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:

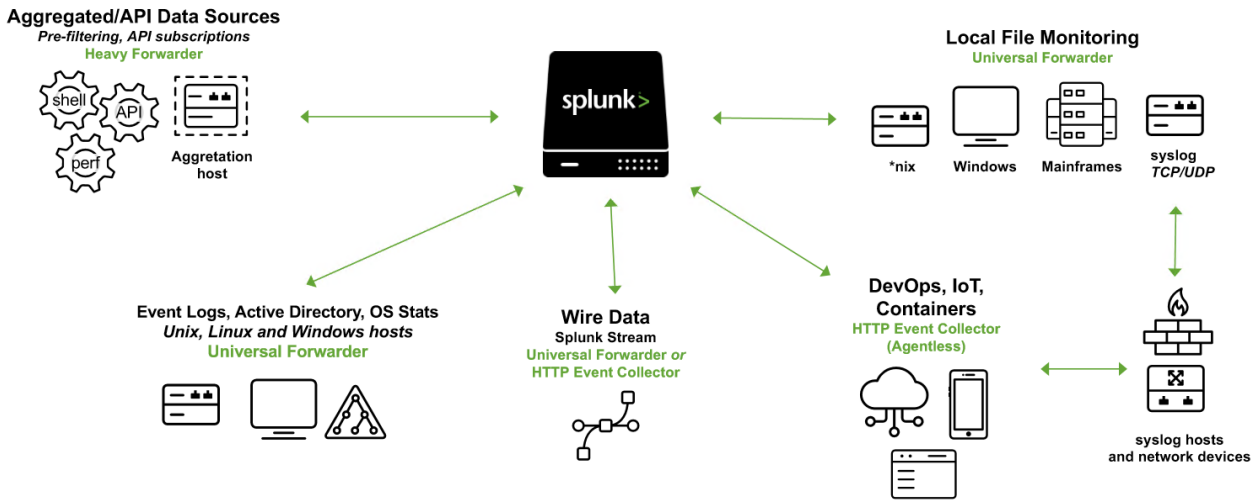
- Have better visibility over a network
- Respond to incidents timely and effectively
- Proactively hunt for threats

Introduction To Splunk

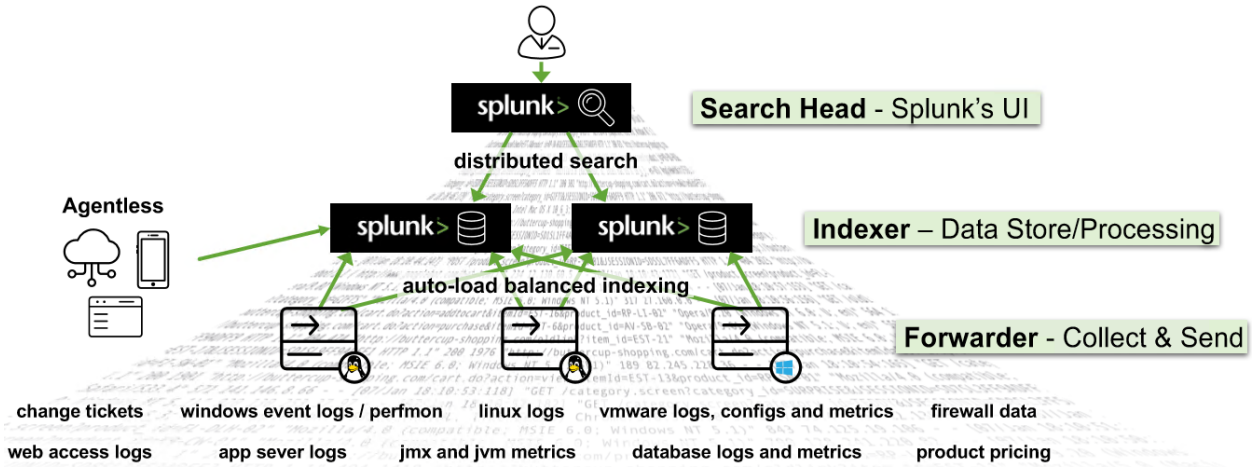
Splunk's creators describe it as a solution to aggregate, analyze and get answers from machine data. Splunk can be used for Application Management, Operations Management, Security & Compliance, etc.

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.

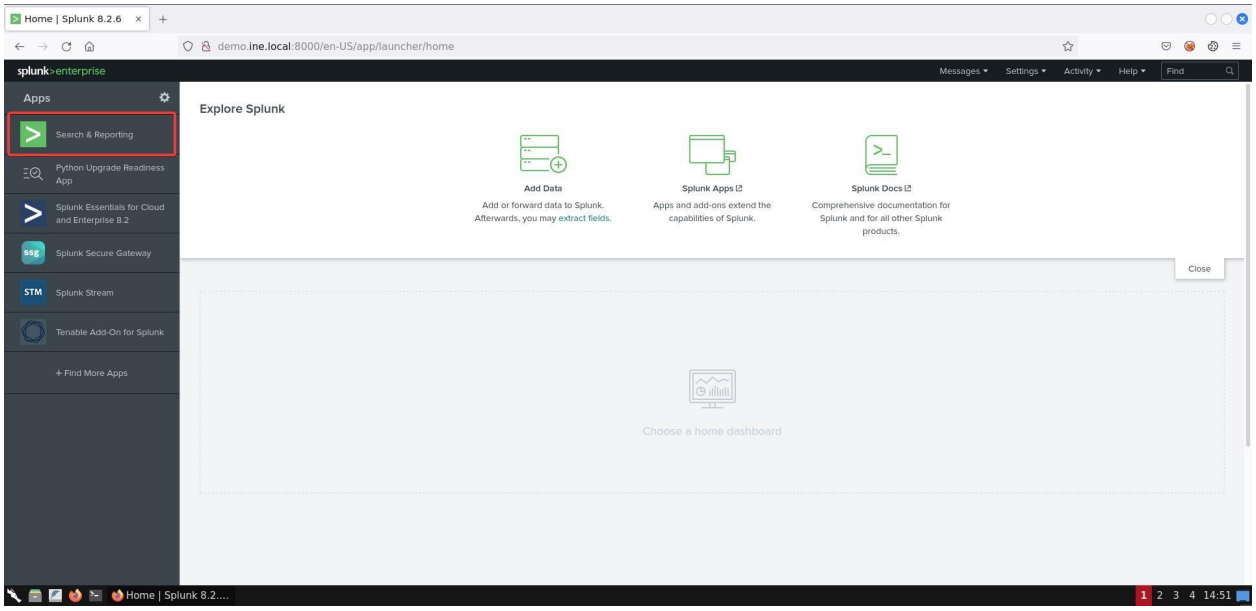
Splunk Users and Roles:

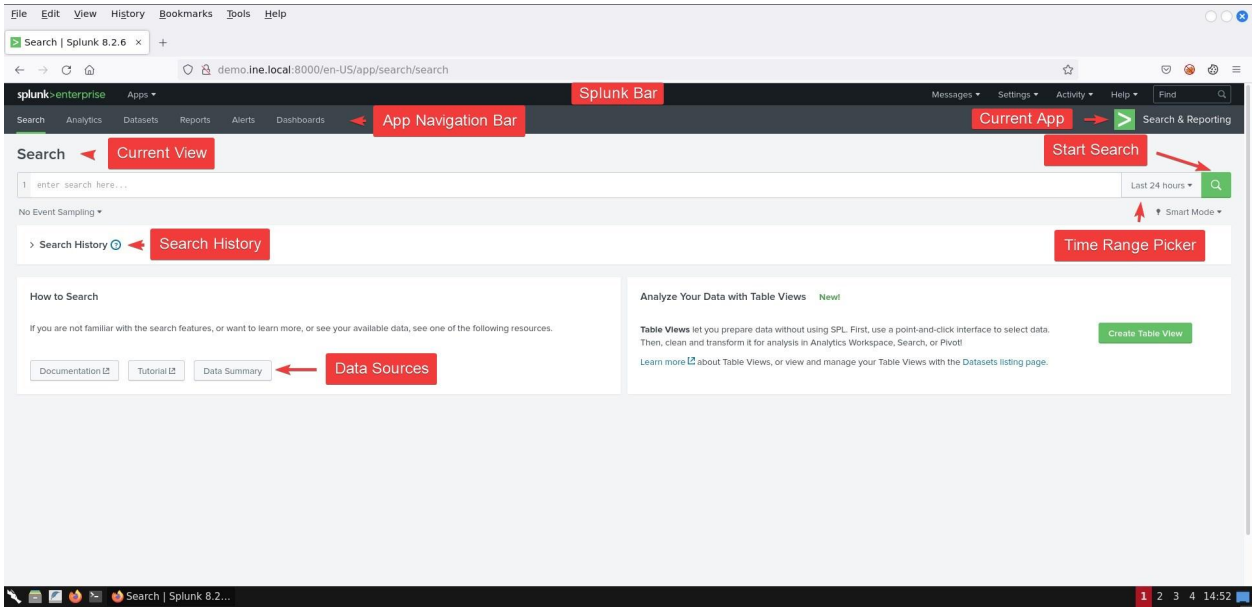
Splunk users are assigned roles which determine their capabilities and data access. Out of the box, there are three main roles:

- **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.

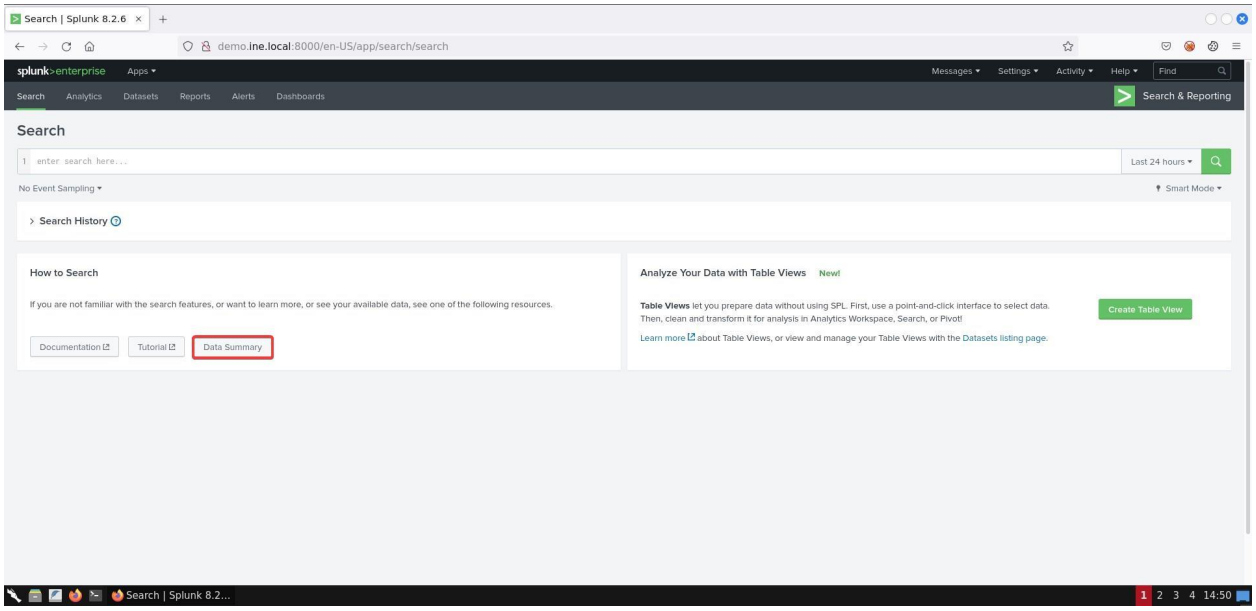
Splunk's Search & Reporting App:

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

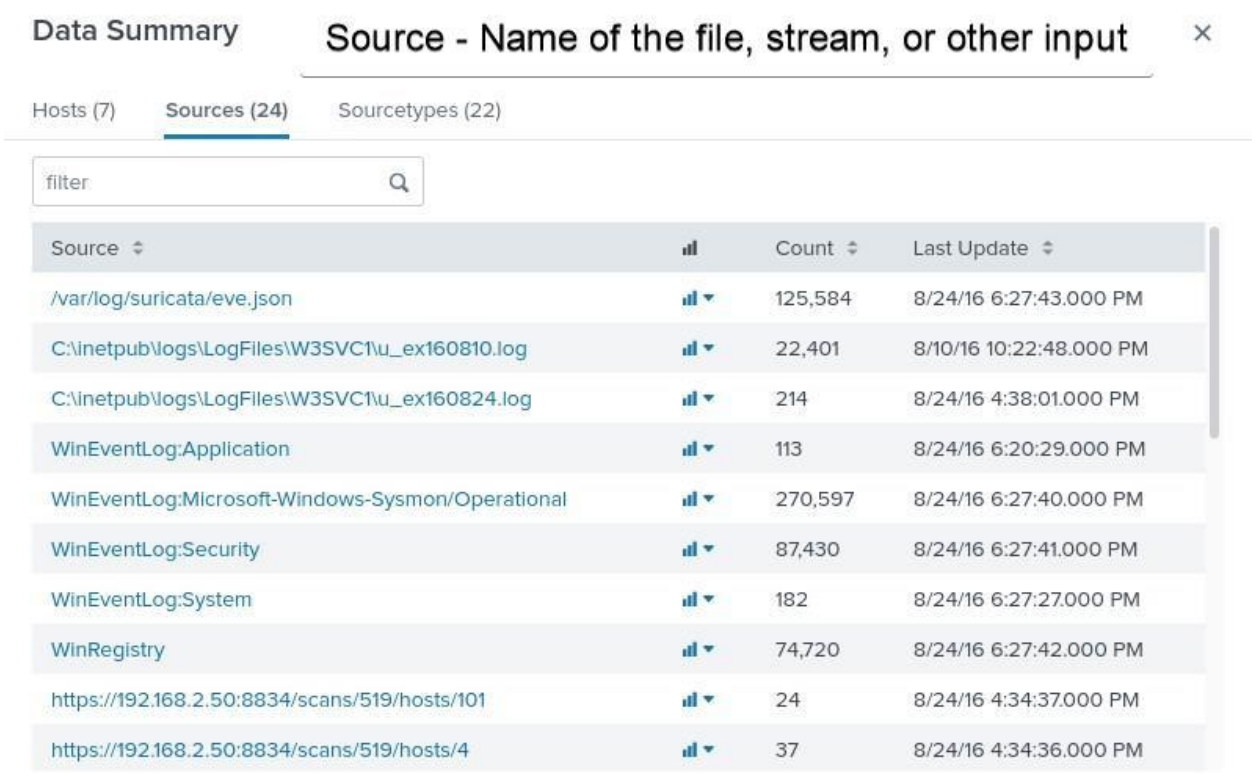


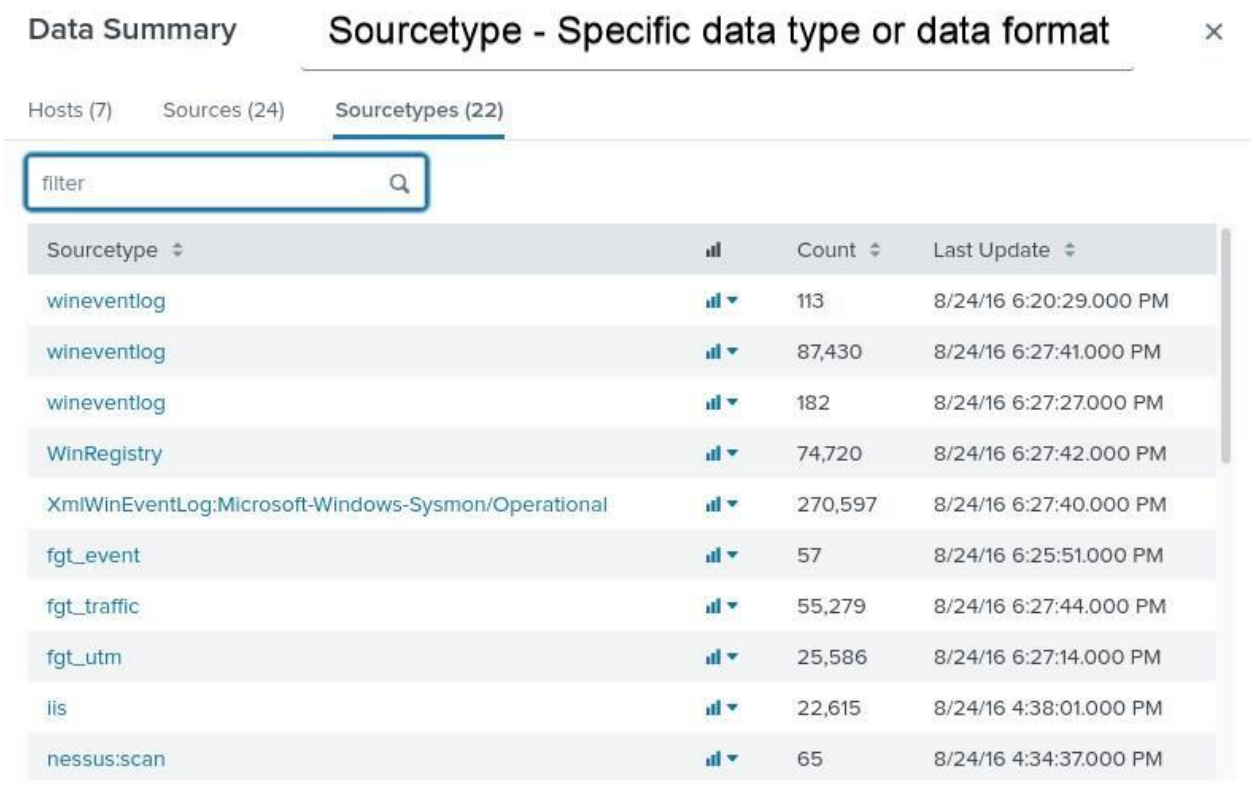


Data Summary can provide you with hosts, sources or sourcetypes on separate tabs.

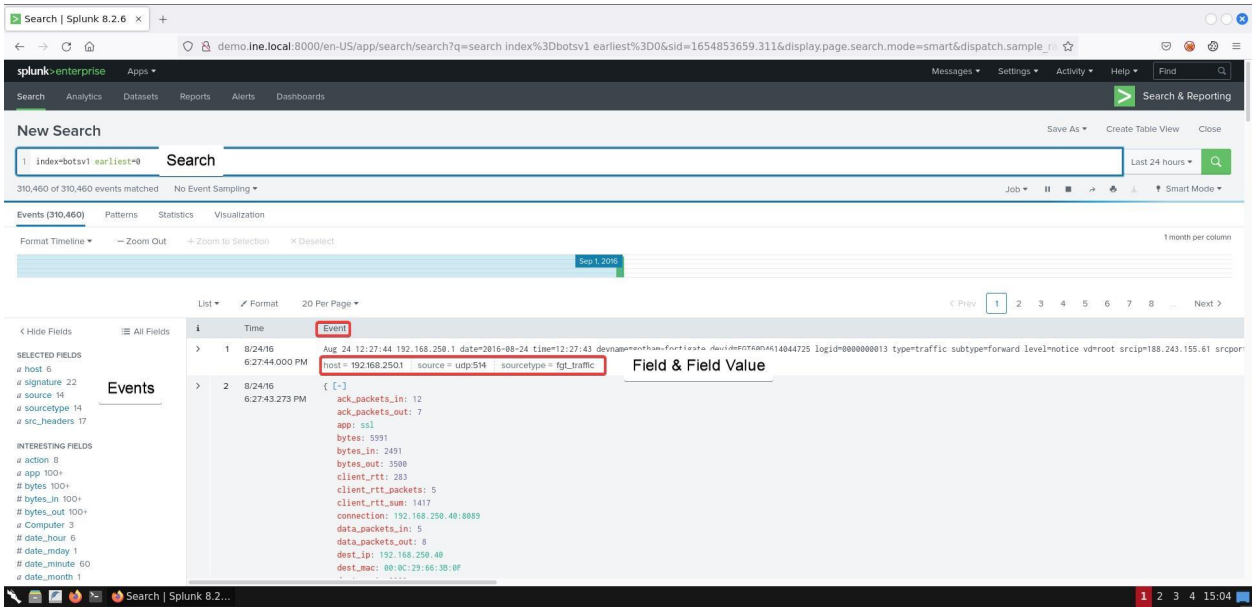

Search | Splunk 8.2.6
demo.ine.local:8000/en-US/app/search/search
splunk>enterprise
Apps
Messages
Settings
Activity
Help
Find
Search
Analytics
Datasets
Reports
Alerts
Dashboards
Search & Reporting
Search
enter search here...
Last 24 hours
No Event Sampling
Smart Mode
Search History
How to Search
If you are not familiar with the search features, or want to learn more, or see your available data, see one of the following resources.
Documentation
Tutorial
Data Summary
Analyze Your Data with Table Views
New!
Table Views let you prepare data without using SPL. First, use a point-and-click interface to select data. Then, clean and transform it for analysis in Analytics Workspace, Search, or Pivot!
Create Table View
Learn more about Table Views, or view and manage your Table Views with the Datasets listing page.
Search | Splunk 8.2...
1 2 3 4 14:50


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.



