about the 7ev3n ransomware, refer to the following link https://www.vmray.com/cyber-security-blog/7ev3n-honet-ransomware-rest-us/.

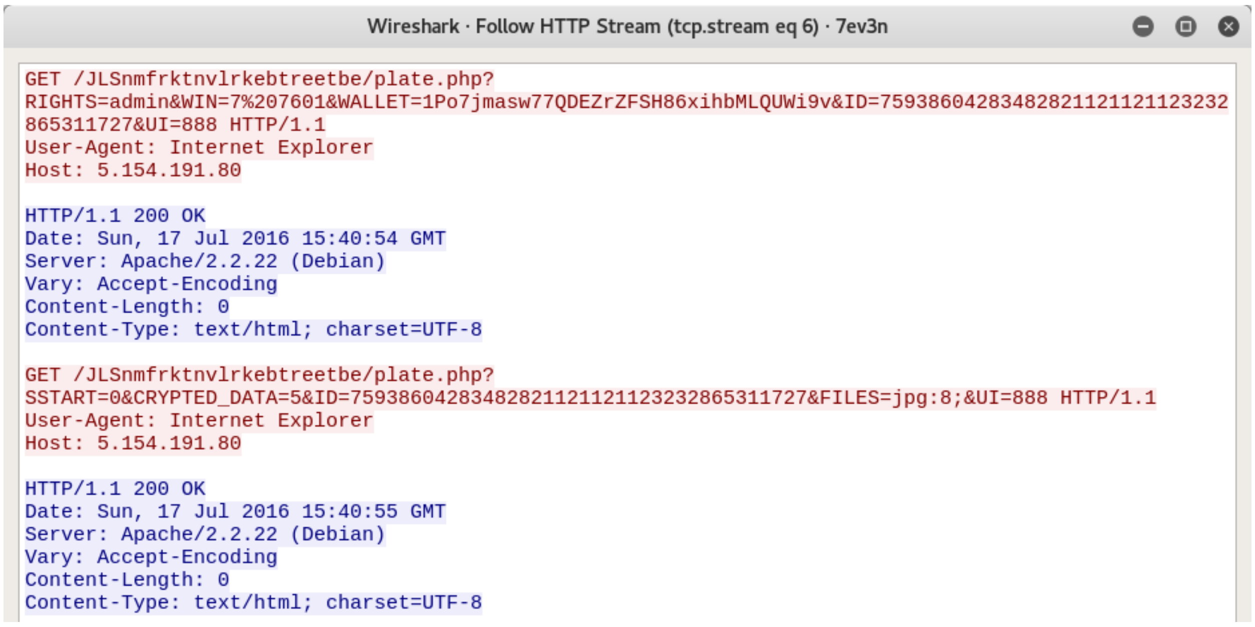
We'll develop a Suricata rule based on our analysis of the *7ev3n.pcap* file, considering that what we see inside the PCAP is malicious.

By opening *7ev3n.pcap* in Wireshark and filtering so that we can see HTTP traffic only, the first thing we notice is this curious-looking request.

Let's follow the whole stream.



```
No.  Time         Source          Destination     Protocol  Length Info
 111 9.466111     5.154.191.80    192.168.2.5     HTTP      259 HTTP/1.1 200 OK  (text/html)
 130 9.792034     192.168.2.5     8.253.41.78     HTTP      357 GET /msdownload/update/v3/static/trustedr/en/authrootstl.cab HTTP/1.1
 184 9.848497     8.253.41.78     192.168.2.5     HTTP      426 HTTP/1.1 200 OK  (application/octet-stream)
 195 9.952763     192.168.2.5     72.21.91.29     HTTP      287 GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBTfqhLjKLEJQZPin0KCzkdAQpVYowQUsT7DaQP4v0cB1Jg
 197 9.967633     72.21.91.29     192.168.2.5     OCSP      842 Response
 202 10.179294    72.21.91.29     192.168.2.5     OCSP      842 [TCP Spurious Retransmission] Response
 218 11.296385    192.168.2.5     69.195.146.130  HTTP      145 GET /json/?fields=countryCode HTTP/1.1
 221 11.330386    69.195.146.130  192.168.2.5     HTTP      229 HTTP/1.1 200 OK  (application/json)
 254 20.501338    192.168.2.5     5.154.191.80    HTTP      273 GET /JLSnmfrktnvlrkebtreetbe/plate.php?RIGHTS=admin&WIN=7%207601&WALLET=1Po7jmas
 256 20.708133    5.154.191.80    192.168.2.5     HTTP      224 HTTP/1.1 200 OK
 259 21.654816    192.168.2.5     5.154.191.80    HTTP      242 GET /JLSnmfrktnvlrkebtreetbe/plate.php?SSTART=0&
 262 21.889083    5.154.191.80    192.168.2.5     HTTP      224 HTTP/1.1 200 OK

Frame 254: 273 bytes on wire (2184 bits), 273 bytes captured (2184 bits)
Ethernet II, Src: Realtek_22:33:02 (00:4f:49:22:33:02), Dst: 0a:ba:3b:db:38:cb (0a:ba:3b:db:38:cb)
Internet Protocol Version 4, Src: 192.168.2.5, Dst: 5.154.191.80
Transmission Control Protocol, Src Port: 49164, Dst Port: 80, Seq: 2849416279, Ack: 2867014523, Len: 219
Hypertext Transfer Protocol
  GET /JLSnmfrktnvlrkebtreetbe/plate.php?RIGHTS=admin&WIN=7%207601&WALLET=1Po7jmasw77QDEZrZFSH86xihbMLQUWi9v&ID=75
  User-Agent: Internet Explorer\r\n
  Host: 5.154.191.80\r\n
  \r\n
  [Full request URI: http://5.154.191.80/JLSnmfrktnvlrkebtreetbe/plate.php?RIGHTS=admin&WIN=7%207601&WALLET=1Po7jm
  [HTTP request 1/2]
  [Response in frame: 256]
  [Next request in frame: 259]
```



```
GET /JLSnmfrktnvlrkebtreetbe/plate.php?
RIGHTS=admin&WIN=7%207601&WALLET=1Po7jmasw77QDEZrZFSH86xihbMLQUWi9v&ID=759386042834828211211211232328
65311727&UI=888 HTTP/1.1
User-Agent: Internet Explorer
Host: 5.154.191.80

HTTP/1.1 200 OK
Date: Sun, 17 Jul 2016 15:40:54 GMT
Server: Apache/2.2.22 (Debian)
Vary: Accept-Encoding
Content-Length: 0
Content-Type: text/html; charset=UTF-8

GET /JLSnmfrktnvlrkebtreetbe/plate.php?
SSTART=0&CRYPTED_DATA=5&ID=75938604283482821121121123232865311727&FILES=jpg:8;&UI=888 HTTP/1.1
User-Agent: Internet Explorer
Host: 5.154.191.80

HTTP/1.1 200 OK
Date: Sun, 17 Jul 2016 15:40:55 GMT
Server: Apache/2.2.22 (Debian)
Vary: Accept-Encoding
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

The requests above can provide us with a lot of clues on how to develop an effective Suricata rule.

Viable rules can be found below.

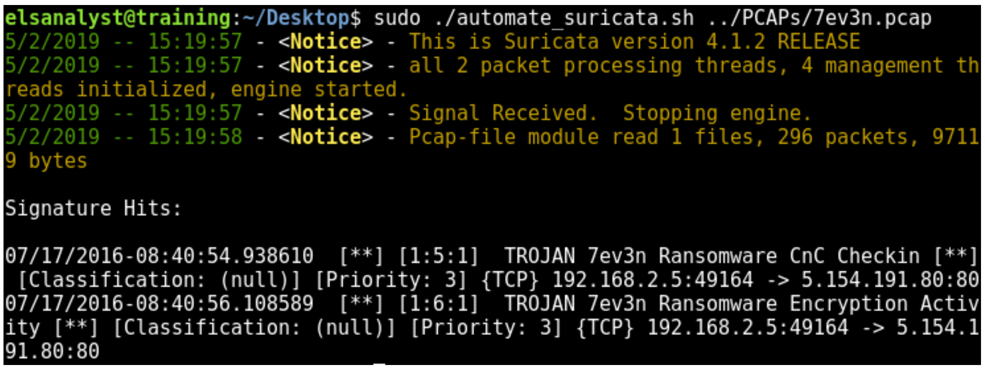
```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:" TROJAN 7ev3n
Ransomware CnC Checkin"; flow:established,to_server; content:"GET";
http_method; content:".php?RIGHTS="; http_uri; content:"&WIN=";
http_uri; distance:0; content:"&WALLET="; http_uri; distance:0;
content:"&ID="; http_uri; distance:0; content:"&UI="; http_uri;
distance:0; content:"Internet|20|Explorer"; http_user_agent; depth:17;
isdataat:!1,relative; http_header_names; content:!"Referer";
content:!"Accept"; sid:5; rev:1;)