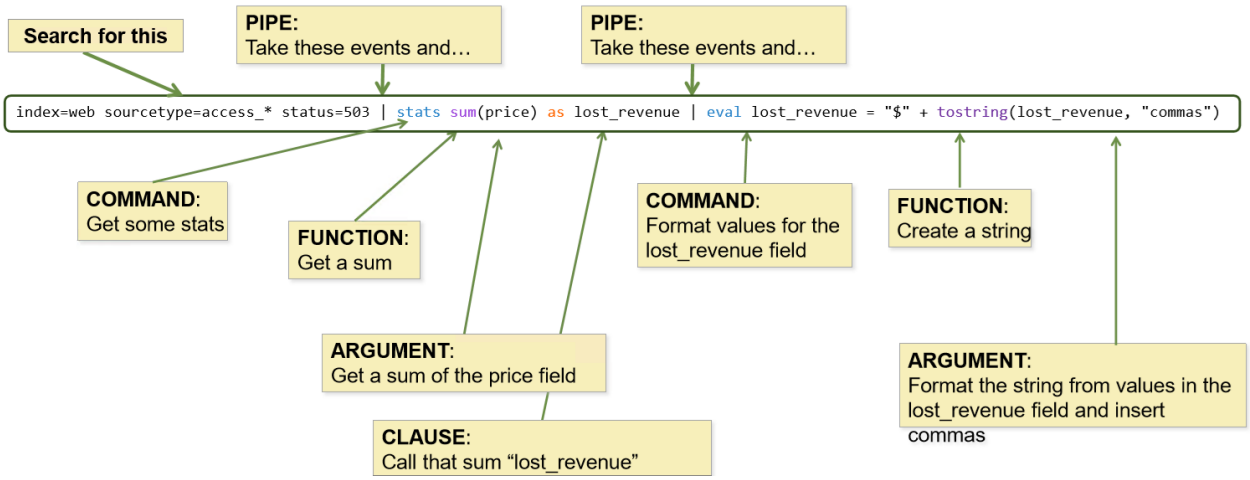
Splunk's Search Processing Language (SPL):

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.

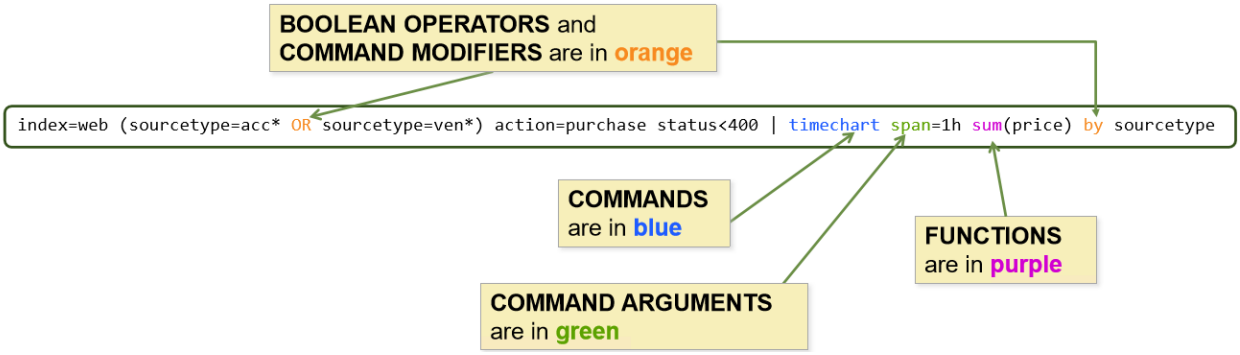
2. **Commands,** where you specify how you want to manipulate the results. For example, create a chart, compute statistics, etc.

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.

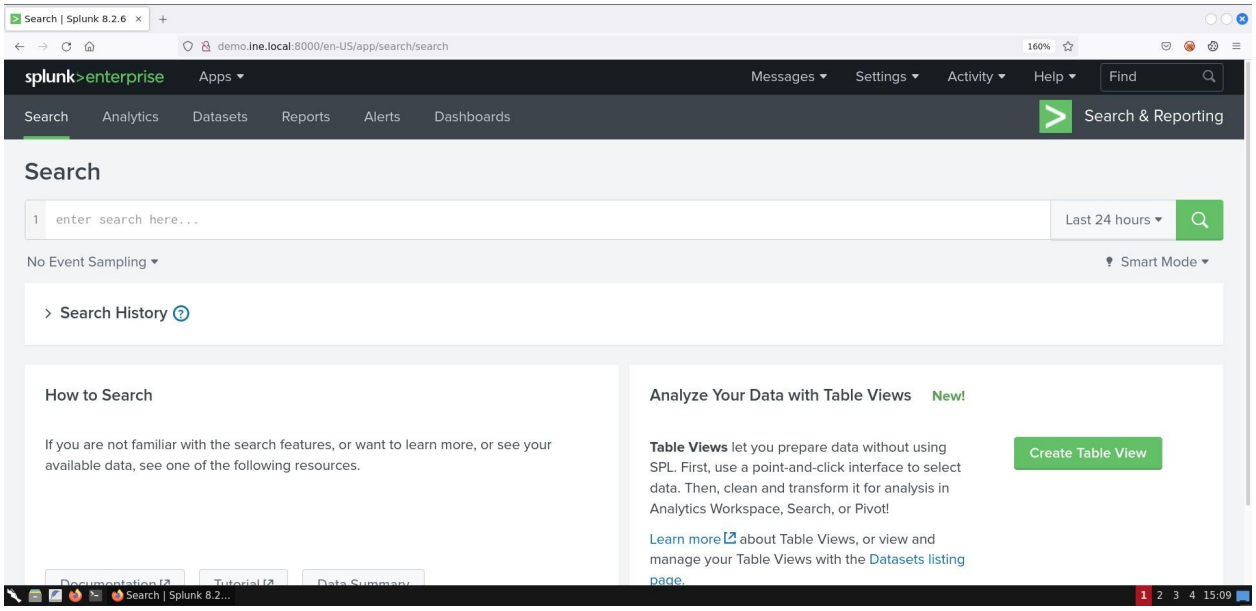
SOLUTIONS

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

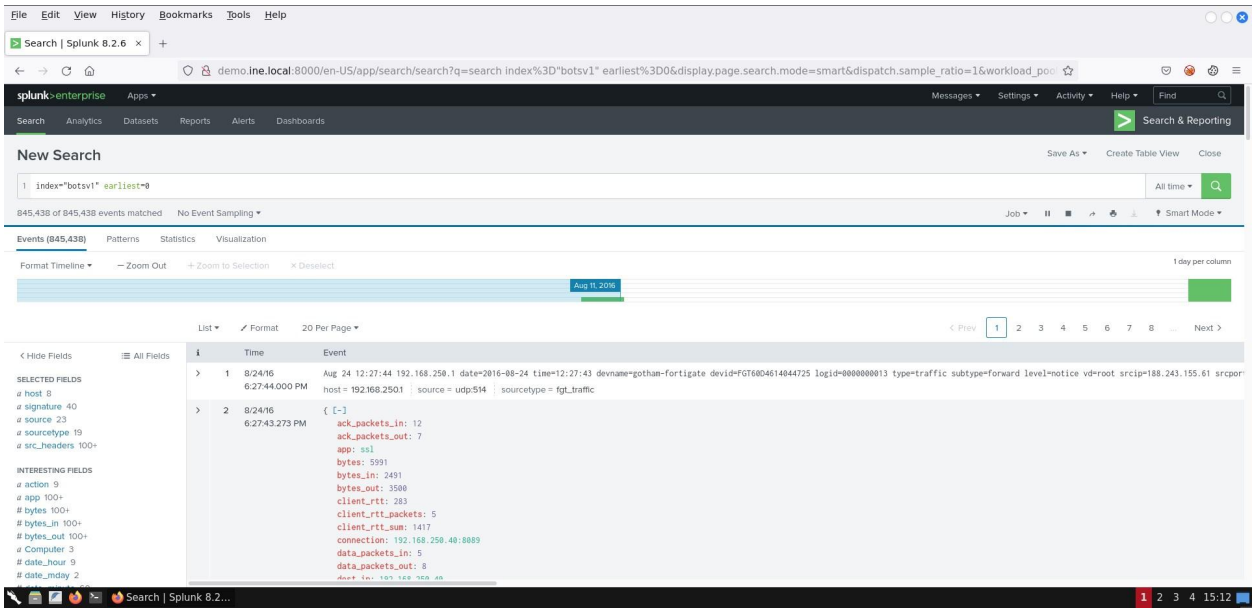
Once you are logged into Splunk's web management interface, click the **Search & Reporting** application that resides on the **Apps** column on your left. You should see something similar to the below.



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.

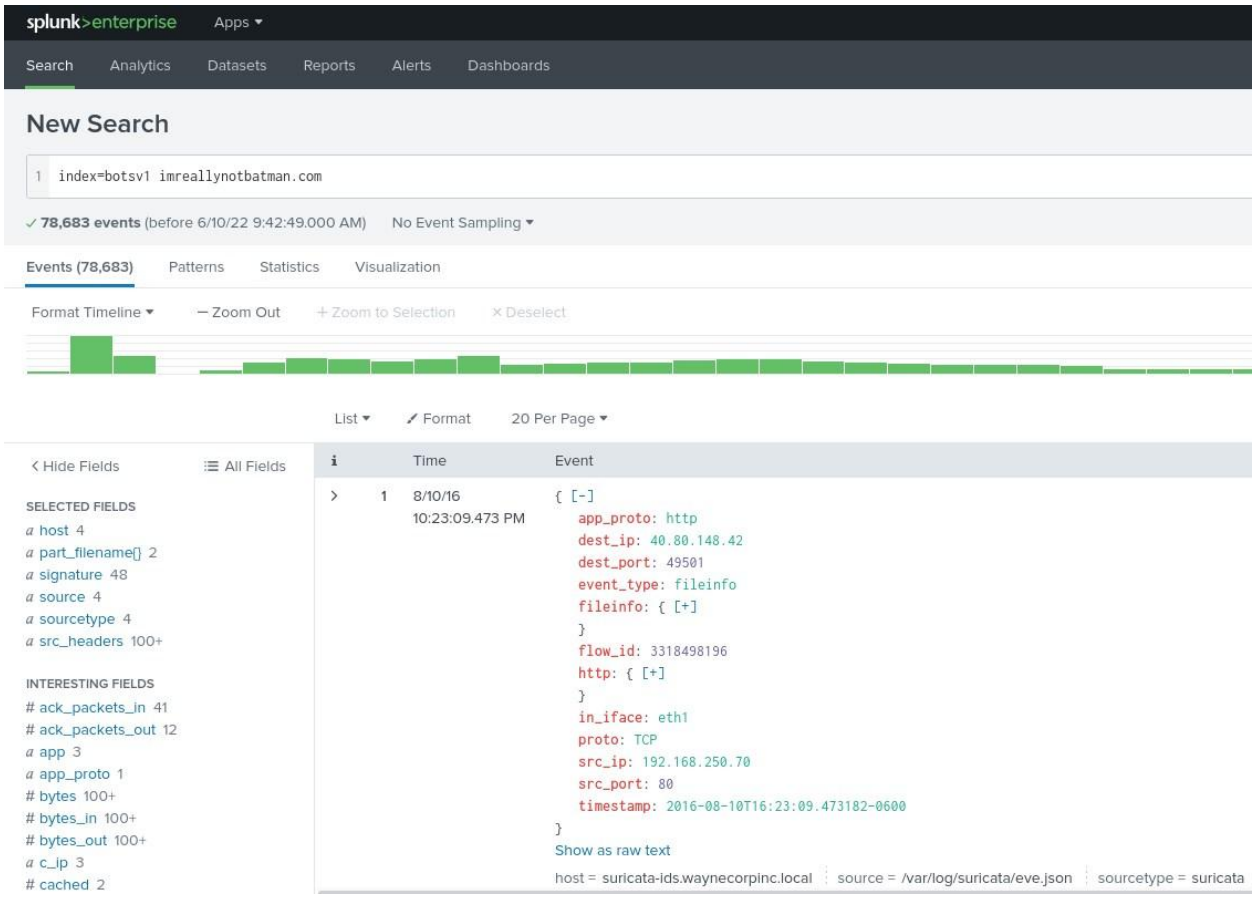


Now that we know everything worked as expected, let's identify any reconnaissance activities against Wayne Enterprises. As a reminder, the organization's website is **imreallynotbatman.com**.

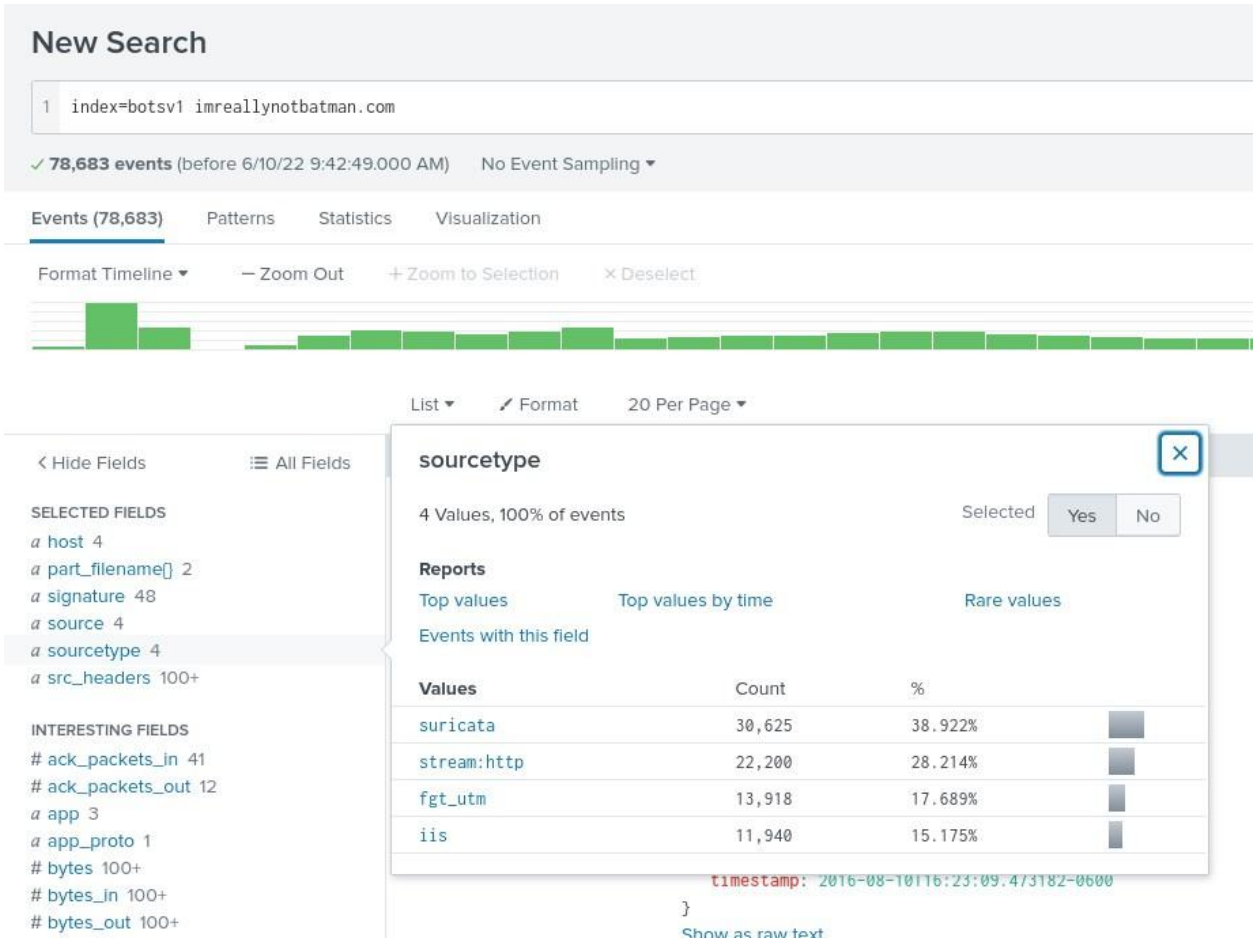
The first thing we should do is determine the sourcetypes to search. Specifically, we should first determine the sourcetypes that are associated with **imreallynotbatman.com**. We can do so by changing the time range picker to **All time** and submitting the following Splunk search.

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

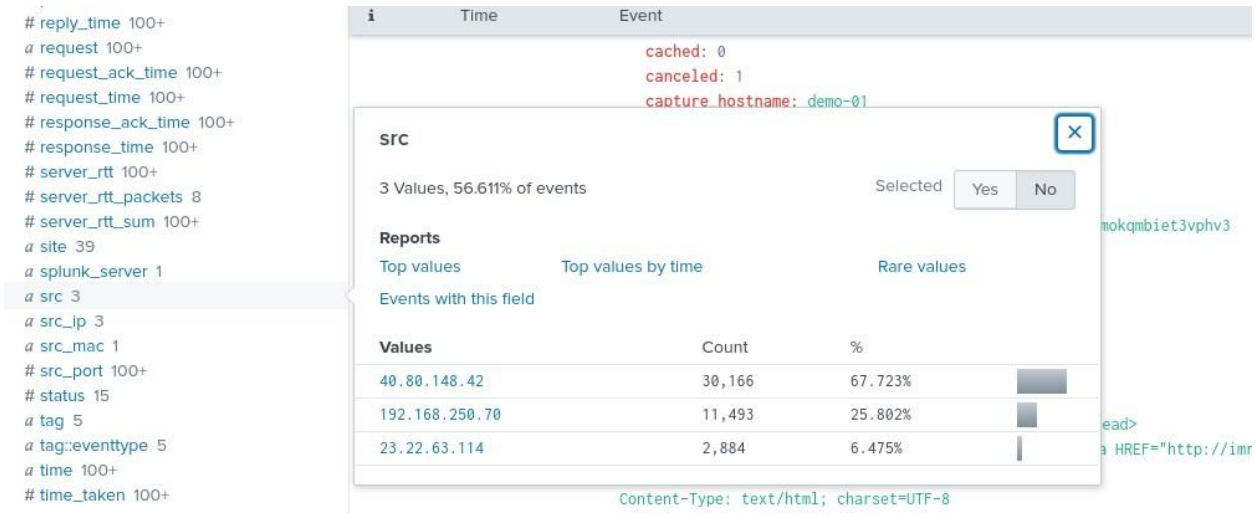
You should see something similar to the below.



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

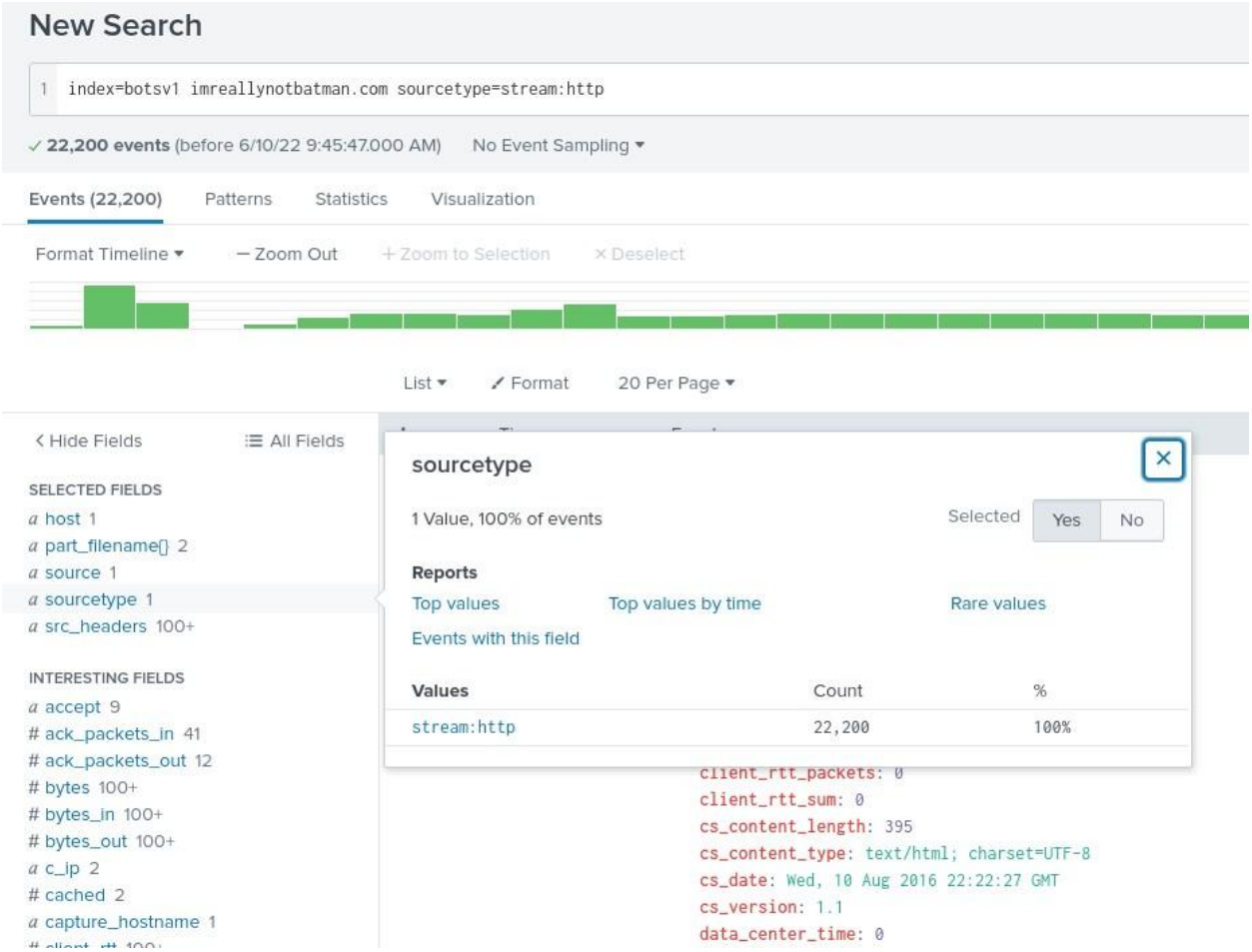


Let's also identify all source addresses. We can do so through the previous search, but this time we will scroll down and click on the **src** field, as follows.



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.)

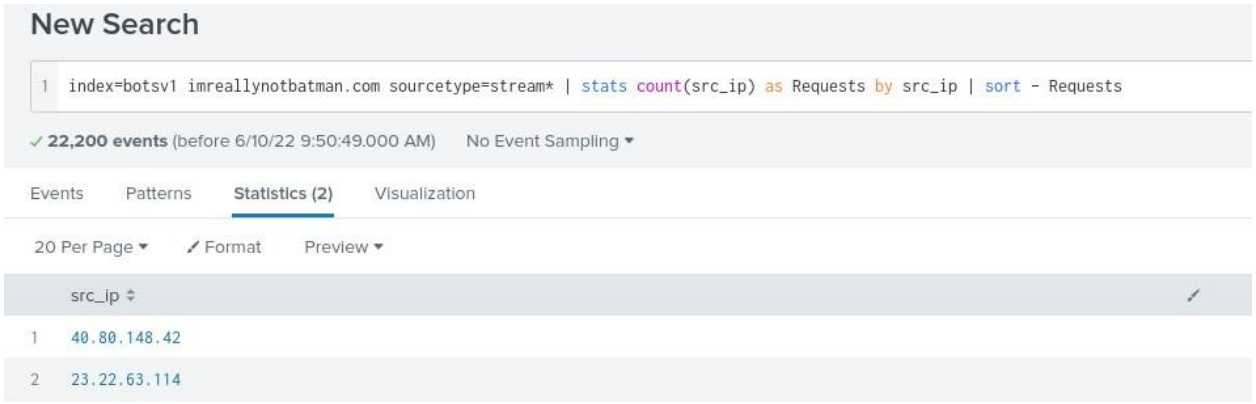
index=botsv1 imreallynotbatman.com sourcetype=stream:http



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.

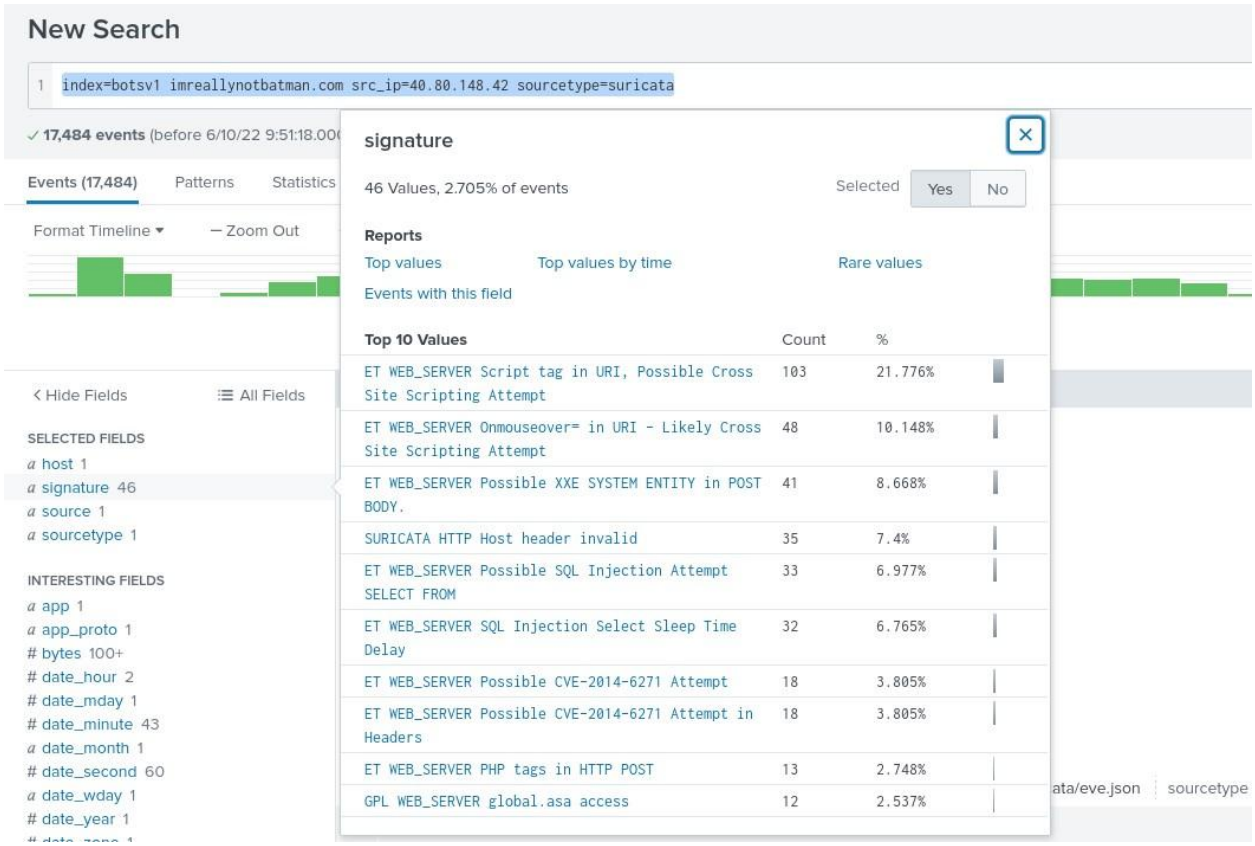
index=botsv1 imreallynotbatman.com sourcetype=stream* | stats count(src_ip) as Requests by src_ip | sort - Requests



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.

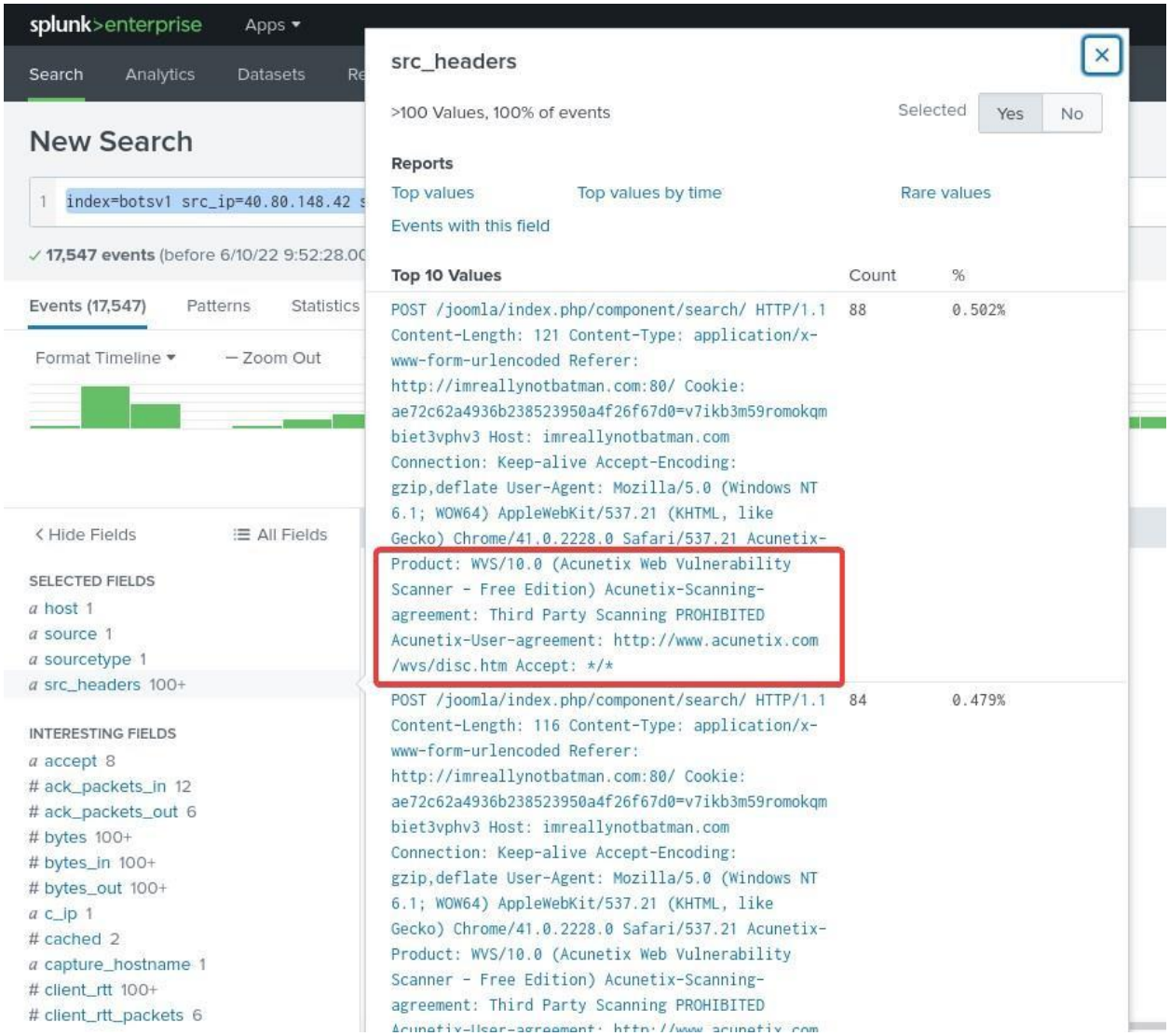


From the Suricata signatures that were triggered, we can conclude that **40.80.148.42** was actually scanning **imreallynotbatman.com**.

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.

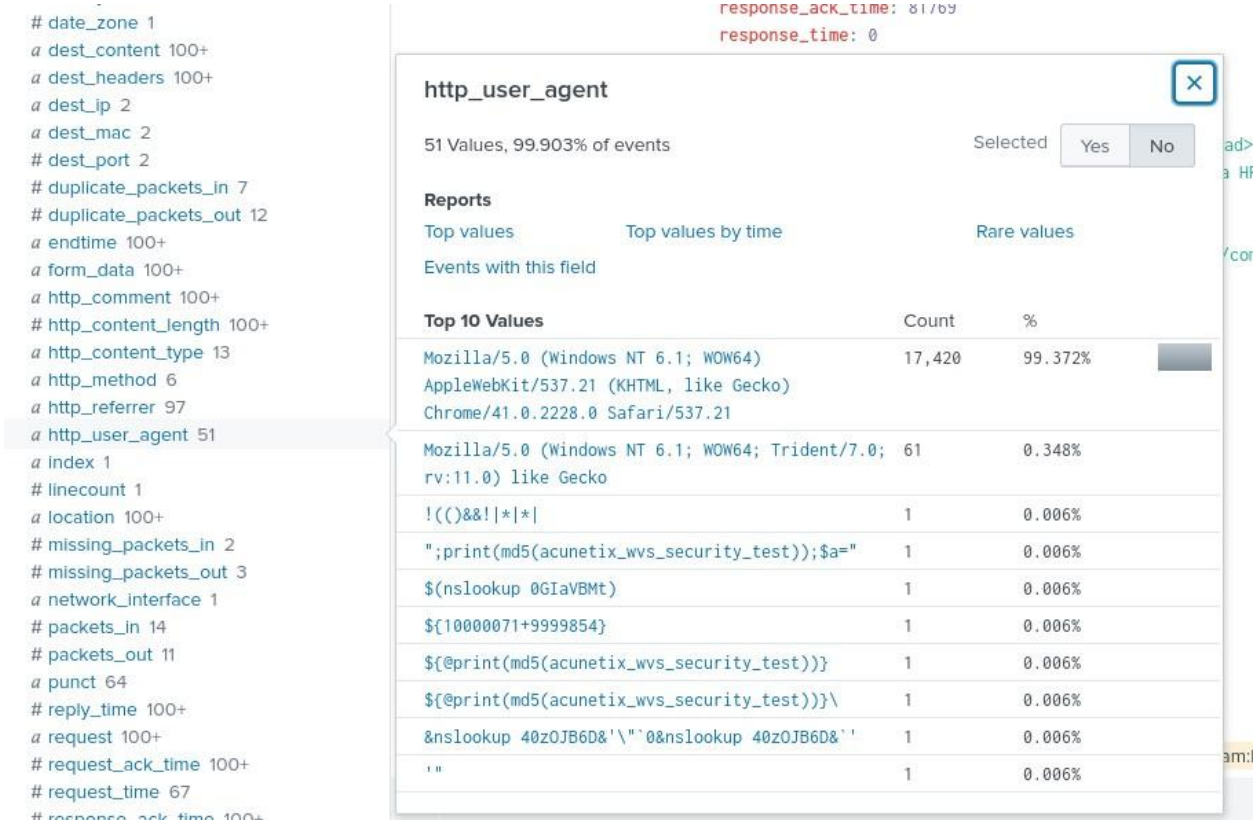
```
index=botsv1 src_ip=40.80.148.42 sourcetype=stream:http
```

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



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.



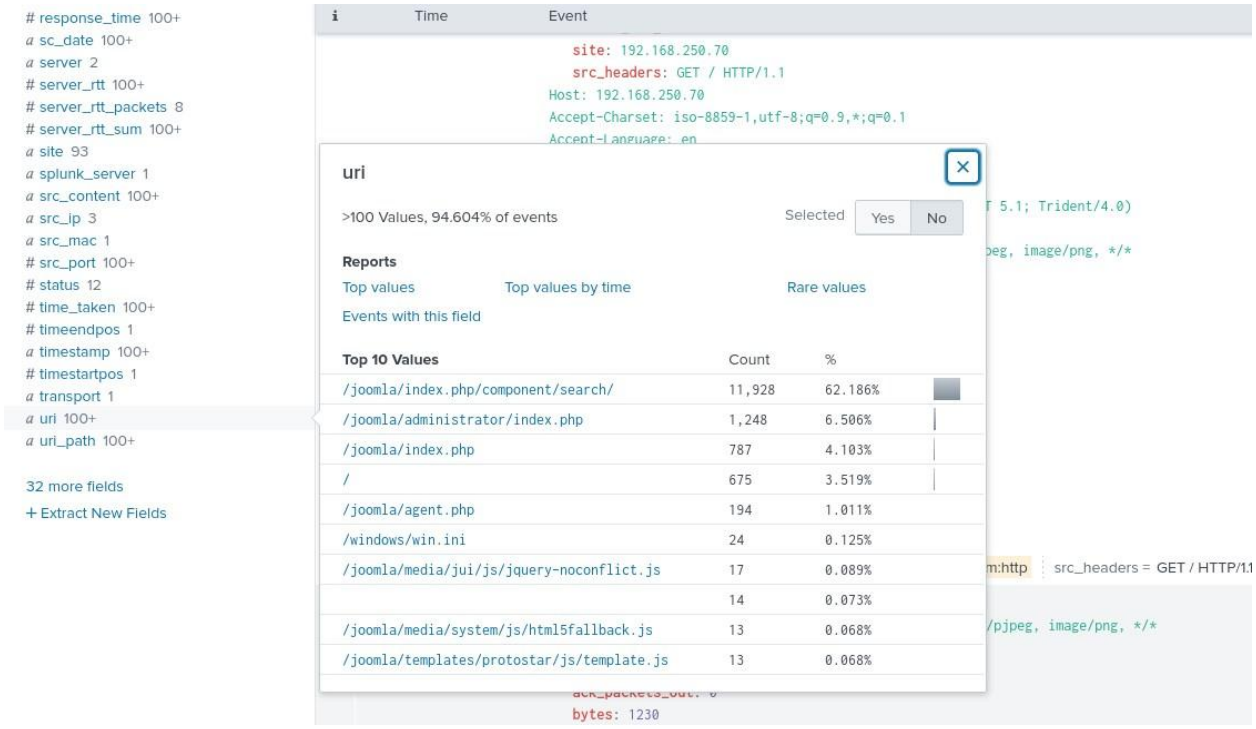
We can easily identify which server was the target through the same search and the **dest** field.

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.