alert http $HOME_NET any -> $EXTERNAL_NET any (msg:" TROJAN 7ev3n
Ransomware Encryption Activity"; flow:established,to_server;
content:"GET"; http_method; content:".php?SSTART="; http_uri;
content:"&CRYPTED_DATA="; http_uri; distance:0; content:"&ID=";
http_uri; distance:0; content:"&FILES="; http_uri; distance:0;
content:"&UI="; http_uri; distance:0; content:"Internet|20|Explorer";
http_user_agent; depth:17; isdataat:!1,relative; http_header_names;
content:!"Referer"; content:!"Accept"; sid:6; rev:1;)
```

`depth:17; isdataat:!1,relative;` is looking to see if there are any data after the last "r" of the "Internet Explorer" string, ensuring that the User Agent field only contains "Internet Explorer". `http_header_names; content:!"Referer"; content:!"Accept";` is leveraging the lack of usual headers for detection purposes.

It's time to test the rule above. You can do so, as follows.

```
cd Desktop

sudo ./automate_suricata.sh ../PCAPs/7ev3n.pcap
```

You should see the following.

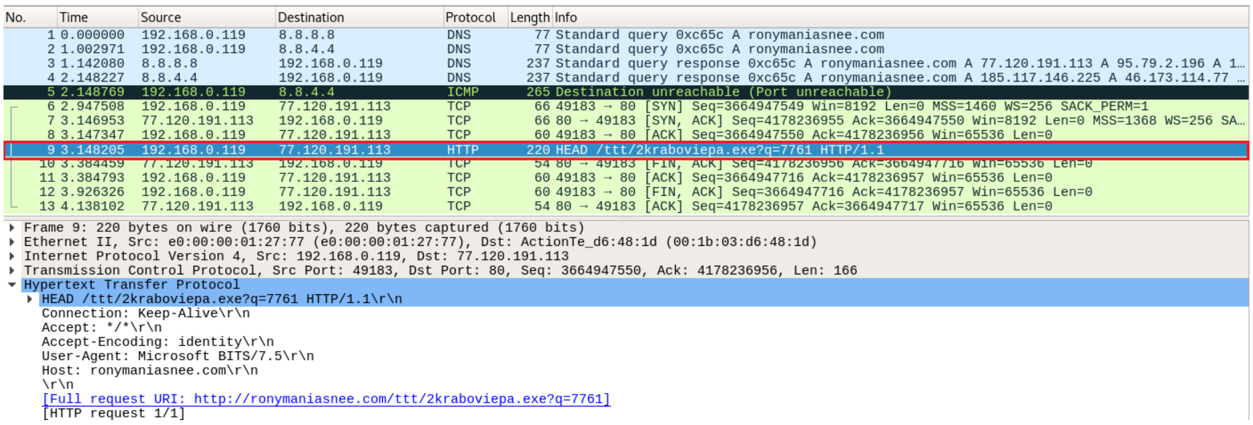


```
elsanalyst@training:~/Desktop$ sudo ./automate_suricata.sh ../PCAPs/7ev3n.pcap
5/2/2019 -- 15:19:57 - <Notice> - This is Suricata version 4.1.2 RELEASE
5/2/2019 -- 15:19:57 - <Notice> - all 2 packet processing threads, 4 management th
reads initialized, engine started.
5/2/2019 -- 15:19:57 - <Notice> - Signal Received.  Stopping engine.
5/2/2019 -- 15:19:58 - <Notice> - Pcap-file module read 1 files, 296 packets, 9711
9 bytes

Signature Hits:

07/17/2016-08:40:54.938610  [**] [1:5:1]  TROJAN 7ev3n Ransomware CnC Checkin [**]
 [Classification: (null)] [Priority: 3] {TCP} 192.168.2.5:49164 -> 5.154.191.80:80
07/17/2016-08:40:56.108589  [**] [1:6:1]  TROJAN 7ev3n Ransomware Encryption Activ
ity [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.2.5:49164 -> 5.154.1
91.80:80
```

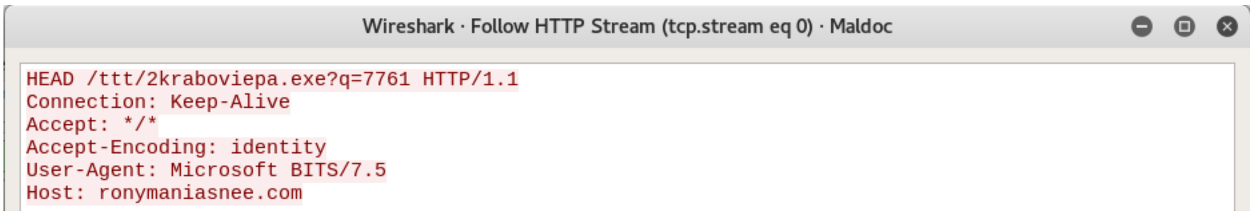
Finally, let's analyze the *Malicious_document.pcap*

Typically, malicious Office documents rely on macro execution, embedded objects, or exploits to deliver a payload onto the victim machine. In such cases, the URI can be so characteristic as to be used as solid rule content. The same applies for the User-Agent.

By opening the *Malicious_document.pcap* in Wireshark, the first thing we notice is this curious-looking HTTP request.