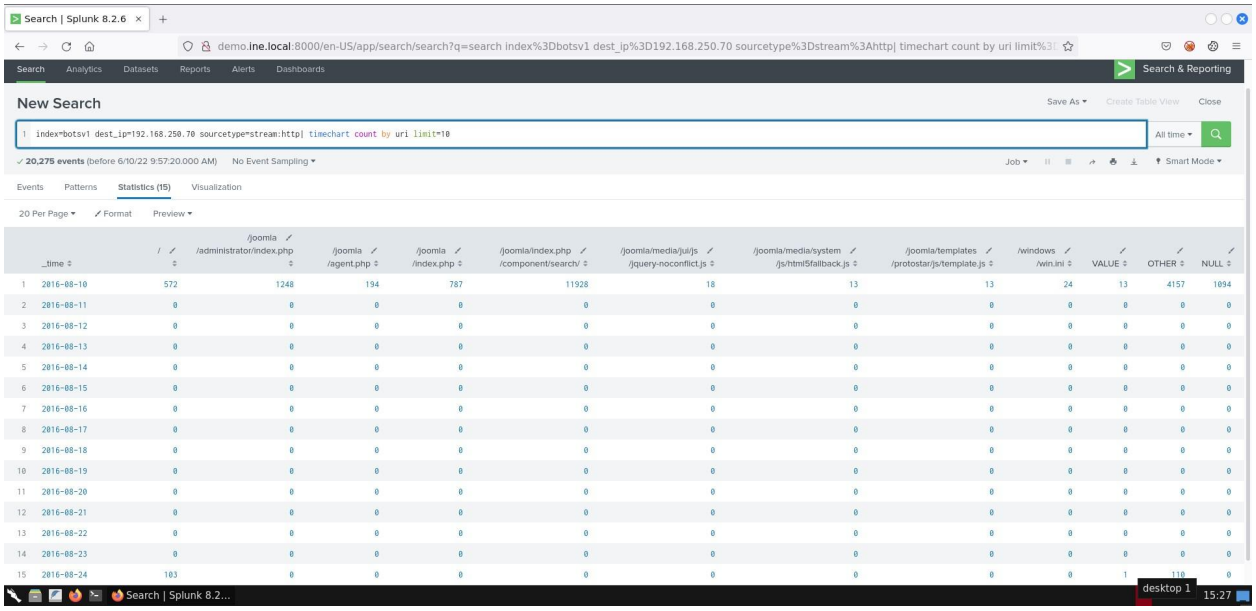
```
index=botsv1 dest_ip=192.168.250.70 sourcetype=stream:http
```

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



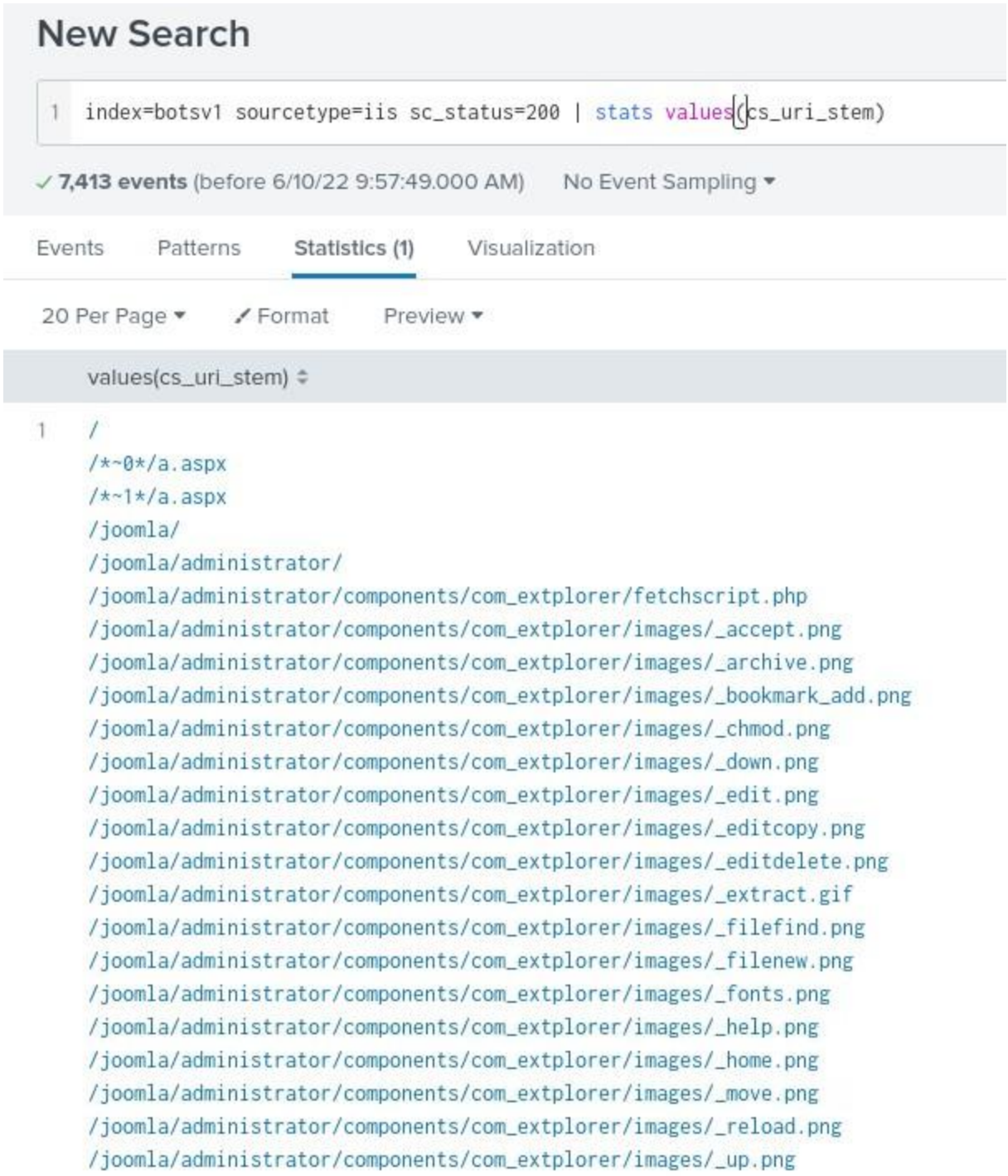
We are also interested in successful page loads. We can identify them, as follows.

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



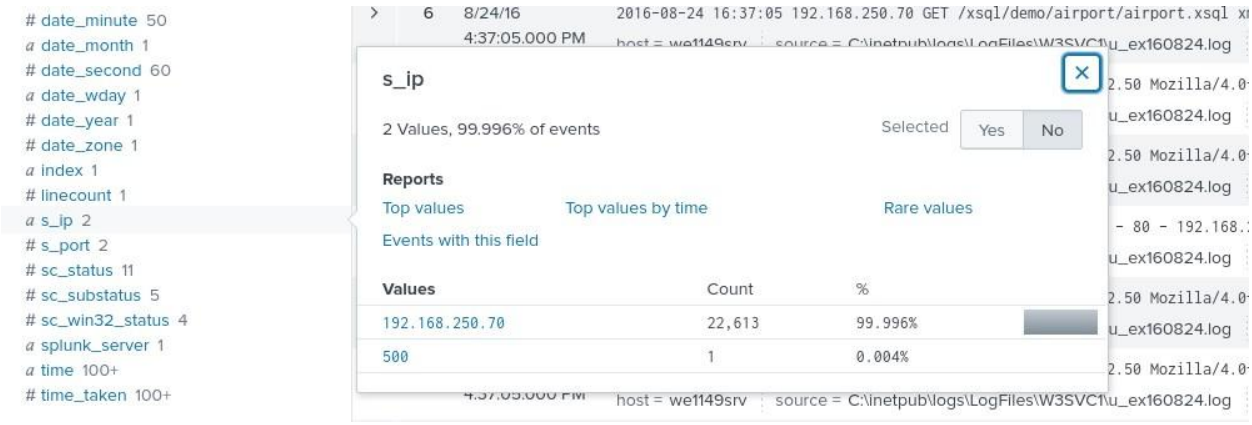
We could have achieved similar results through the **iis** sourcetype, as follows. (This time we are using a transformational search command called **stats** that will allow us to count the number of events grouped by URI.)

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

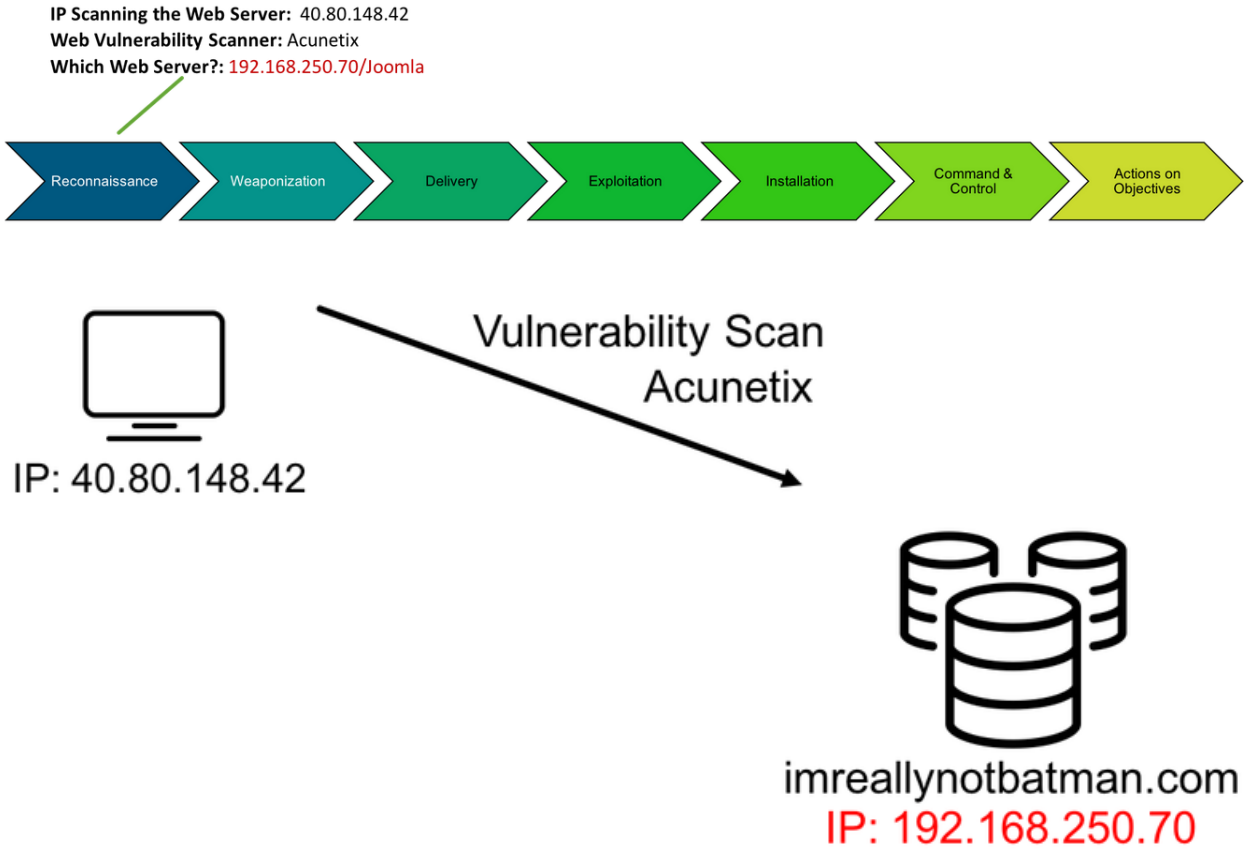


You may be wondering why we aren't specifying **192.168.250.70.** This is because if we submit the below and check the host field, we will find only one host, **we1149srv.** This host's IP address is **192.168.250.70**

index=botsv1 sourcetype=iis



Below are our findings so far.



Task 2: Identify any weaponization activities on your network

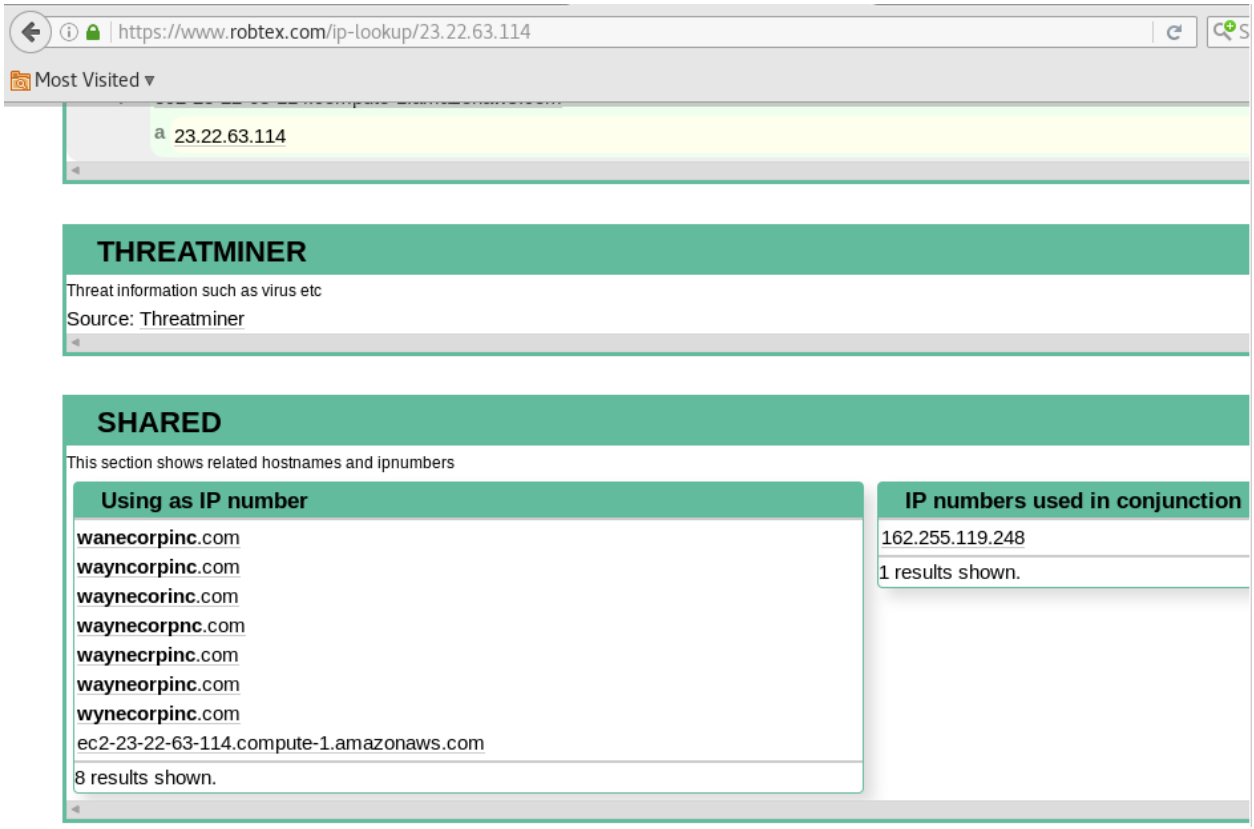
At this point, we need to understand that Splunk is not panacea. During our investigations, not every answer can be

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.



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.

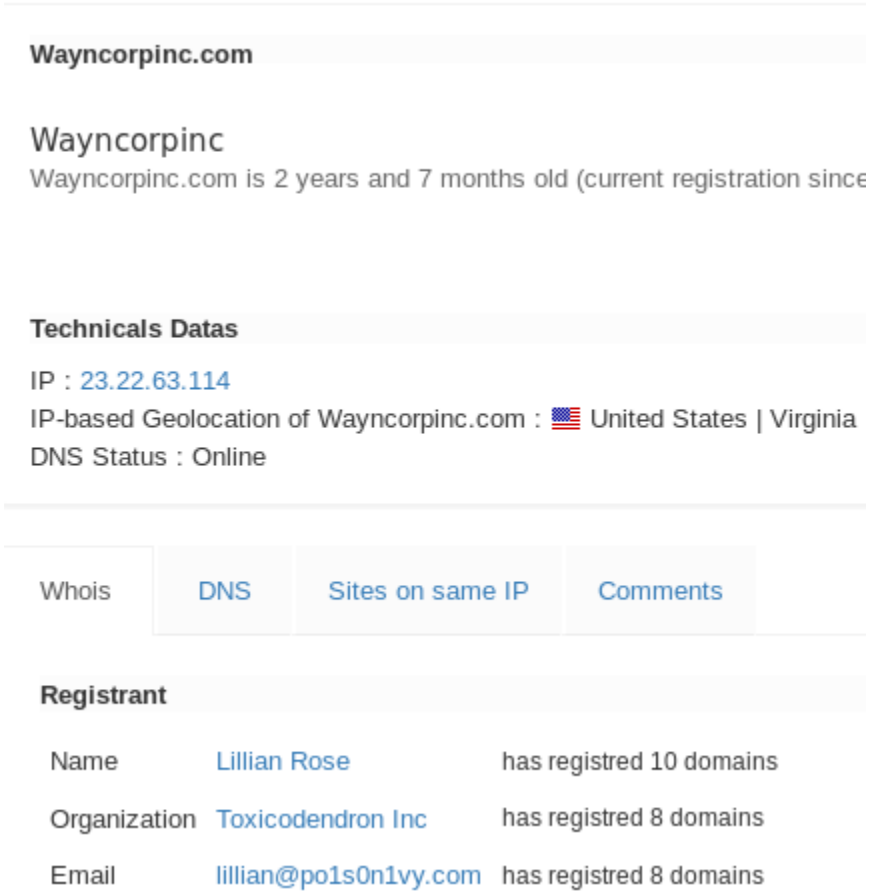
REVERSE DNS

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
po1s0n1vy.com	2018-07-09
www.po1s0n1vy.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.

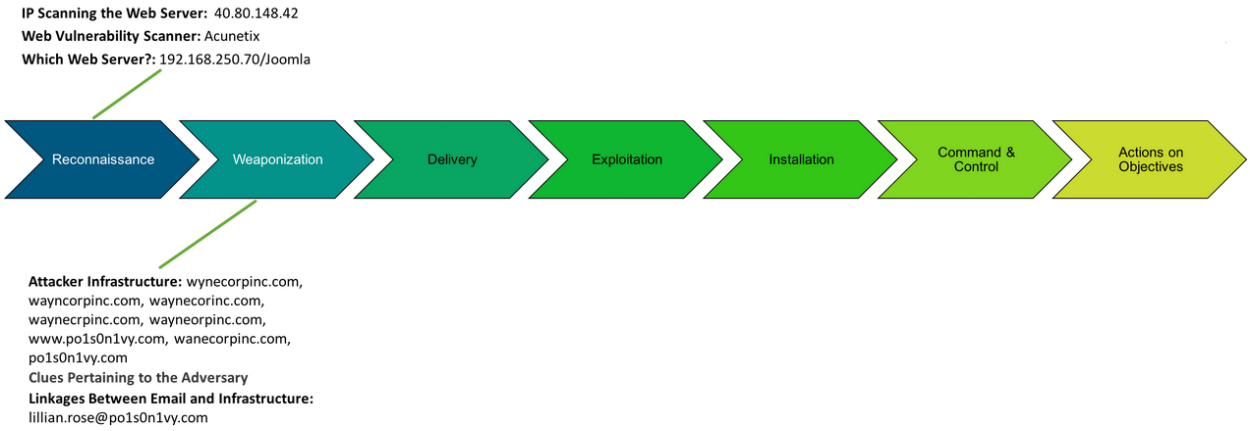
While checking the whois information of **wayncorpinc.com** we come across the following.



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	2016-09-01 09:03:44
	AVG	Agent5.APHV	
	AVware	Trojan.Win32.Generic!BT	
	Ad-Aware	Trojan.GenericKD.3470547	
	AegisLab	Agent5.Aphv.Gen!c	

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.

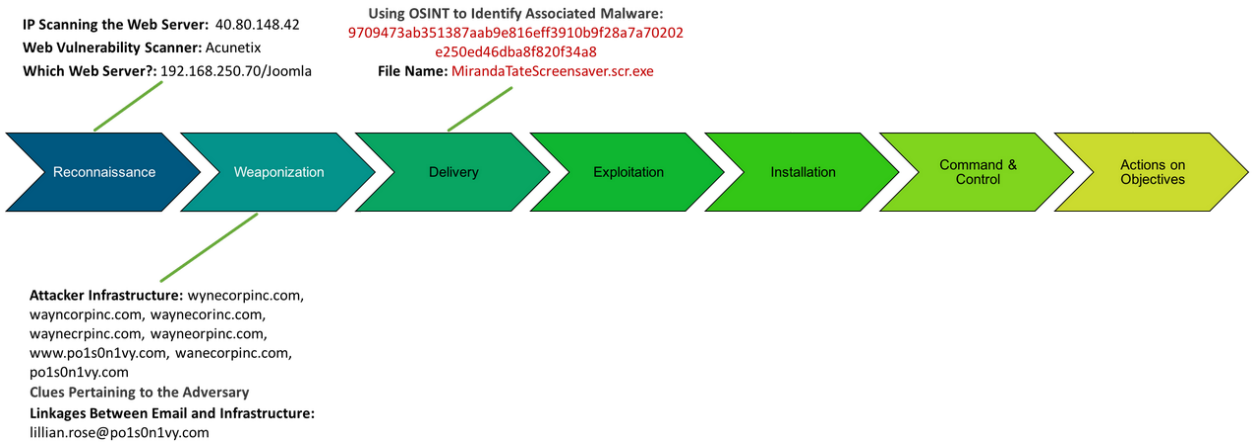
Search for domains, IPs, MD5|SHA1|SHA256, email address or ssl(ssl:), user-agent(ua:), AV family(av:), filename (filename:), URI (uri:), registry (reg:), mutex (mutex:)...

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.

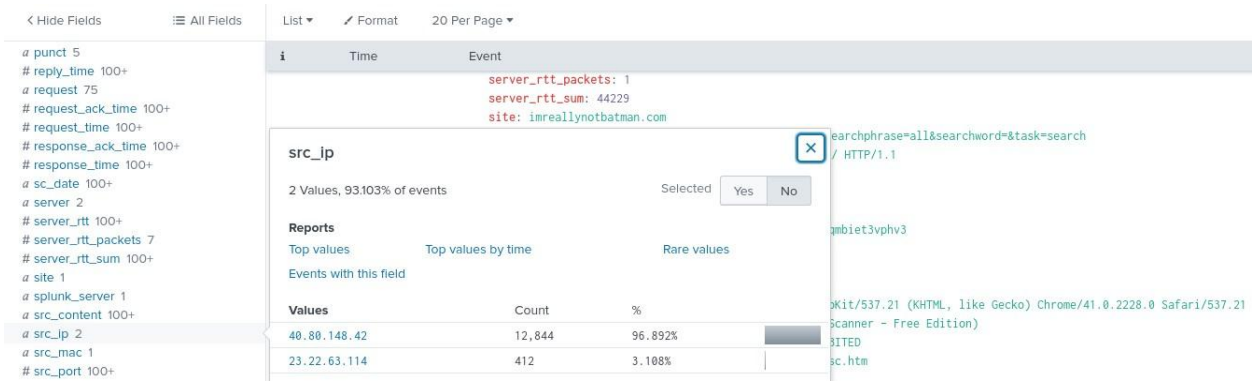


Task 4: Identify any exploitation activities on your network through Splunk searches

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
```

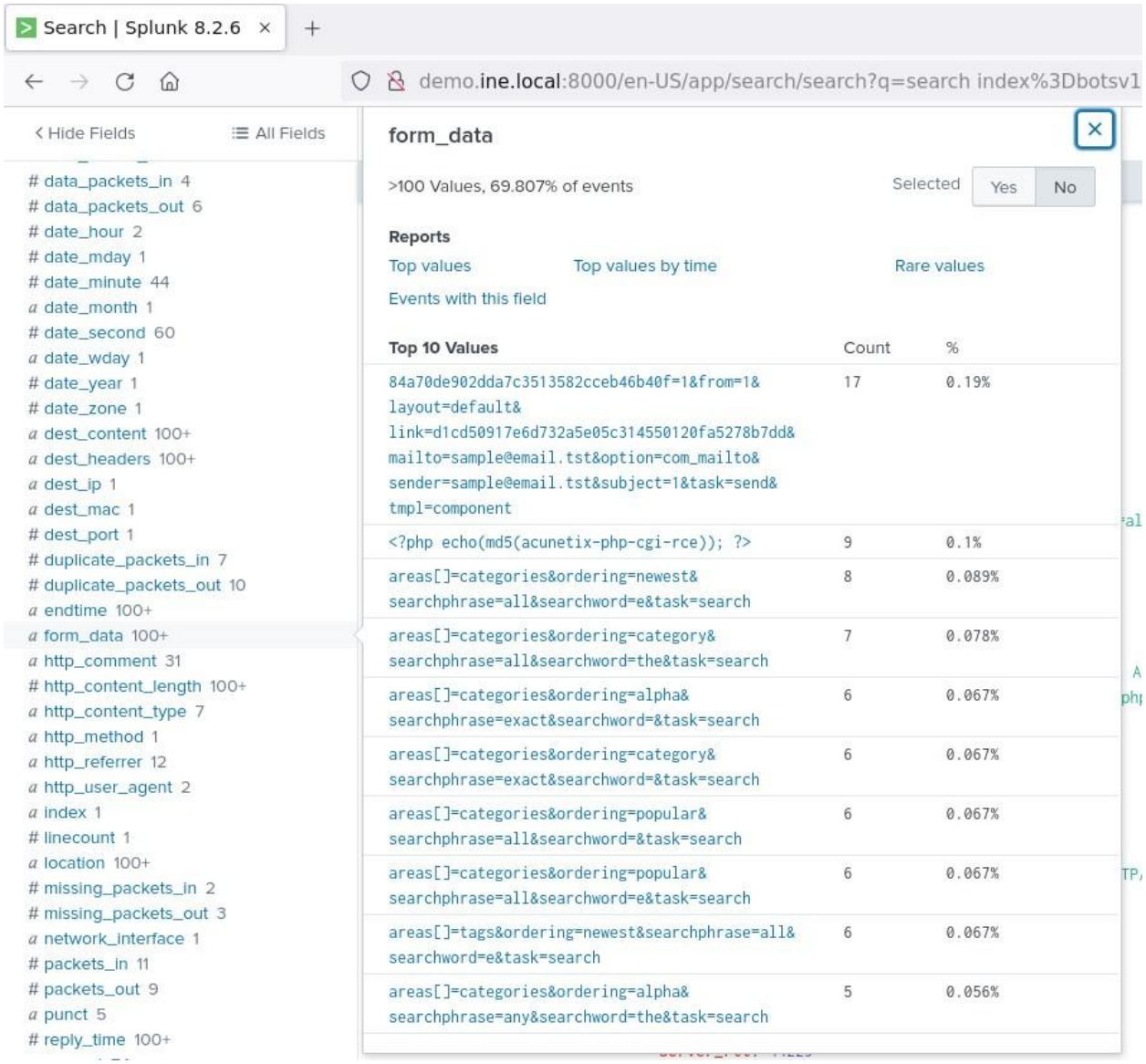
The **src** field contains what we are looking for. We specified that we are interested in POST requests since logins are usually performed through POST requests (more on that in a bit).



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

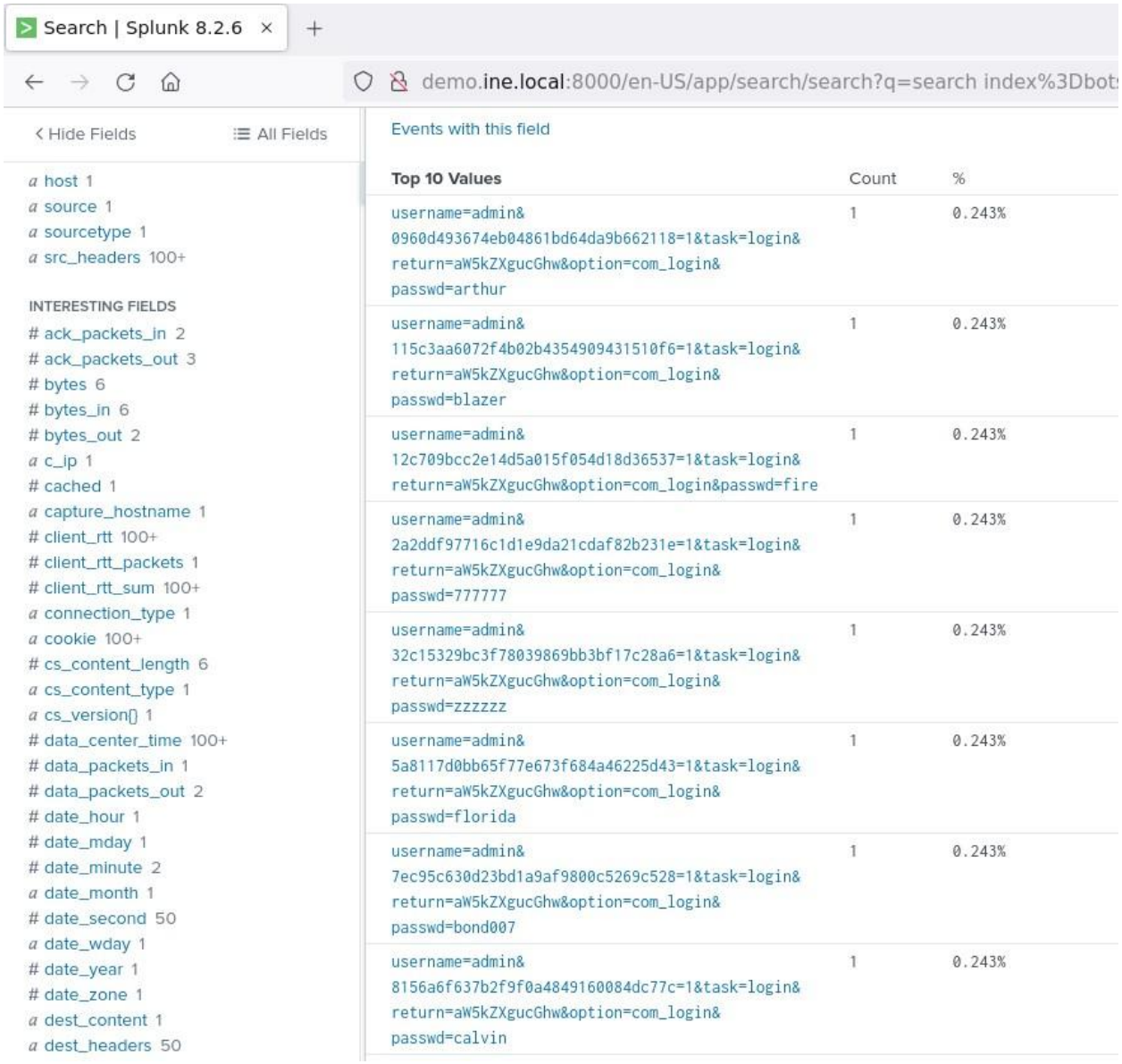
```
index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70" http_method=POST src_ip="40.80.148.42"
```

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



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

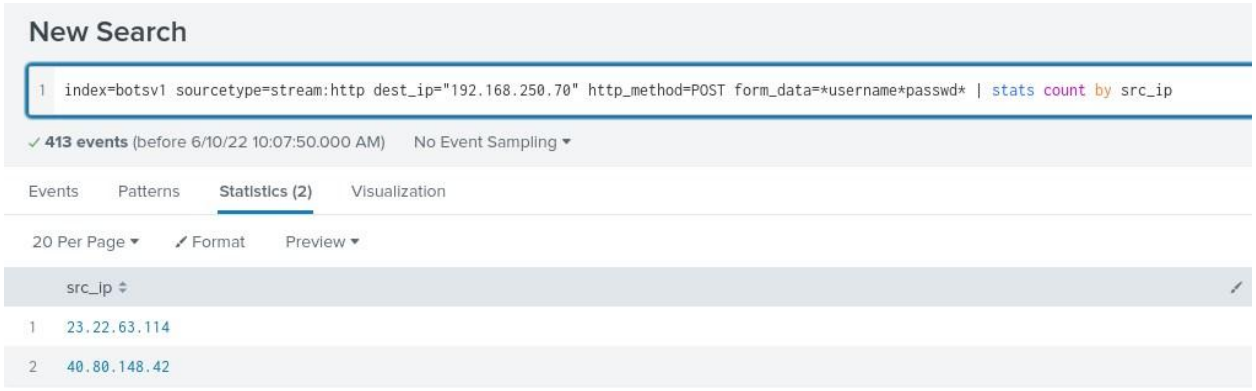
```
index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70" http_method=POST src_ip="23.22.63.114"
```



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

Let's make sure, as follows.

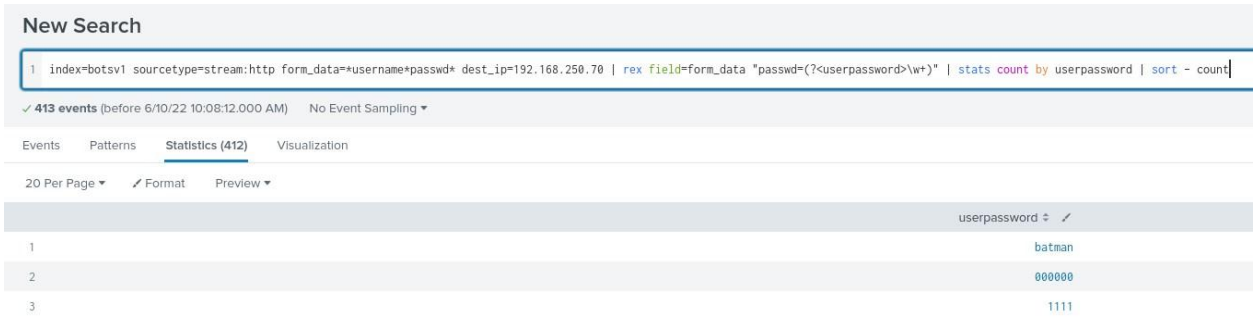
```
index=botsv1 sourcetype=stream:http dest_ip="192.168.250.70" http_method=POST form_data=*username*passwd* | stats count by src_ip
```



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
```



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.

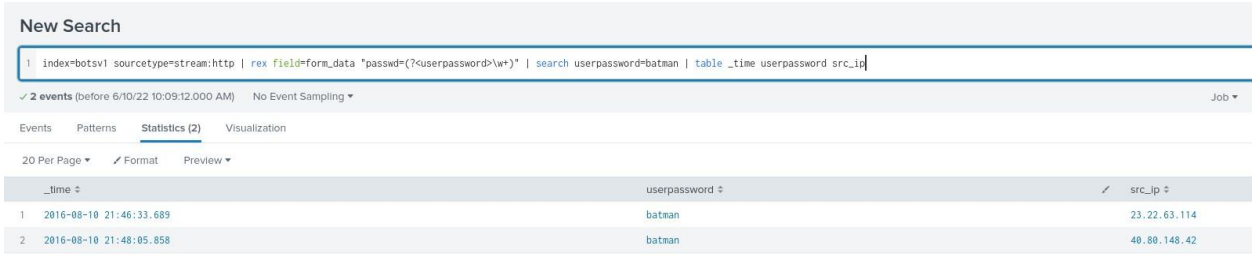
If we want to get an idea of the time of the compromise and the URI that was targeted, we can do that as follows.

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

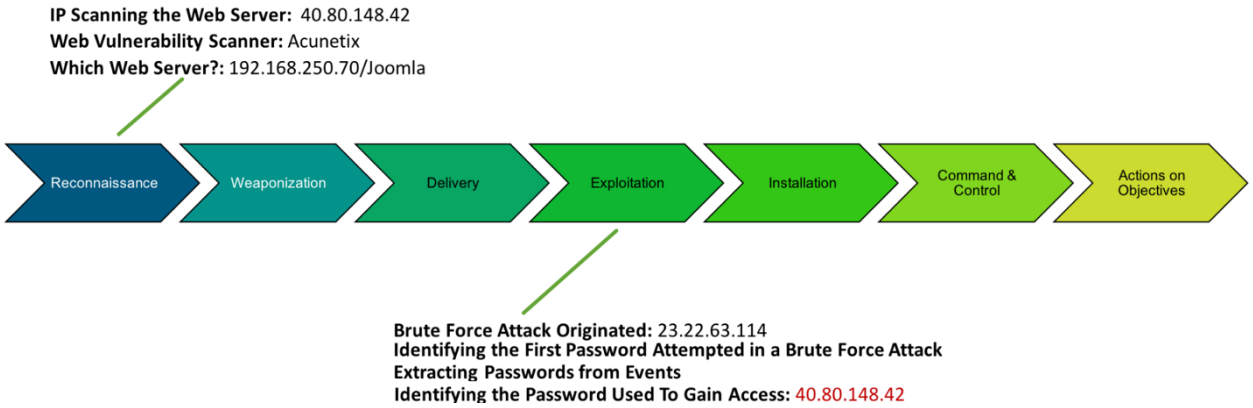
```
src_ip=40.80.148.42 | rex field=form_data
"passwd=(?<userpassword>\w+)"| search userpassword=* | table
_time uri userpassword
```

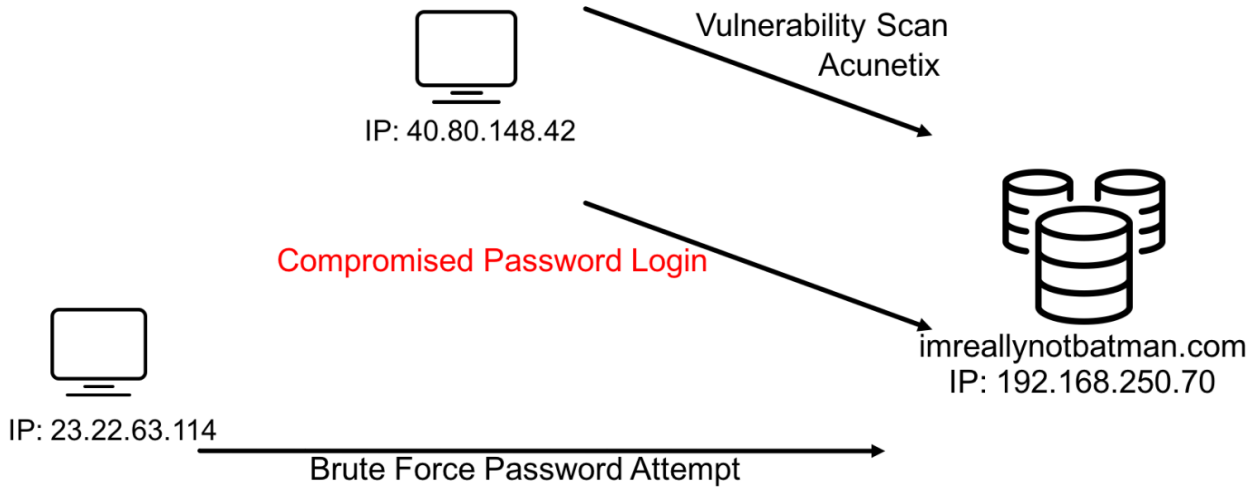
Finally, if we want to view the two successful logins we can do so, as follows.

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



Below are our findings so far.