```
No.  Time      Source          Destination     Protocol Length Info
   1 0.000000  192.168.0.119   8.8.8.8         DNS        77 Standard query 0xc65c A ronymaniasnee.com
   2 1.002971  192.168.0.119   8.8.4.4         DNS        77 Standard query 0xc65c A ronymaniasnee.com
   3 1.142080  8.8.8.8         192.168.0.119   DNS       237 Standard query response 0xc65c A ronymaniasnee.com A 77.120.191.113 A 95.79.2.196 A 1…
   4 2.148227  8.8.4.4         192.168.0.119   DNS       237 Standard query response 0xc65c A ronymaniasnee.com A 185.117.146.225 A 46.173.114.77 …
   5 2.148769  192.168.0.119   8.8.4.4         ICMP      265 Destination unreachable (Port unreachable)
   6 2.947508  192.168.0.119   77.120.191.113  TCP        66 49183 → 80 [SYN] Seq=3664947549 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
   7 3.146953  77.120.191.113  192.168.0.119   TCP        66 80 → 49183 [SYN, ACK] Seq=4178236955 Ack=3664947550 Win=8192 Len=0 MSS=1368 WS=256 SA…
   8 3.147347  192.168.0.119   77.120.191.113  TCP        60 49183 → 80 [ACK] Seq=3664947550 Ack=4178236956 Win=65536 Len=0
   9 3.148205  192.168.0.119   77.120.191.113  HTTP      220 HEAD /ttt/2kraboviepa.exe?q=7761 HTTP/1.1
  10 3.384459  77.120.191.113  192.168.0.119   TCP        54 80 → 49183 [FIN, ACK] Seq=4178236956 Ack=3664947716 Win=65536 Len=0
  11 3.384793  192.168.0.119   77.120.191.113  TCP        60 49183 → 80 [ACK] Seq=3664947716 Ack=4178236957 Win=65536 Len=0
  12 3.926326  192.168.0.119   77.120.191.113  TCP        60 49183 → 80 [FIN, ACK] Seq=3664947716 Ack=4178236957 Win=65536 Len=0
  13 4.138102  77.120.191.113  192.168.0.119   TCP        54 80 → 49183 [ACK] Seq=4178236957 Ack=3664947717 Win=65536 Len=0

▸ Frame 9: 220 bytes on wire (1760 bits), 220 bytes captured (1760 bits)
▸ Ethernet II, Src: e0:00:00:01:27:77 (e0:00:00:01:27:77), Dst: ActionTe_d6:48:1d (00:1b:03:d6:48:1d)
▸ Internet Protocol Version 4, Src: 192.168.0.119, Dst: 77.120.191.113
▸ Transmission Control Protocol, Src Port: 49183, Dst Port: 80, Seq: 3664947550, Ack: 4178236956, Len: 166
▾ Hypertext Transfer Protocol
  ▸ HEAD /ttt/2kraboviepa.exe?q=7761 HTTP/1.1\r\n
    Connection: Keep-Alive\r\n
    Accept: */*\r\n
    Accept-Encoding: identity\r\n
    User-Agent: Microsoft BITS/7.5\r\n
    Host: ronymaniasnee.com\r\n
    \r\n
    [Full request URI: http://ronymaniasnee.com/ttt/2kraboviepa.exe?q=7761]
    [HTTP request 1/1]
```



Wireshark · Follow HTTP Stream (tcp.stream eq 0) · Maldoc

```
HEAD /ttt/2kraboviepa.exe?q=7761 HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Encoding: identity
User-Agent: Microsoft BITS/7.5
Host: ronymaniasnee.com
```

The request above can provide us with a lot of clues on how to develop an effective Suricata rule. A viable rule can be found below.

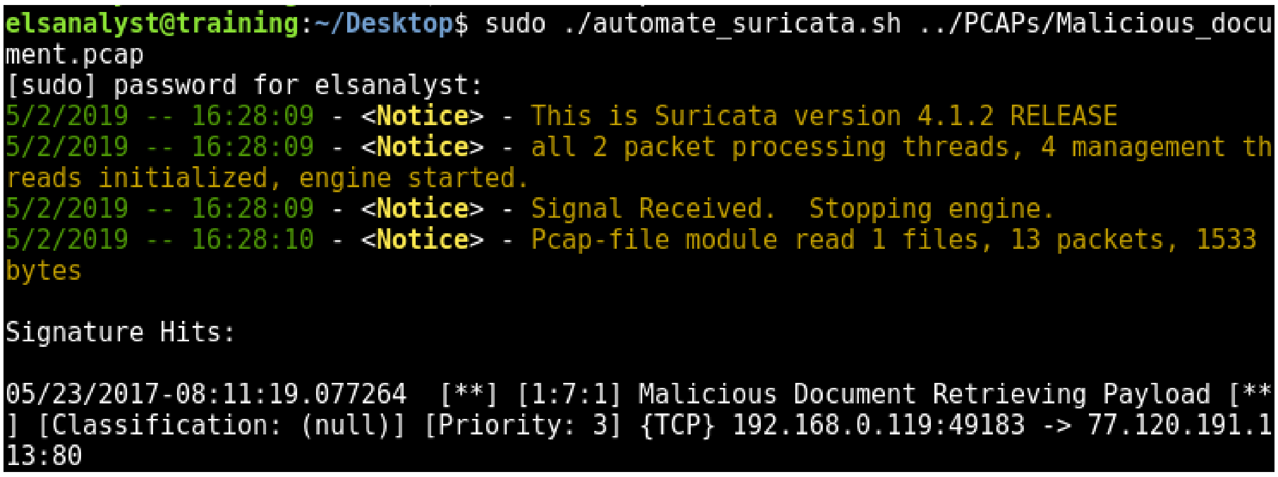
```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"Malicious Document
Retrieving Payload"; flow:established,to_server; content:".exe?q=";
fast_pattern; content:"Microsoft BITS/"; http_user_agent; depth:15;
pcre:"/\.exe\?q=[0-9]{3,5}$/U"; http_header_names; content:!"Referer";
sid:7; rev:1;)
```

`flow:established,to_server;` is used since we are dealing with TCP and the directionality is towards the malicious server. We won't focus on the curious-looking HEAD HTTP method so that we can catch variations. `fast_pattern;` is used so that the Suricata engine "focuses more" on the `.exe?q=` content ('User-Agent:' will be a match very often, but `.exe?q=` appears less often in network traffic). `pcre:"/\.exe\?q=[0-9]{3,5}$/U";` means the sensor should match every time it observes three to five numbers after *q=*. Finally, we are once again leveraging the lack of usual headers for detection purposes.

It's time to test the rule above. You can do so, as follows.

```
cd Desktop
sudo ./automate_suricata.sh ../PCAPs/Malicious_document.pcap
```

You should see the following.



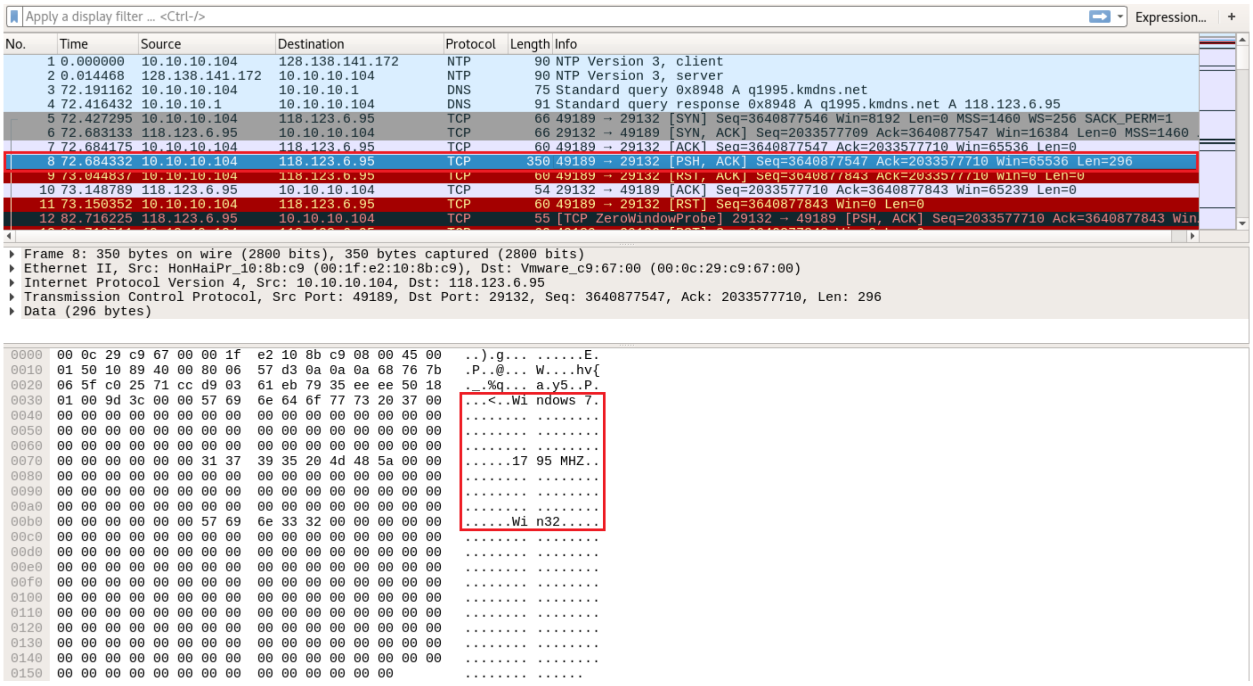
```
elsanalyst@training:~/Desktop$ sudo ./automate_suricata.sh ../PCAPs/Malicious_docu
ment.pcap
[sudo] password for elsanalyst:
5/2/2019 -- 16:28:09 - <Notice> - This is Suricata version 4.1.2 RELEASE
5/2/2019 -- 16:28:09 - <Notice> - all 2 packet processing threads, 4 management th
reads initialized, engine started.
5/2/2019 -- 16:28:09 - <Notice> - Signal Received.  Stopping engine.
5/2/2019 -- 16:28:10 - <Notice> - Pcap-file module read 1 files, 13 packets, 1533
bytes

Signature Hits:

05/23/2017-08:11:19.077264  [**] [1:7:1] Malicious Document Retrieving Payload [**
] [Classification: (null)] [Priority: 3] {TCP} 192.168.0.119:49183 -> 77.120.191.1
13:80
```

Task 3: Analyze the provided PCAP files and develop your own rules (Level: Intermediate)

Let's start with *DDoSClient.pcap*

We'll develop a Suricata rule based on our analysis of the *DDoSClient.pcap* file, considering that what we see inside the PCAP is malicious.

By opening *DDoSClient.pcap* in Wireshark, the first thing we notice is some curious-looking TCP traffic, containing information such as OS, CPU MHZ, and CPU Architecture.



```
No.  Time        Source           Destination      Protocol Length Info
   1 0.000000    10.10.10.104     128.138.141.172  NTP       90 NTP Version 3, client
   2 0.014468    128.138.141.172  10.10.10.104     NTP       90 NTP Version 3, server
   3 72.191162   10.10.10.104     10.10.10.1       DNS       75 Standard query 0x8948 A q1995.kmdns.net
   4 72.416432   10.10.10.1       10.10.10.104     DNS       91 Standard query response 0x8948 A q1995.kmdns.net A 118.123.6.95
   5 72.427295   10.10.10.104     118.123.6.95     TCP       66 49189 → 29132 [SYN] Seq=3640877546 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
   6 72.683133   118.123.6.95     10.10.10.104     TCP       66 29132 → 49189 [SYN, ACK] Seq=2033577709 Ack=3640877547 Win=16384 Len=0 MSS=1460
   7 72.684175   10.10.10.104     118.123.6.95     TCP       60 49189 → 29132 [ACK] Seq=3640877547 Ack=2033577710 Win=65536 Len=0
   8 72.684332   10.10.10.104     118.123.6.95     TCP      350 49189 → 29132 [PSH, ACK] Seq=3640877547 Ack=2033577710 Win=65536 Len=296
   9 73.044837   10.10.10.104     118.123.6.95     TCP       60 49189 → 29132 [RST, ACK] Seq=3640877843 Ack=2033577710 Win=0 Len=0
  10 73.148789   118.123.6.95     10.10.10.104     TCP       54 29132 → 49189 [ACK] Seq=2033577710 Ack=3640877843 Win=65239 Len=0
  11 73.150352   10.10.10.104     118.123.6.95     TCP       60 49189 → 29132 [RST] Seq=3640877843 Win=0 Len=0
  12 82.716225   118.123.6.95     10.10.10.104     TCP       55 [TCP ZeroWindowProbe] 29132 → 49189 [PSH, ACK] Seq=2033577710 Ack=3640877843 Win

▸ Frame 8: 350 bytes on wire (2800 bits), 350 bytes captured (2800 bits)
▸ Ethernet II, Src: HonHaiPr_10:8b:c9 (00:1f:e2:10:8b:c9), Dst: Vmware_c9:67:00 (00:0c:29:c9:67:00)
▸ Internet Protocol Version 4, Src: 10.10.10.104, Dst: 118.123.6.95
▸ Transmission Control Protocol, Src Port: 49189, Dst Port: 29132, Seq: 3640877547, Ack: 2033577710, Len: 296
▸ Data (296 bytes)