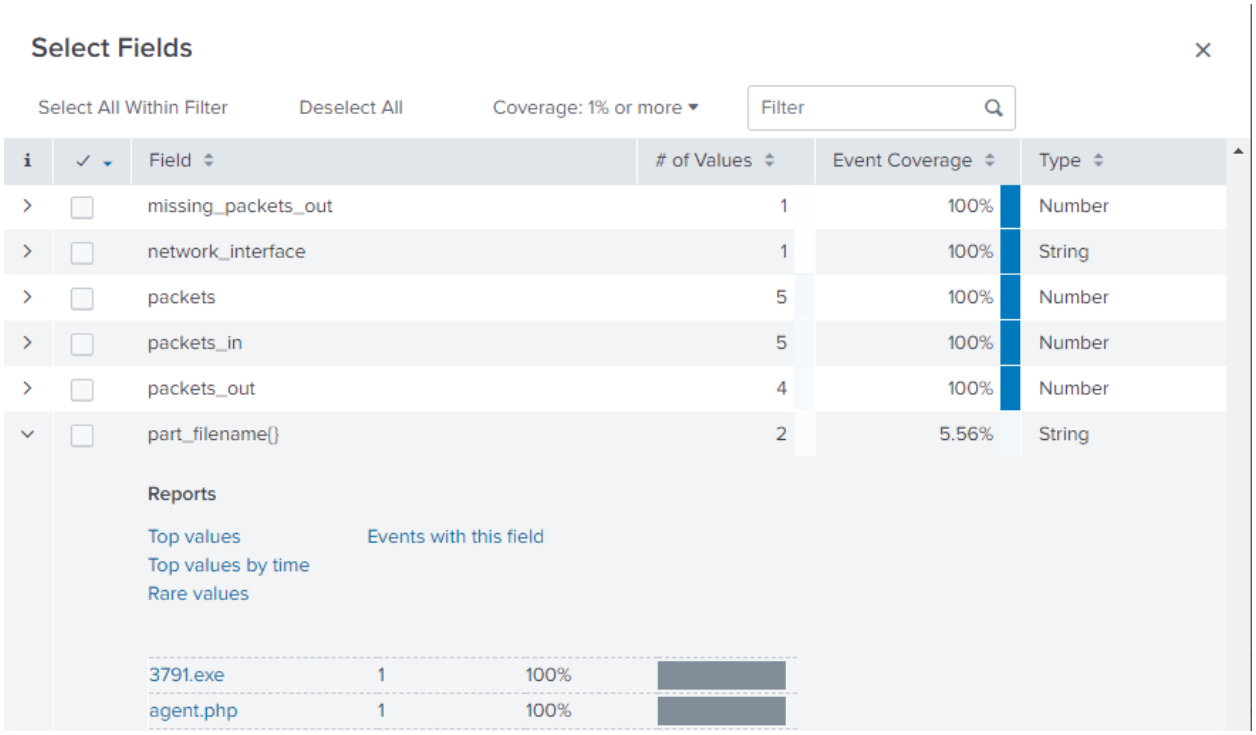
Task 5: Identify any installation activities on your network through Splunk searches

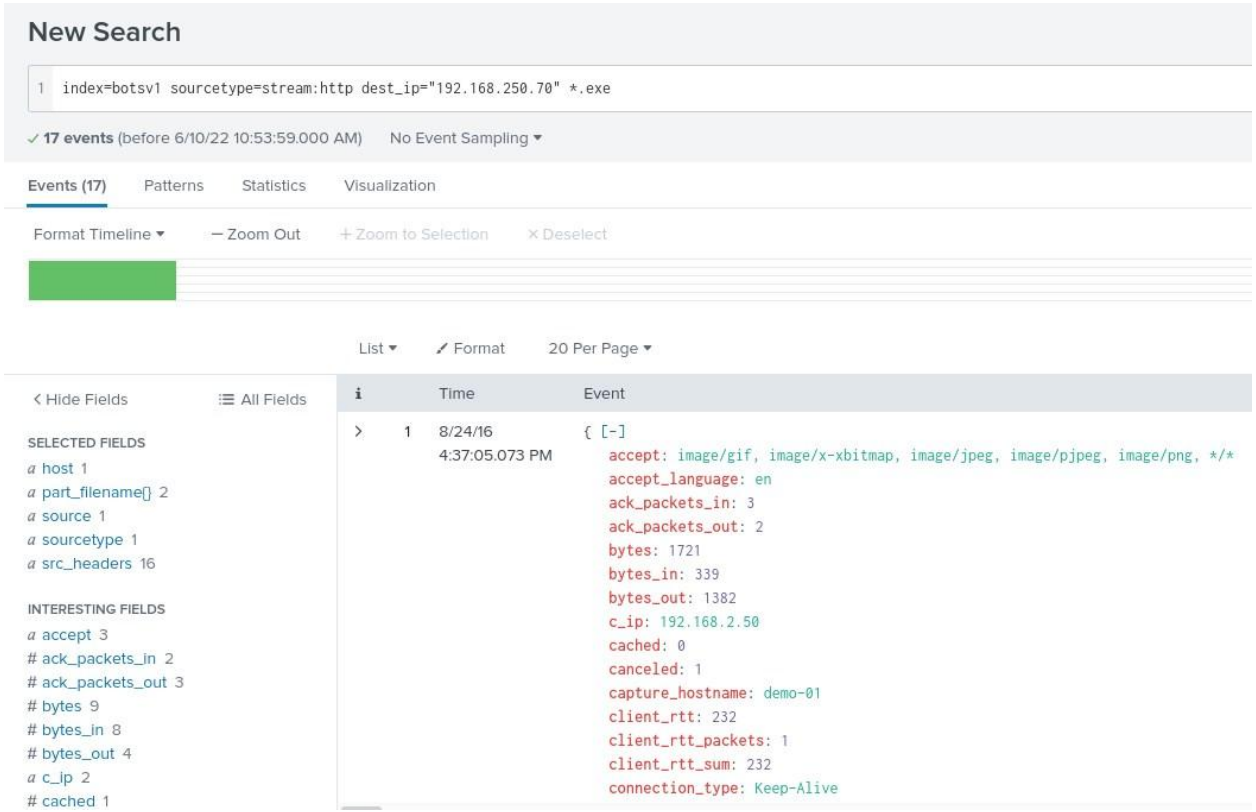
As far as the installation phase of the cyber kill chain is concerned, we are mostly interested in identifying any malware being uploaded.

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

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

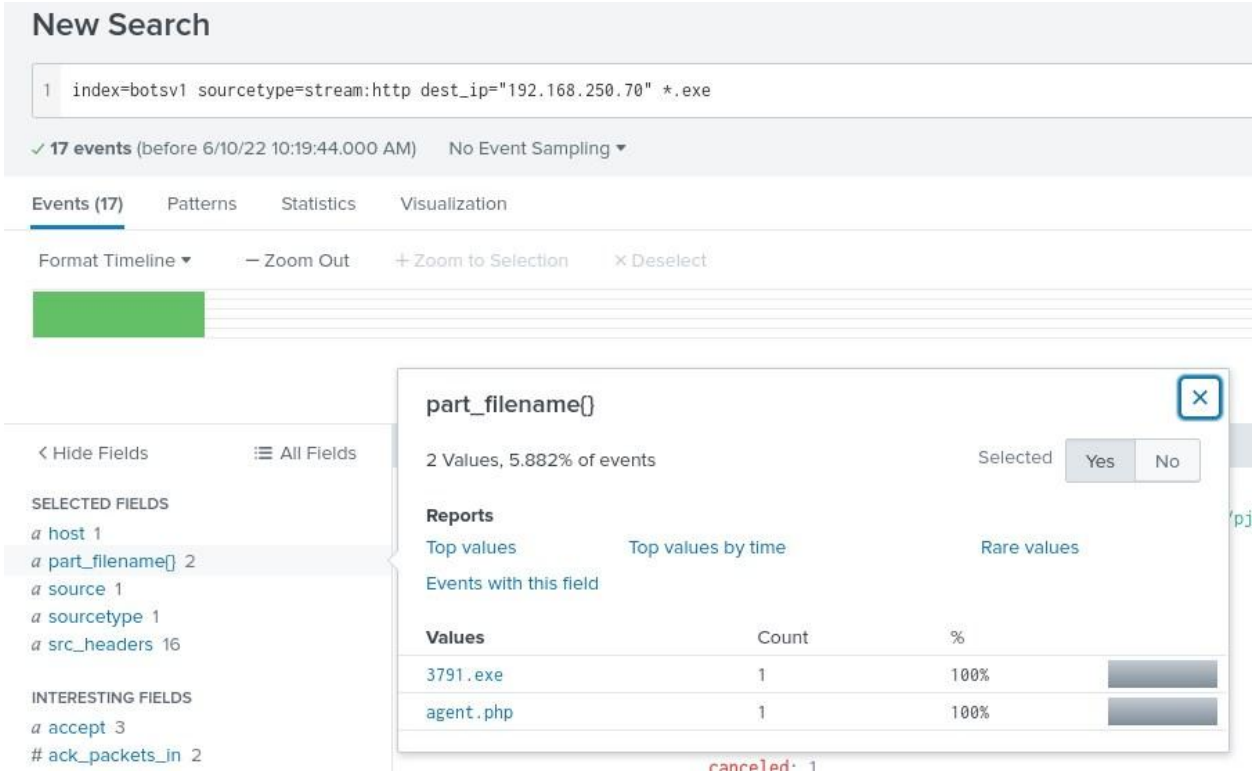
The **part_filename{}** field contains the information we want to check. It won't be visible by default, so add it.






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

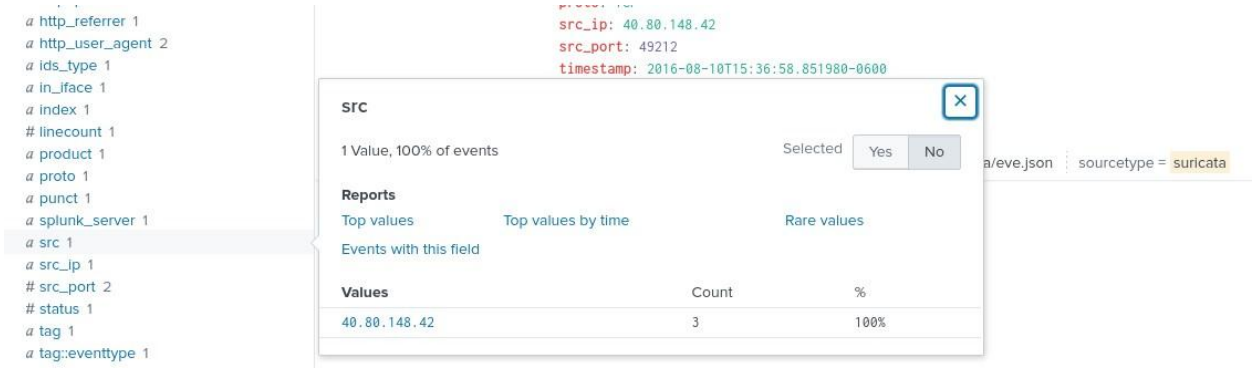
The **fileinfo.filename** field contains the information we want to check.



3791.exe must be the uploaded malware.

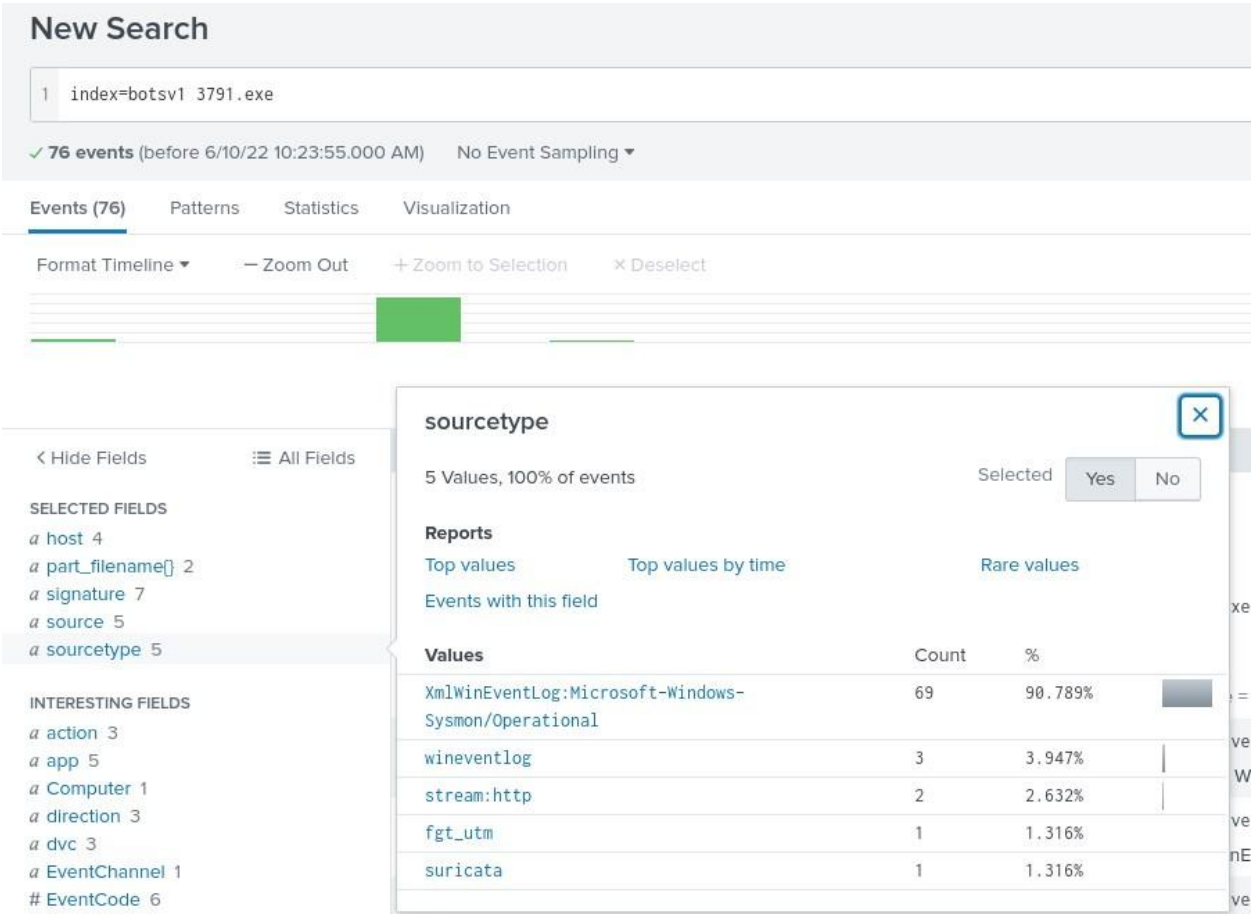
The source from where the file was uploaded can easily be identified, as follows.

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



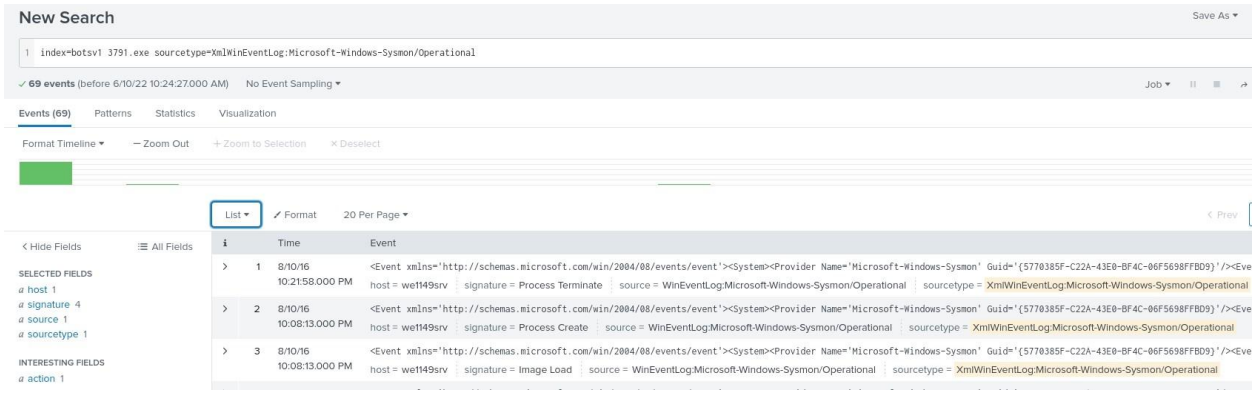
It would be great if we could also identify the hash of the uploaded file. But what sourcetype should we use? Let's find out, as follows.

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



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

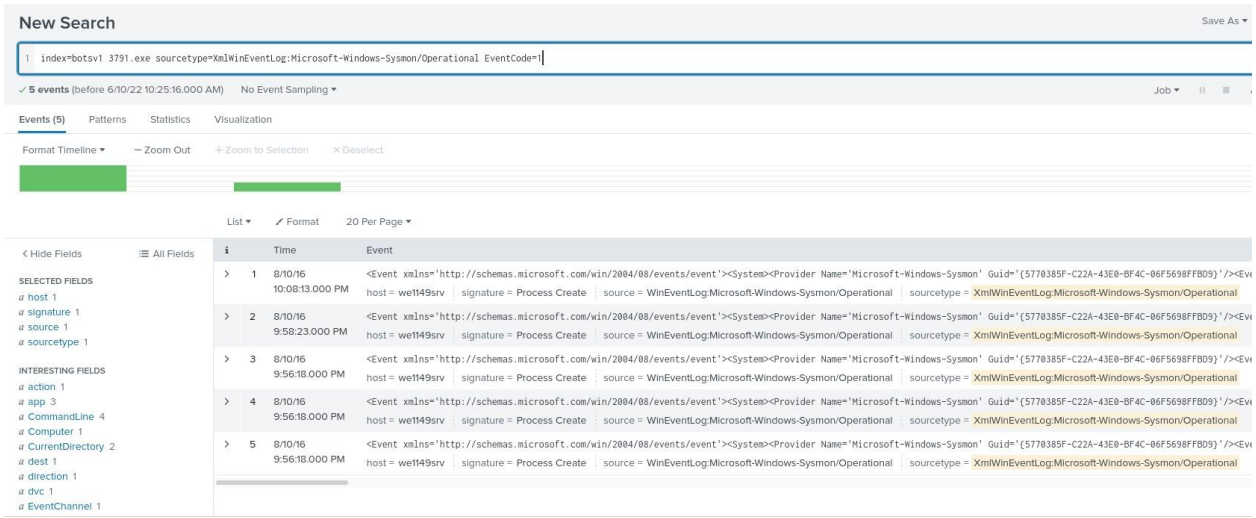
```
index=botsv1 3791.exe
sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational
```



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.

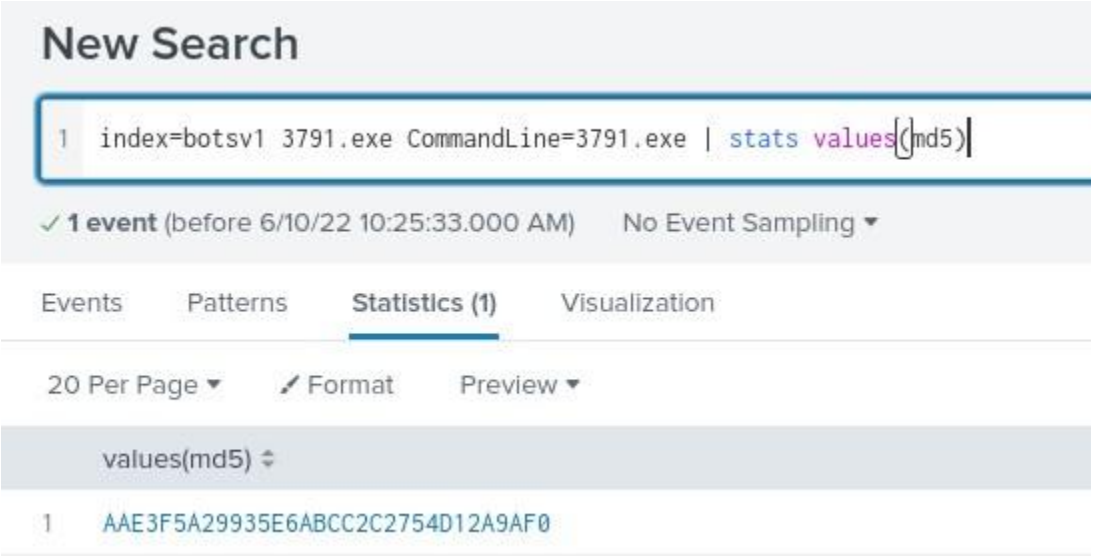
index=botsv1 3791.exe sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational EventCode=1



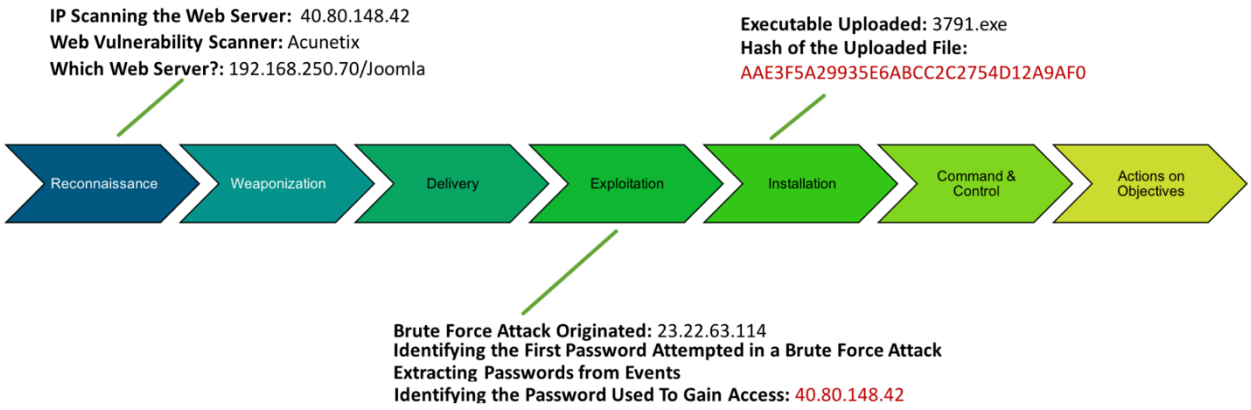
The search above includes EventCode 1 since this EventCode is related to process creation events. Unfortunately, to get the MD5 hash of the uploaded file, we need to narrow things down even further. Specifically, we will need to search for 3791.exe inside the command line field, since this field captures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

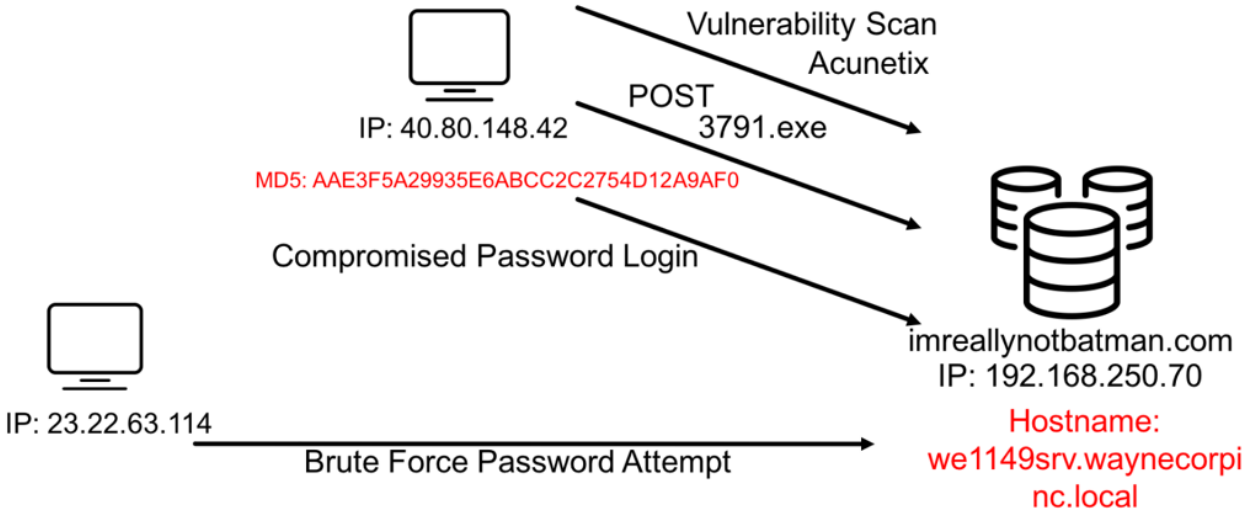
So, the final search is the following.