0000  00 0c 29 c9 67 00 00 1f  e2 10 8b c9 08 00 45 00   ..).g... ......E.
0010  01 50 10 89 40 00 80 06  57 d3 0a 0a 0a 68 76 7b   .P..@... W....hv{
0020  06 5f c0 25 71 cc d9 03  61 eb 79 35 ee ee 50 18   ._.%q... a.y5..P.
0030  01 00 9d 3c 00 00 57 69  6e 64 6f 77 73 20 37 00   ...<..Wi ndows 7.
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 31 37  39 35 20 4d 48 5a 00 00   ......17 95 MHZ..
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00b0  00 00 00 00 00 00 57 69  6e 33 32 00 00 00 00 00   ......Wi n32.....
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0150  00 00 00 00 00 00 00 00  00 00 00 00 00 00         ........ ......
```

The traffic above can provide us with enough clues on how to develop an effective Suricata rule. A viable rule can be found below.

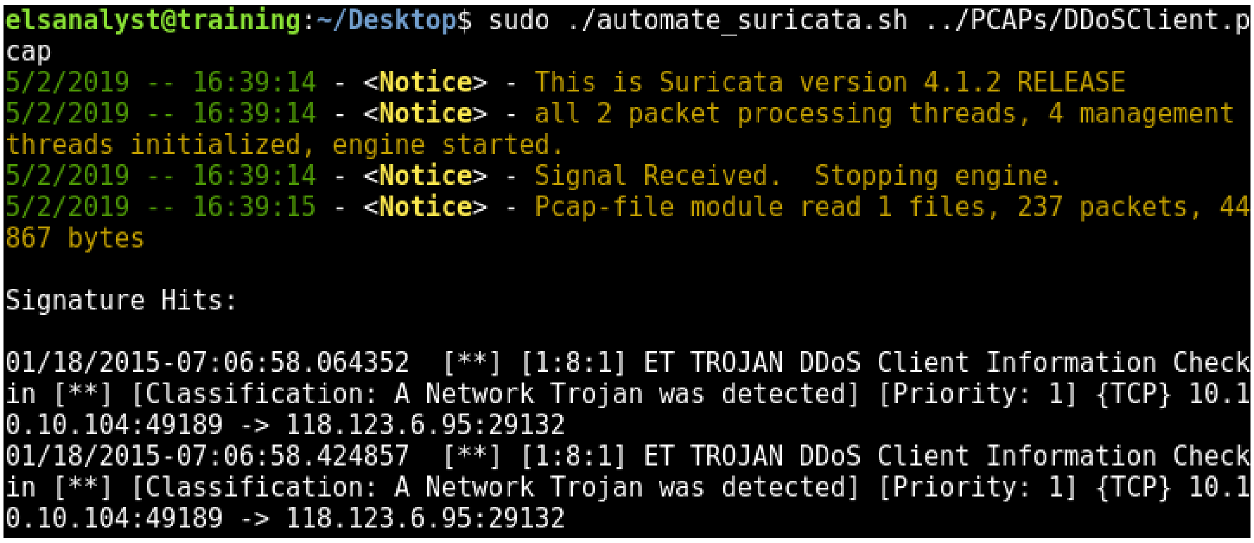
```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN DDoS
Client Information Checkin"; flow:established,to_server;
content:"Windows"; nocase; depth:7; content:"MHZ|00 00 00 00 00 00|";
distance:0; nocase; content:"|00 00 00 00 00 00|Win"; distance:0;
nocase; classtype:trojan-activity; sid:8; rev:1;)
```

Notice that we are using "Windows" and "MHZ" instead of "Windows 7" and "1795 MHZ", to detect variations. `depth:7;` is used because "Windows" is seven character's long. The first `distance:0;` is used because the nulls appear right after "Windows" (the number of nulls in the rule was chosen randomly).

It's time to test the rule above. You can do so, as follows.

```
cd Desktop
sudo ./automate_suricata.sh ../PCAPs/DDoSClient.pcap
```

You should see the following.

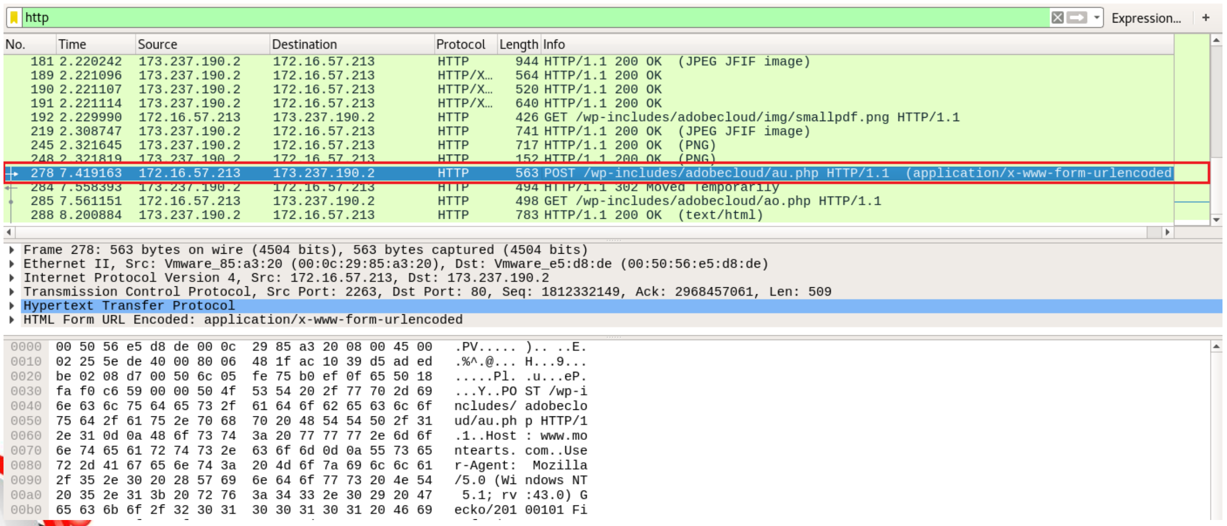


```
elsanalyst@training:~/Desktop$ sudo ./automate_suricata.sh ../PCAPs/DDoSClient.p
cap
5/2/2019 -- 16:39:14 - <Notice> - This is Suricata version 4.1.2 RELEASE
5/2/2019 -- 16:39:14 - <Notice> - all 2 packet processing threads, 4 management
threads initialized, engine started.
5/2/2019 -- 16:39:14 - <Notice> - Signal Received.  Stopping engine.
5/2/2019 -- 16:39:15 - <Notice> - Pcap-file module read 1 files, 237 packets, 44
867 bytes

Signature Hits:

01/18/2015-07:06:58.064352  [**] [1:8:1] ET TROJAN DDoS Client Information Check
in [**] [Classification: A Network Trojan was detected] [Priority: 1] {TCP} 10.1
0.10.104:49189 -> 118.123.6.95:29132
01/18/2015-07:06:58.424857  [**] [1:8:1] ET TROJAN DDoS Client Information Check
in [**] [Classification: A Network Trojan was detected] [Priority: 1] {TCP} 10.1
0.10.104:49189 -> 118.123.6.95:29132
```

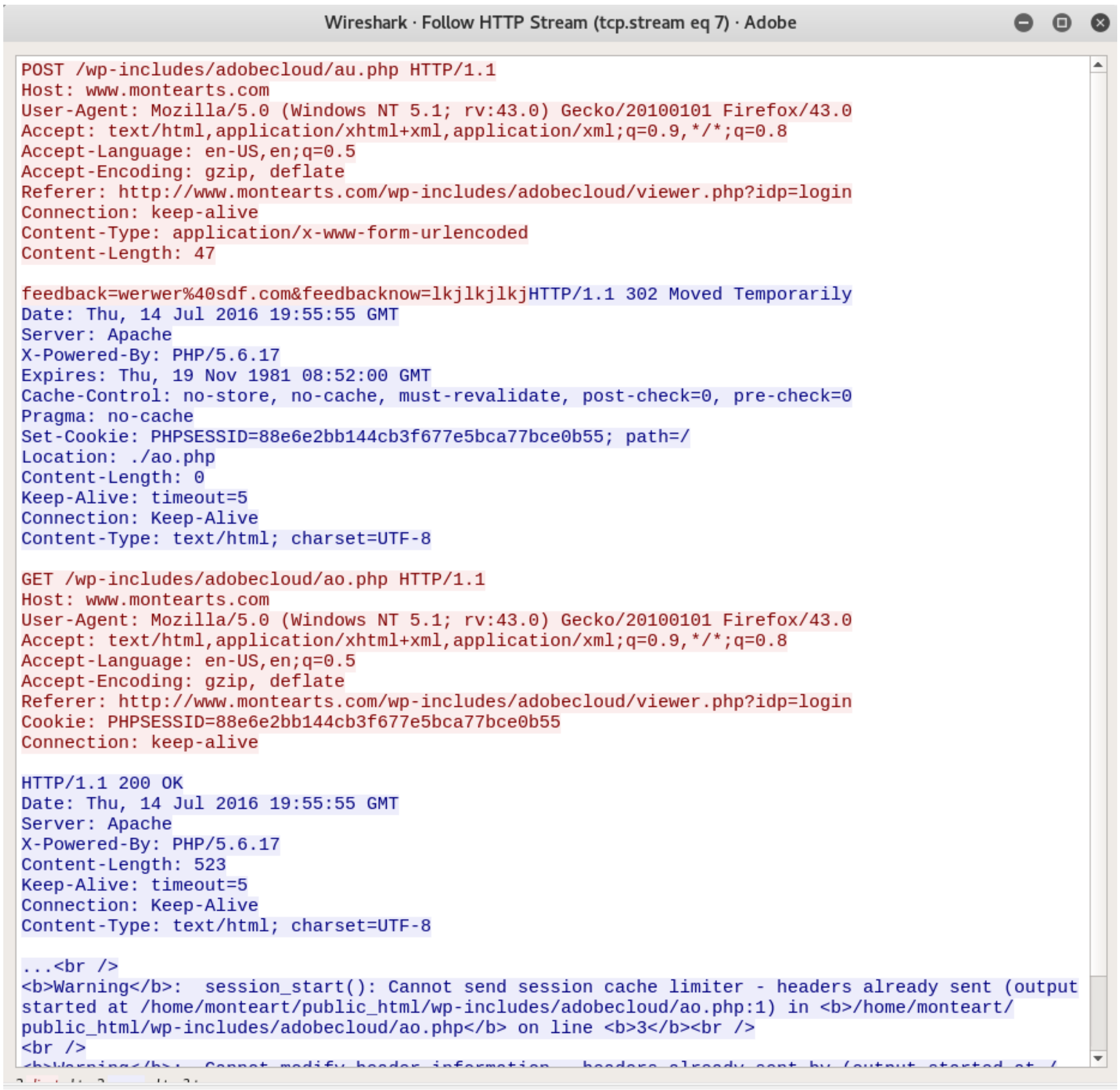
Now, it's time to analyze the *Adobe.pcap*

We'll develop a Suricata rule based on our analysis of the *Adobe.pcap*, considering that what we see inside the PCAP is malicious.

By opening *Adobe.pcap* in Wireshark and filtering so that we can see HTTP traffic only, the first thing we notice is this curious-looking POST request. Let's follow the whole stream.



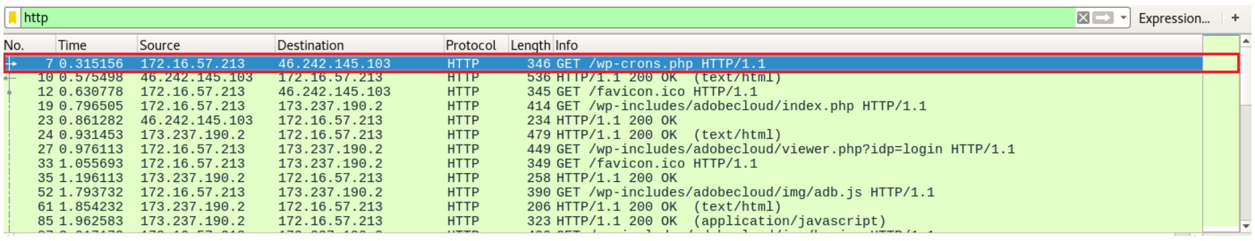
```
No.  Time      Source          Destination     Protocol Length Info
 181 2.220242  173.237.190.2   172.16.57.213   HTTP       944 HTTP/1.1 200 OK  (JPEG JFIF image)
 189 2.221096  173.237.190.2   172.16.57.213   HTTP/X…    564 HTTP/1.1 200 OK
 190 2.221107  173.237.190.2   172.16.57.213   HTTP/X…    520 HTTP/1.1 200 OK
 191 2.221114  173.237.190.2   172.16.57.213   HTTP/X…    640 HTTP/1.1 200 OK
 192 2.229990  172.16.57.213   173.237.190.2   HTTP       426 GET /wp-includes/adobecloud/img/smallpdf.png HTTP/1.1
 219 2.308747  173.237.190.2   172.16.57.213   HTTP       741 HTTP/1.1 200 OK  (JPEG JFIF image)
 245 2.321645  173.237.190.2   172.16.57.213   HTTP       717 HTTP/1.1 200 OK  (PNG)
 248 2.321819  173.237.190.2   172.16.57.213   HTTP       152 HTTP/1.1 200 OK  (PNG)
 278 7.419163  172.16.57.213   173.237.190.2   HTTP       563 POST /wp-includes/adobecloud/au.php HTTP/1.1  (application/x-www-form-urlencoded
 284 7.558393  173.237.190.2   172.16.57.213   HTTP       494 HTTP/1.1 302 Moved Temporarily
 285 7.561151  172.16.57.213   173.237.190.2   HTTP       498 GET /wp-includes/adobecloud/ao.php HTTP/1.1
 288 8.200884  173.237.190.2   172.16.57.213   HTTP       783 HTTP/1.1 200 OK  (text/html)

Frame 278: 563 bytes on wire (4504 bits), 563 bytes captured (4504 bits)
Ethernet II, Src: Vmware_85:a3:20 (00:0c:29:85:a3:20), Dst: Vmware_e5:d8:de (00:50:56:e5:d8:de)
Internet Protocol Version 4, Src: 172.16.57.213, Dst: 173.237.190.2
Transmission Control Protocol, Src Port: 2263, Dst Port: 80, Seq: 1812332149, Ack: 2968457061, Len: 509
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded