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



Below are our findings so far.



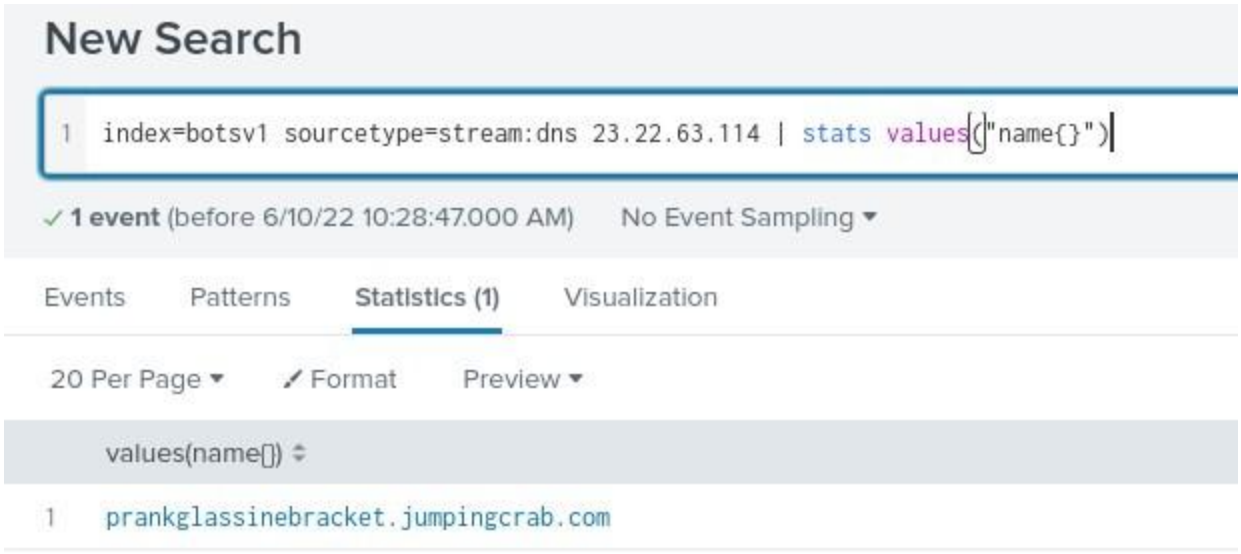


Task 6: Identify any command and control-related activities on your network through Splunk searches

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{}")
```



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

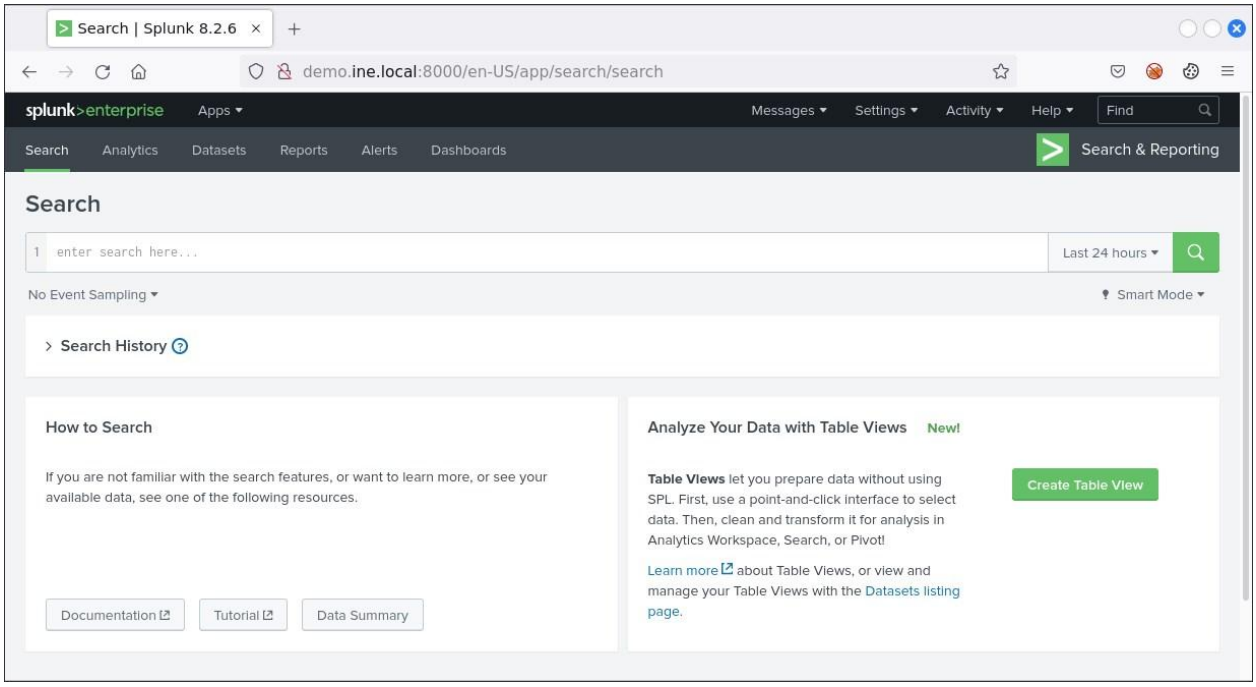
SOLUTIONS

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.

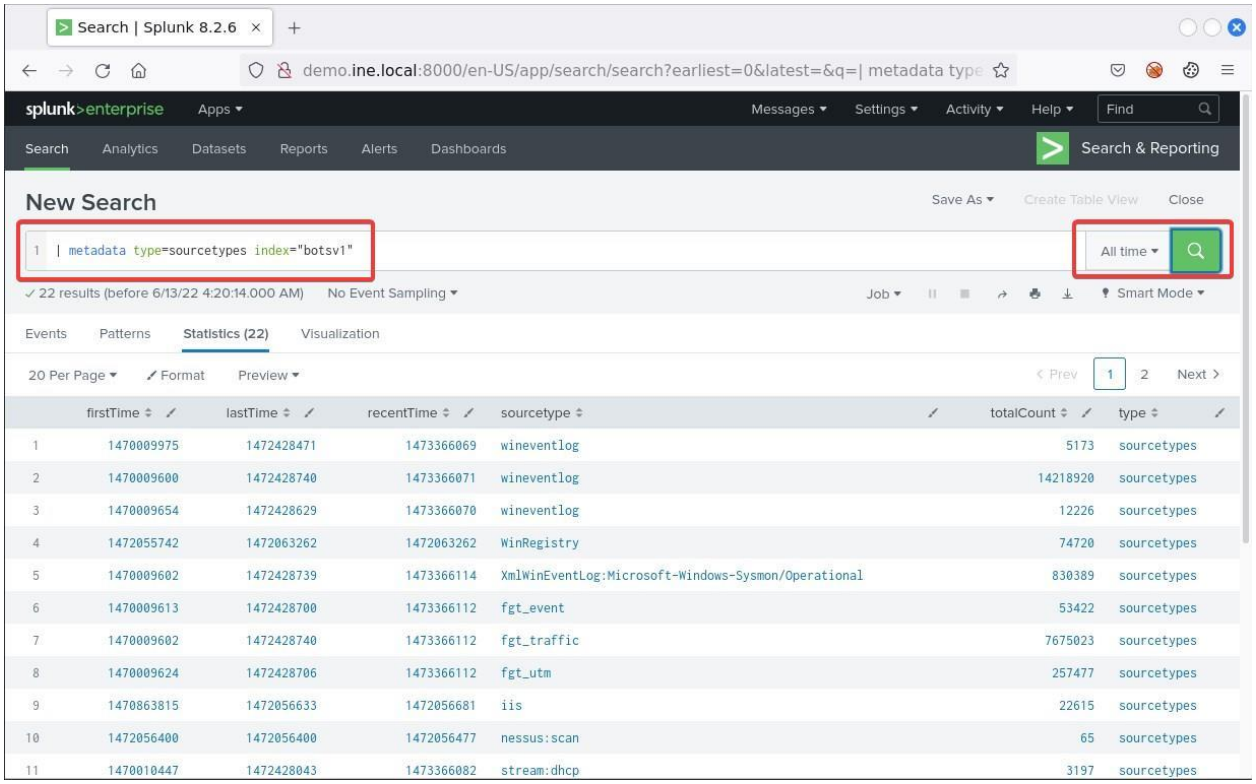


Before starting your investigation change the time range picker to **All time**.

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.



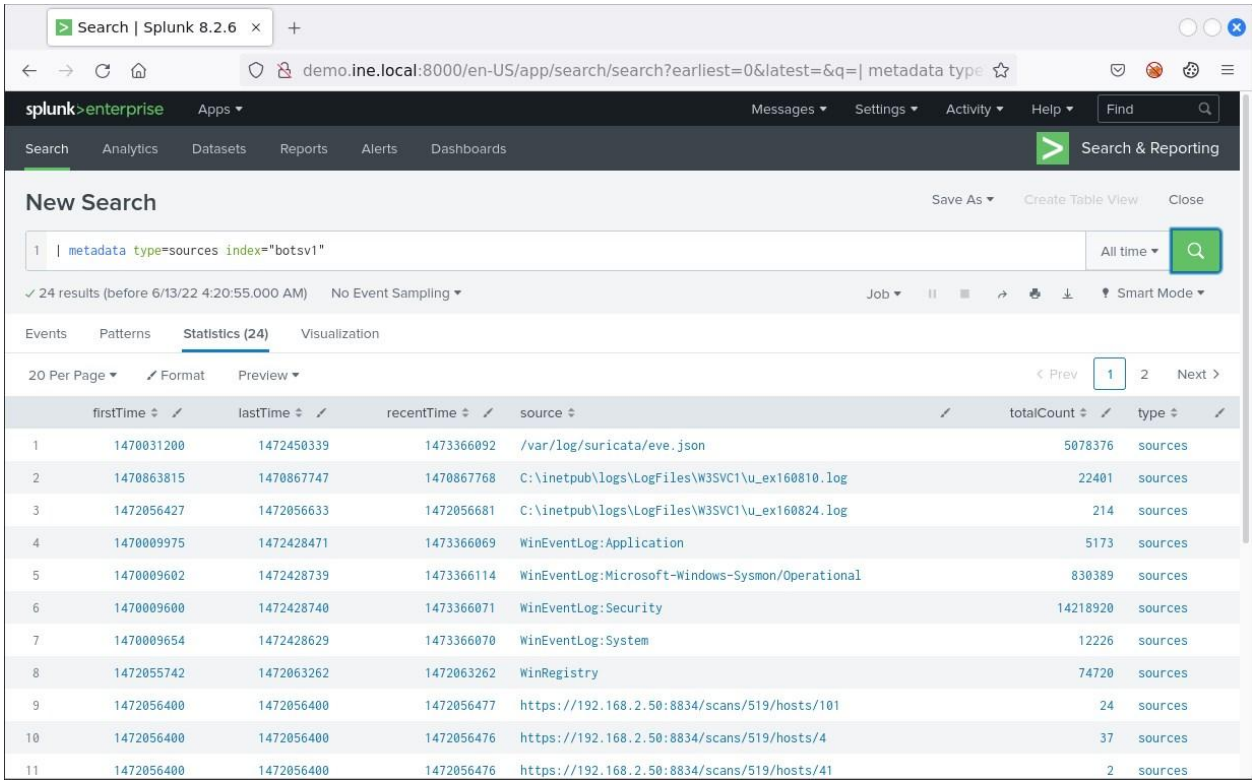
As you can see, Splunk has ingested Windows event logs, Sysmon logs, Fortigate UTM logs, Suricata logs etc.

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.

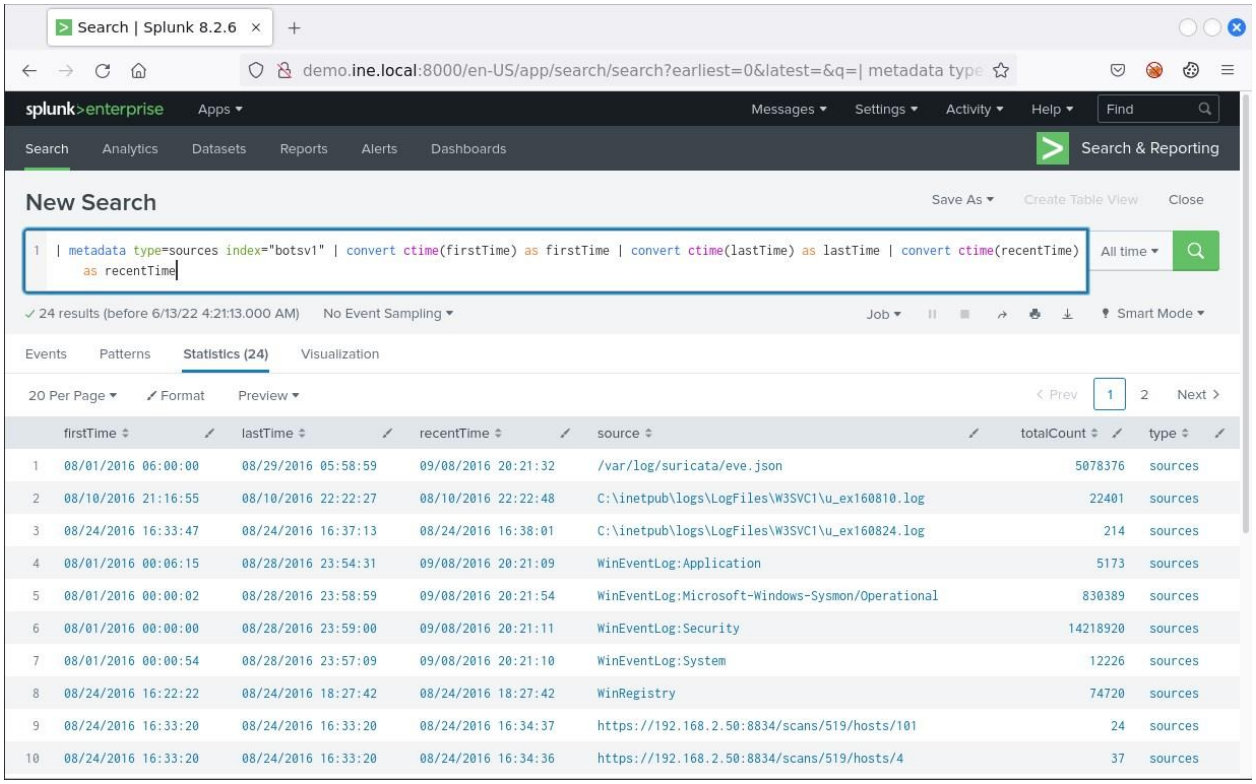


The results between the last two searches are the same. The second search will provide you with a little more detail about the available sourcetypes.

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
```

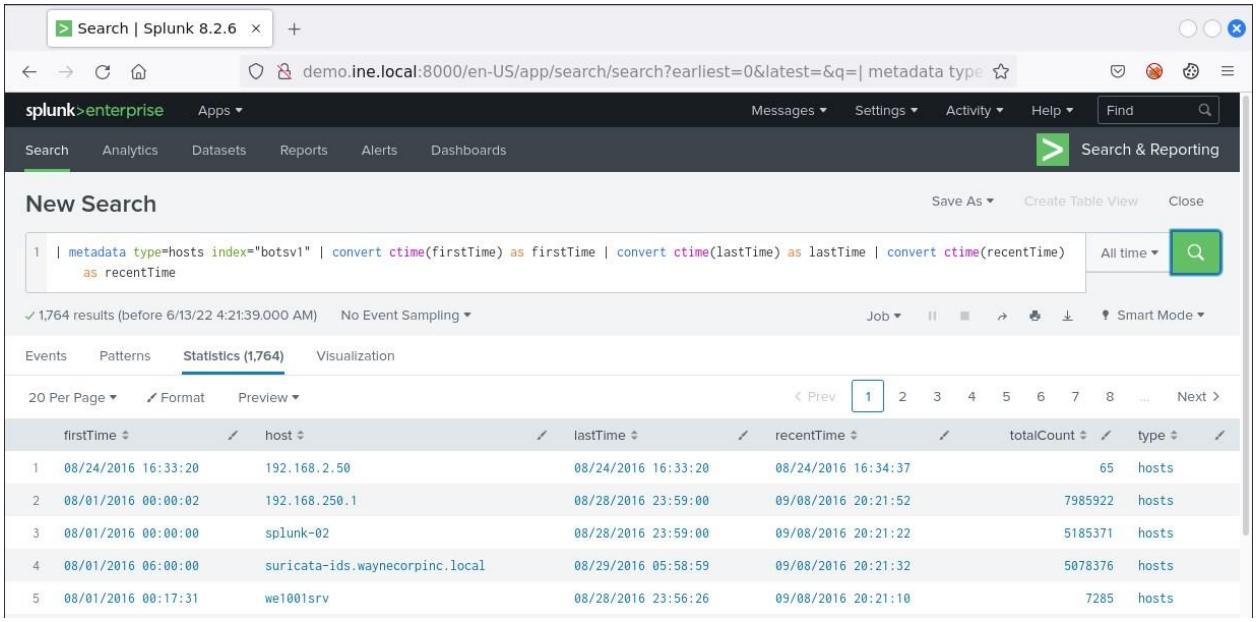
You should see the following.



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.

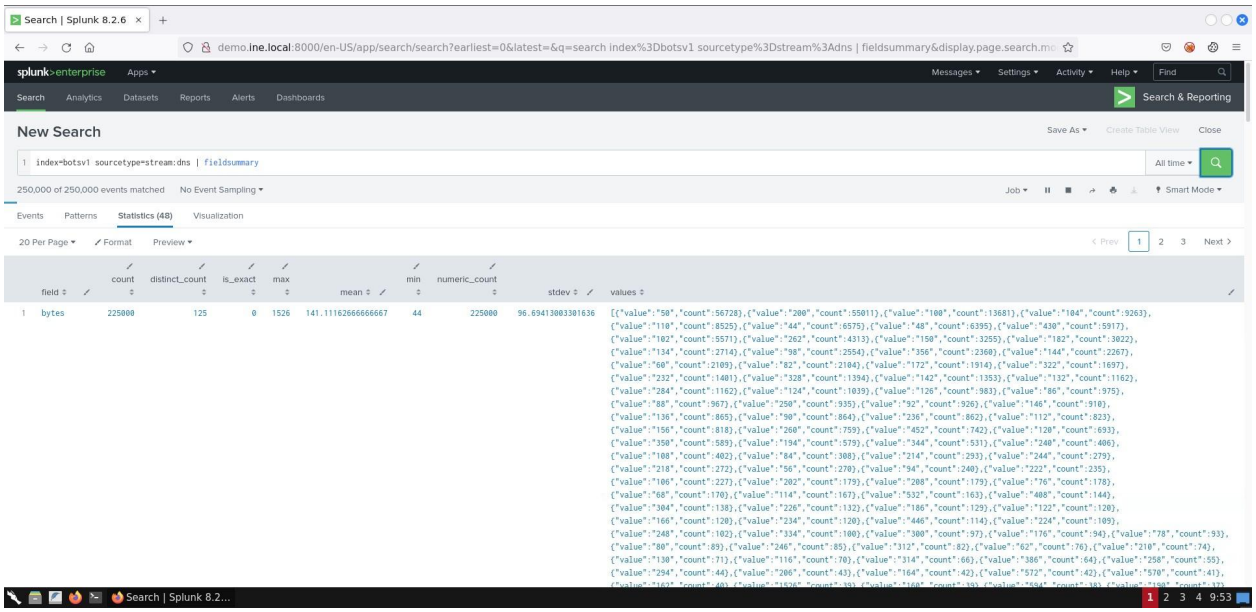
```
| metadata type=hosts index="botsv1" | convert ctime(firstTime) as firstTime | convert ctime(lastTime) as lastTime | convert ctime(recentTime) as recentTime
```

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



A great sourcetype to start with is **stream:dns**.