0000  00 50 56 e5 d8 de 00 0c  29 85 a3 20 08 00 45 00   .PV..... ).. ..E.
0010  02 25 5e de 40 00 80 06  48 1f ac 10 39 d5 ad ed   .%^.@... H...9...
0020  be 02 08 d7 00 50 6c 05  fe 75 b0 ef 0f 65 50 18   .....Pl. .u...eP.
0030  fa f0 c6 59 00 00 50 4f  53 54 20 2f 77 70 2d 69   ...Y..PO ST /wp-i
0040  6e 63 6c 75 64 65 73 2f  61 64 6f 62 65 63 6c 6f   ncludes/ adobeclo
0050  75 64 2f 61 75 2e 70 68  70 20 48 54 54 50 2f 31   ud/au.ph p HTTP/1
0060  2e 31 0d 0a 48 6f 73 74  3a 20 77 77 77 2e 6d 6f   .1..Host : www.mo
0070  6e 74 65 61 72 74 73 2e  63 6f 6d 0d 0a 55 73 65   ntearts. com..Use
0080  72 2d 41 67 65 6e 74 3a  20 4d 6f 7a 69 6c 6c 61   r-Agent:  Mozilla
0090  2f 35 2e 30 20 28 57 69  6e 64 6f 77 73 20 4e 54   /5.0 (Wi ndows NT
00a0  20 35 2e 31 3b 20 72 76  3a 34 33 2e 30 29 20 47    5.1; rv :43.0) G
00b0  65 63 6b 6f 2f 32 30 31  30 30 31 30 31 20 46 69   ecko/201 00101 Fi
```


Wireshark · Follow HTTP Stream (tcp.stream eq 7) · Adobe

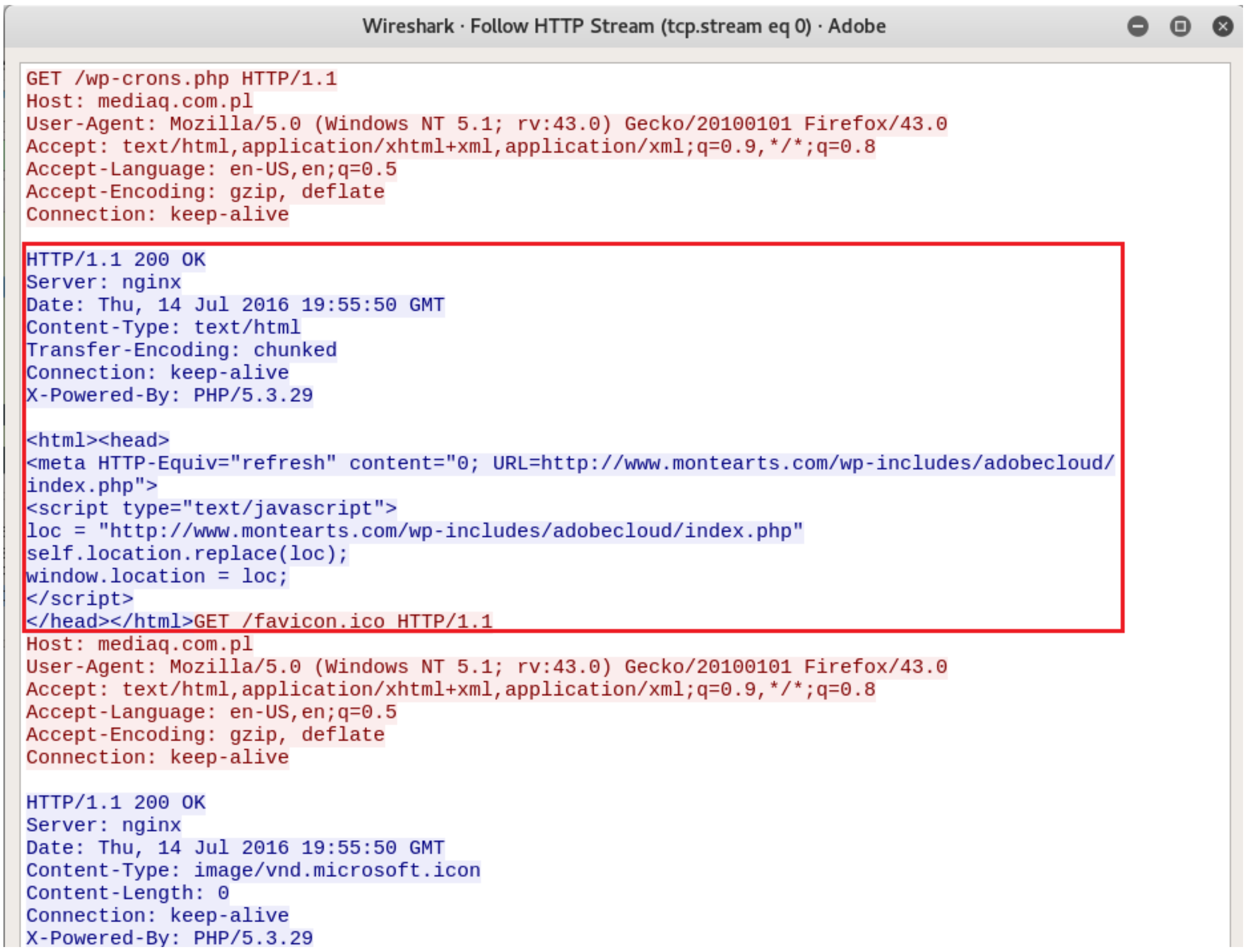
```
POST /wp-includes/adobecloud/au.php HTTP/1.1
Host: www.montearts.com
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:43.0) Gecko/20100101 Firefox/43.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.montearts.com/wp-includes/adobecloud/viewer.php?idp=login
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 47

feedback=werwer%40sdf.com&feedbacknow=lkjlkjlkjHTTP/1.1 302 Moved Temporarily
Date: Thu, 14 Jul 2016 19:55:55 GMT
Server: Apache
X-Powered-By: PHP/5.6.17
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID=88e6e2bb144cb3f677e5bca77bce0b55; path=/
Location: ./ao.php
Content-Length: 0
Keep-Alive: timeout=5
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

GET /wp-includes/adobecloud/ao.php HTTP/1.1
Host: www.montearts.com
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:43.0) Gecko/20100101 Firefox/43.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.montearts.com/wp-includes/adobecloud/viewer.php?idp=login
Cookie: PHPSESSID=88e6e2bb144cb3f677e5bca77bce0b55
Connection: keep-alive

HTTP/1.1 200 OK
Date: Thu, 14 Jul 2016 19:55:55 GMT
Server: Apache
X-Powered-By: PHP/5.6.17
Content-Length: 523
Keep-Alive: timeout=5
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

...<br />
<b>Warning</b>:  session_start(): Cannot send session cache limiter - headers already sent (output
started at /home/monteart/public_html/wp-includes/adobecloud/ao.php:1) in <b>/home/monteart/
public_html/wp-includes/adobecloud/ao.php</b> on line <b>3</b><br />
<br />
```

The request above can provide us with a lot of clues on how to develop an effective Suricata rule. Try creating one on your own…

There was another curious-looking HTTP, the following one.



```
No.  Time      Source          Destination     Protocol Length Info
  7 0.315156  172.16.57.213   46.242.145.103  HTTP      346 GET /wp-crons.php HTTP/1.1
 10 0.575498  46.242.145.103  172.16.57.213   HTTP      536 HTTP/1.1 200 OK  (text/html)
 12 0.630778  172.16.57.213   46.242.145.103  HTTP      345 GET /favicon.ico HTTP/1.1
 19 0.796505  172.16.57.213   173.237.190.2   HTTP      414 GET /wp-includes/adobecloud/index.php HTTP/1.1
 23 0.861282  46.242.145.103  172.16.57.213   HTTP      234 HTTP/1.1 200 OK
 24 0.931453  173.237.190.2   172.16.57.213   HTTP      479 HTTP/1.1 200 OK  (text/html)
 27 0.976113  172.16.57.213   173.237.190.2   HTTP      449 GET /wp-includes/adobecloud/viewer.php?idp=login HTTP/1.1
 33 1.055693  172.16.57.213   173.237.190.2   HTTP      349 GET /favicon.ico HTTP/1.1
 35 1.196113  173.237.190.2   172.16.57.213   HTTP      258 HTTP/1.1 200 OK
 52 1.793732  172.16.57.213   173.237.190.2   HTTP      390 GET /wp-includes/adobecloud/img/adb.js HTTP/1.1
 61 1.854232  173.237.190.2   172.16.57.213   HTTP      206 HTTP/1.1 200 OK  (text/html)
 85 1.962583  173.237.190.2   172.16.57.213   HTTP      323 HTTP/1.1 200 OK  (application/javascript)
```



```
GET /wp-crons.php HTTP/1.1
Host: mediaq.com.pl
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:43.0) Gecko/20100101 Firefox/43.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Server: nginx
Date: Thu, 14 Jul 2016 19:55:50 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.3.29

<html><head>
<meta HTTP-Equiv="refresh" content="0; URL=http://www.montearts.com/wp-includes/adobecloud/
index.php">
<script type="text/javascript">
loc = "http://www.montearts.com/wp-includes/adobecloud/index.php"
self.location.replace(loc);
window.location = loc;
</script>
</head></html>GET /favicon.ico HTTP/1.1
Host: mediaq.com.pl
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:43.0) Gecko/20100101 Firefox/43.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Server: nginx
Date: Thu, 14 Jul 2016 19:55:50 GMT
Content-Type: image/vnd.microsoft.icon
Content-Length: 0
Connection: keep-alive
X-Powered-By: PHP/5.3.29
```

What is suspicious about the portion inside the red rectangle above, is the way in which it redirects the client elsewhere. Typically, you would see either `<META HTTP-EQUIV="refresh"` or redirection through JavaScript, not both at the same time.

This is quite uncommon, so let's make a rule out of it. A viable rule can be found below.

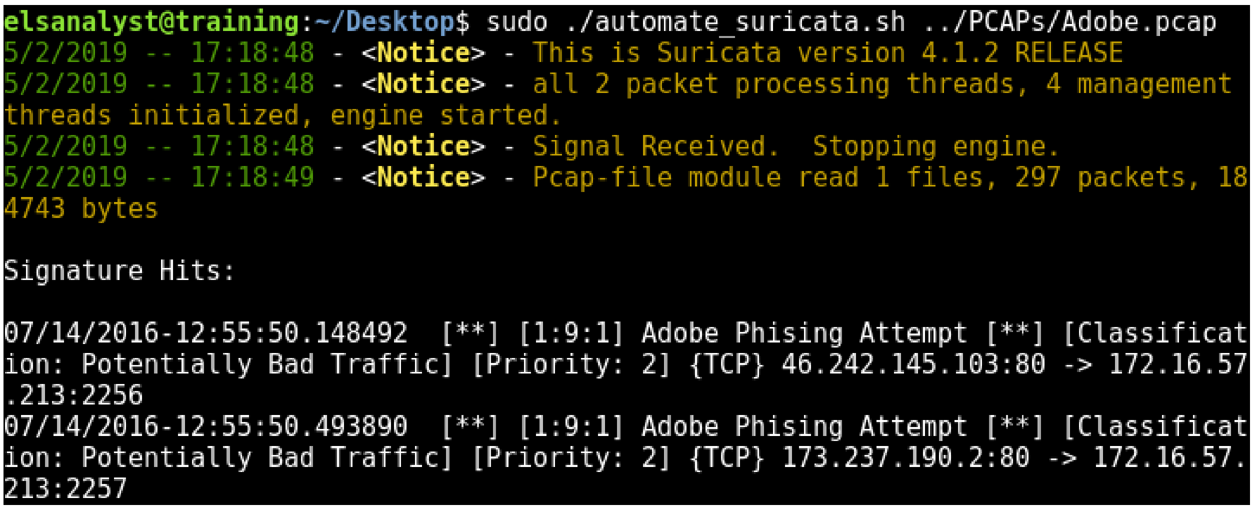
```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"Adobe Phising
Attempt"; flow:established,to_client; content:"200"; http_stat_code;
http_content_type; content:"text/html"; nocase; file_data;
content:"<META HTTP-EQUIV="; nocase; within:100; content:"refresh";
distance:1; nocase; within:7; content:"self.location.replace"; nocase;
distance:0; content:"window.location"; nocase; distance:0;
classtype:bad-unknown; sid:9; rev:1;)
```

file_data; is a buffer including what will be rendered in the browser. `within:100` means we want to see the content within the first 100 bytes. The remaining part is easy to comprehend.

It's time to test the rule above. You can do so, as follows.

```
cd Desktop
sudo ./automate_suricata.sh ../PCAPs/Adobe.pcap
```

You should see the following.



```
elsanalyst@training:~/Desktop$ sudo ./automate_suricata.sh ../PCAPs/Adobe.pcap
5/2/2019 -- 17:18:48 - <Notice> - This is Suricata version 4.1.2 RELEASE
5/2/2019 -- 17:18:48 - <Notice> - all 2 packet processing threads, 4 management
threads initialized, engine started.
5/2/2019 -- 17:18:48 - <Notice> - Signal Received.  Stopping engine.
5/2/2019 -- 17:18:49 - <Notice> - Pcap-file module read 1 files, 297 packets, 18
4743 bytes

Signature Hits:

07/14/2016-12:55:50.148492  [**] [1:9:1] Adobe Phising Attempt [**] [Classificat
ion: Potentially Bad Traffic] [Priority: 2] {TCP} 46.242.145.103:80 -> 172.16.57
.213:2256
07/14/2016-12:55:50.493890  [**] [1:9:1] Adobe Phising Attempt [**] [Classificat
ion: Potentially Bad Traffic] [Priority: 2] {TCP} 173.237.190.2:80 -> 172.16.57.
213:2257
```

Suricata Resources:

1. https://www.stamus-networks.com/open-source/
2. https://resources.sei.cmu.edu/asset_files/Presentation/2016_017_001_449890.pdf
3. https://blog.inliniac.net/2014/04/08/detecting-openssl-heartbleed-with-suricata/
4. https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/decoding-hancitor-malware-with-suricata-and-lua/
5. https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/advanced-malware-detection-with-suricata-lua-scripting/