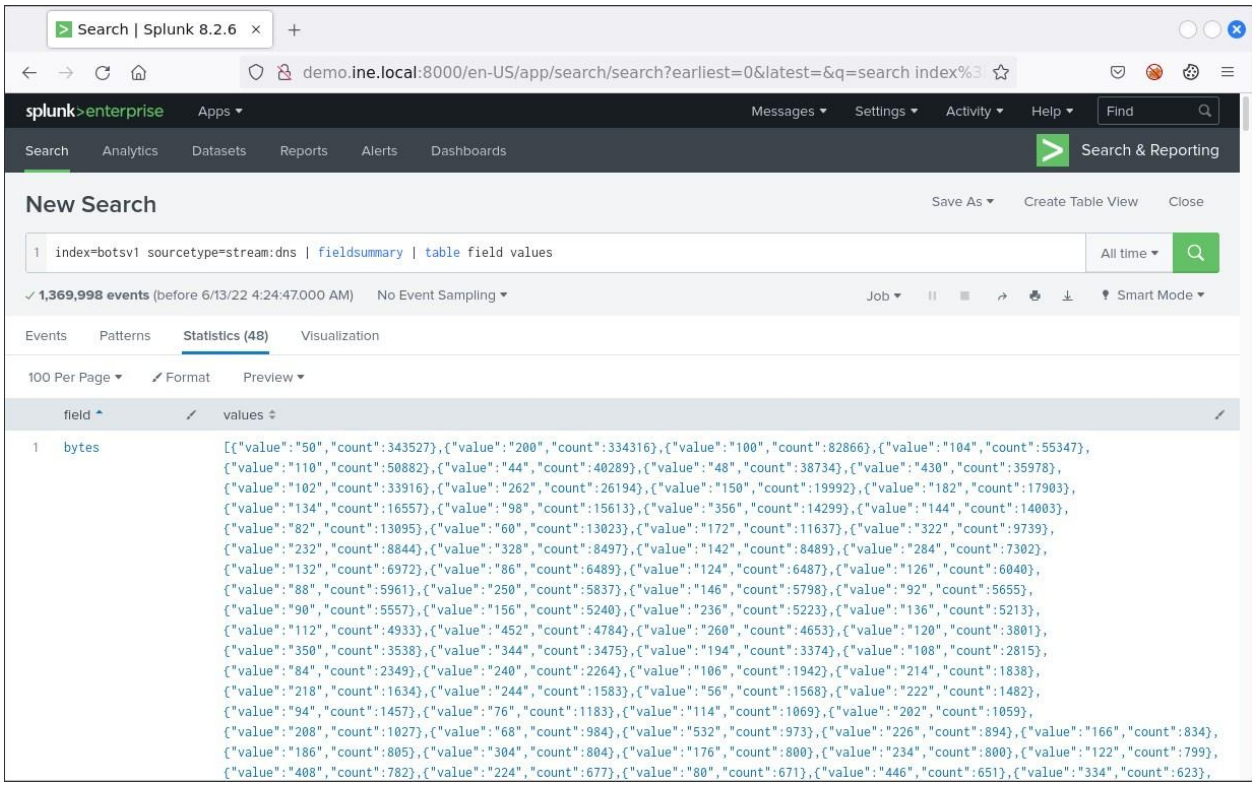
```
index=botsv1 sourcetype=stream:dns | fieldsummary
```



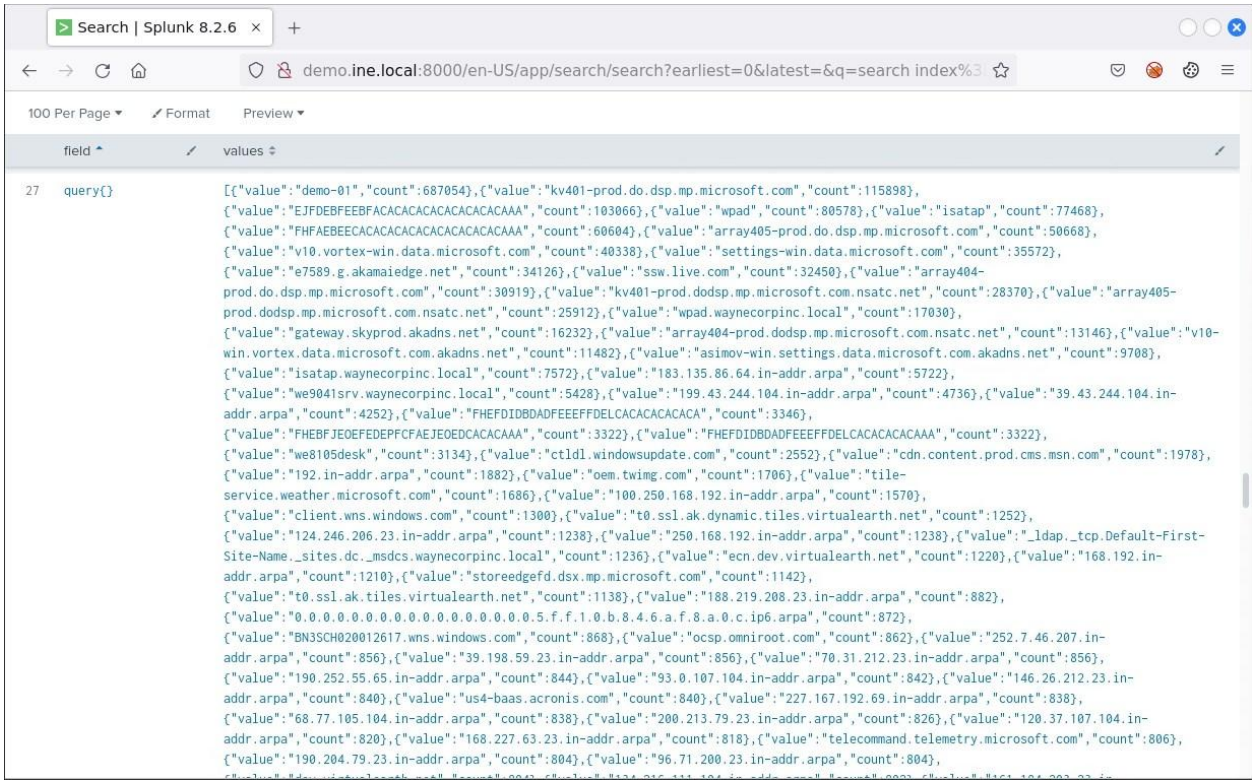
The results of the search above may be difficult to read, so create a table that will contain **field** and **values** entries only. You can that by submitting the following search.

```
index=botsv1 sourcetype=stream:dns | fieldsummary | table field values
```

You should see something similar to the below.



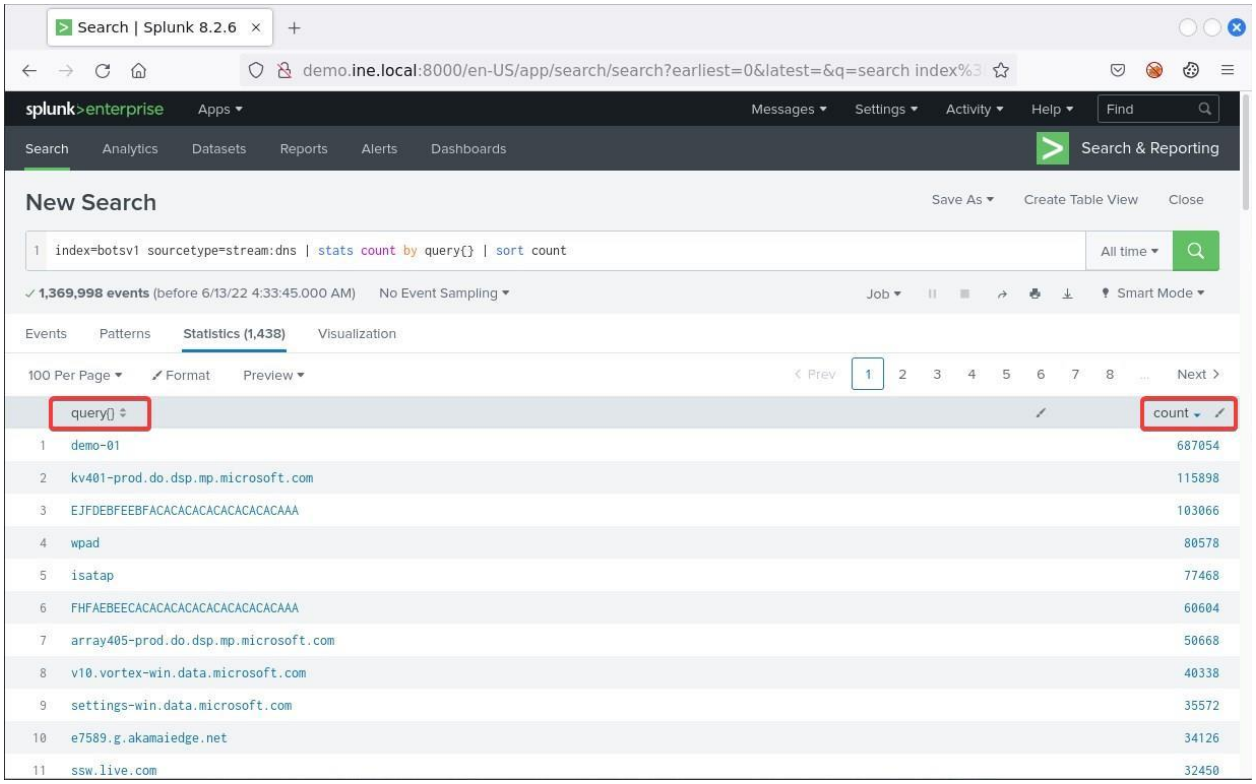
You now need to determine which of the available fields is more important. **dest** could provide you with useful information, but the most interesting field in these results is **query{}**, since it can provide you with information related to interactions with remote (and possibly malicious) servers.



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.

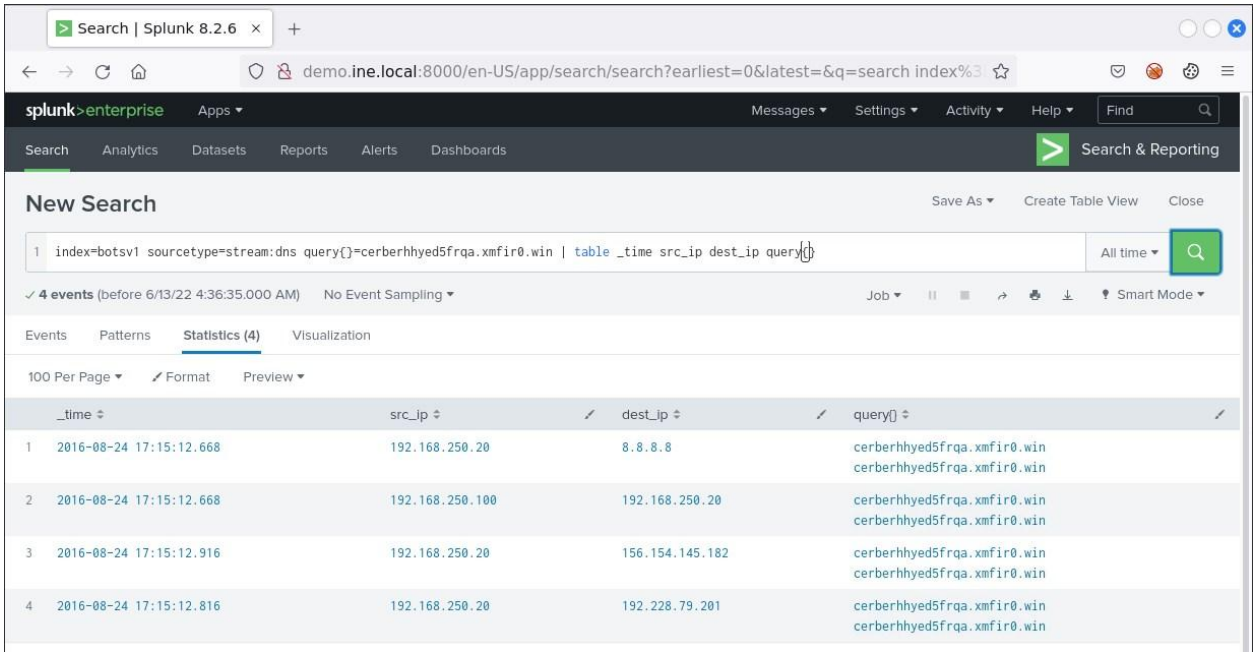


Going through all queries you will spot some curious-looking domain names. Such a domain name is **cerberhhyed5frqa.xmfir0.win**

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.

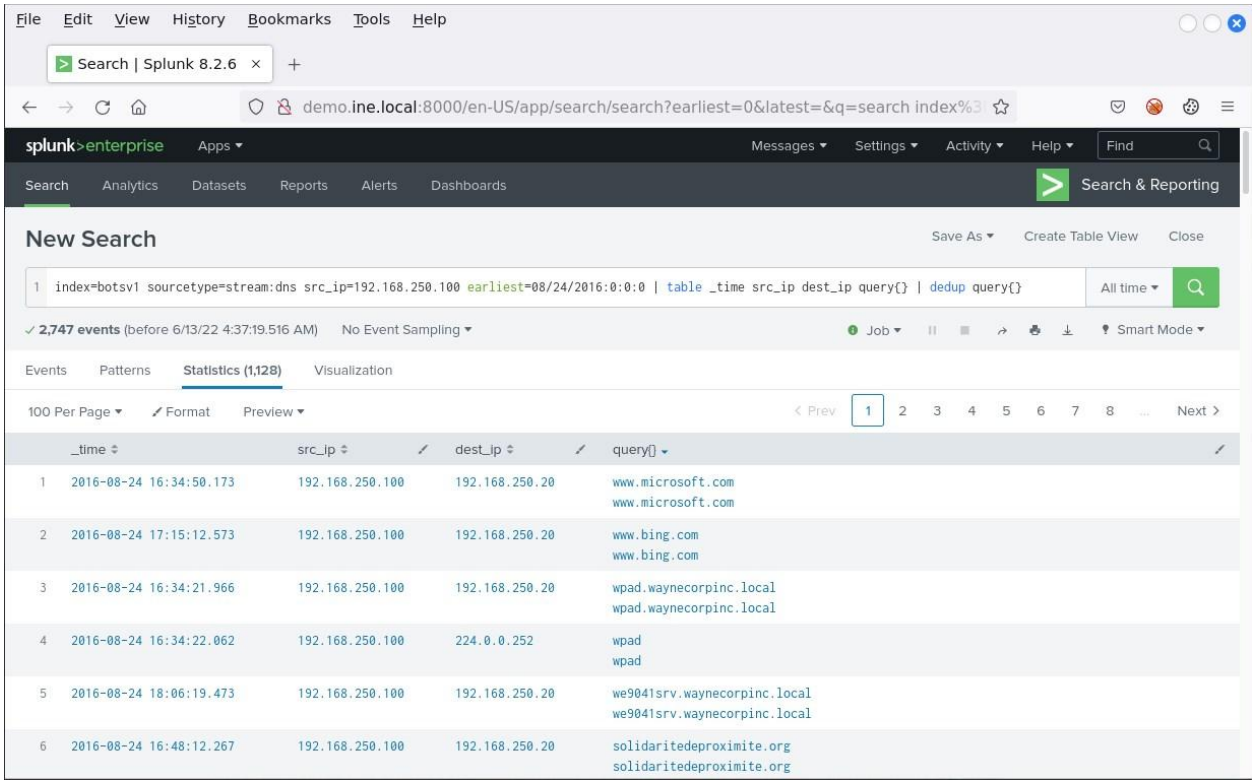


In the results (sorted by time) above you can see the **192.168.250.100** host making a DNS query to **192.168.250.20**. **192.168.250.20** in turn makes a number of external DNS queries. From this behavior you can assume that **192.168.250.20** is a DNS server and **192.168.250.100** is probably a compromised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

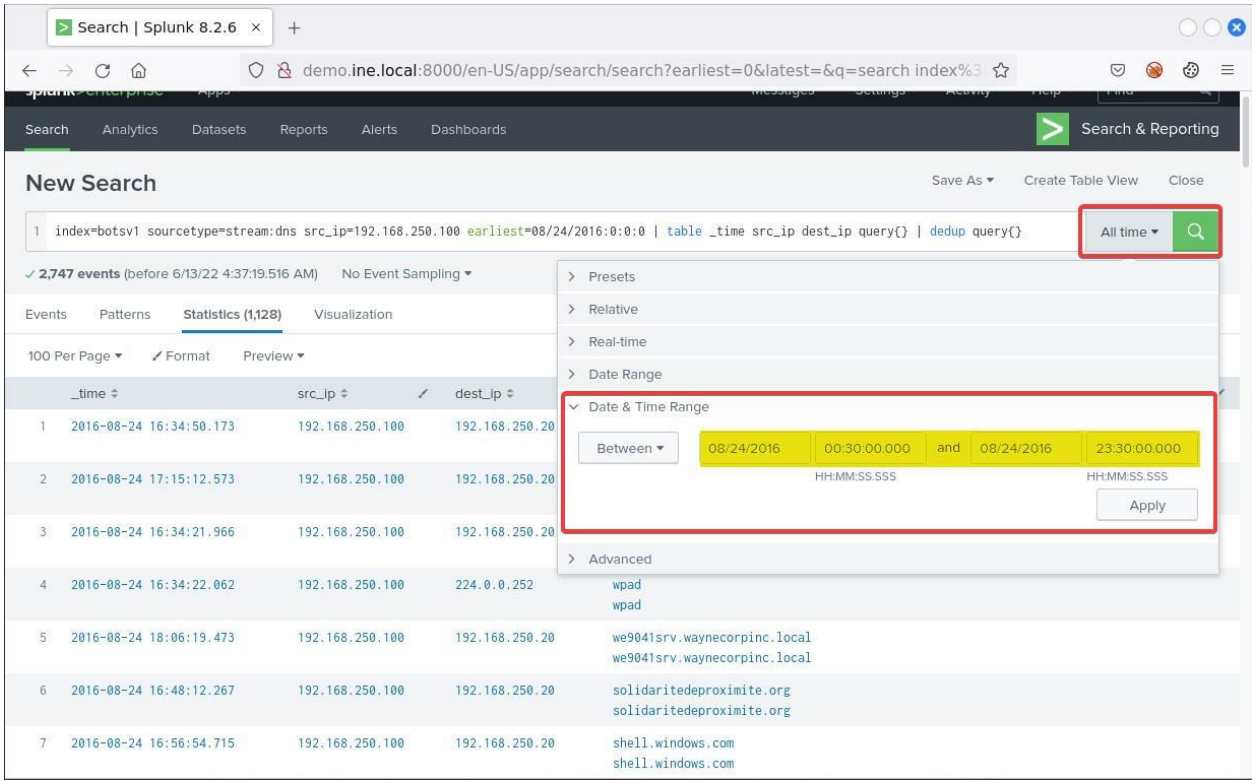


Notice that the earliest events are located at the bottom of the table.

The possibly compromised **192.168.250.100** system is looking for *isatap* and *wpad* right after visiting the curious-looking **cerberhhyed5frqa.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.

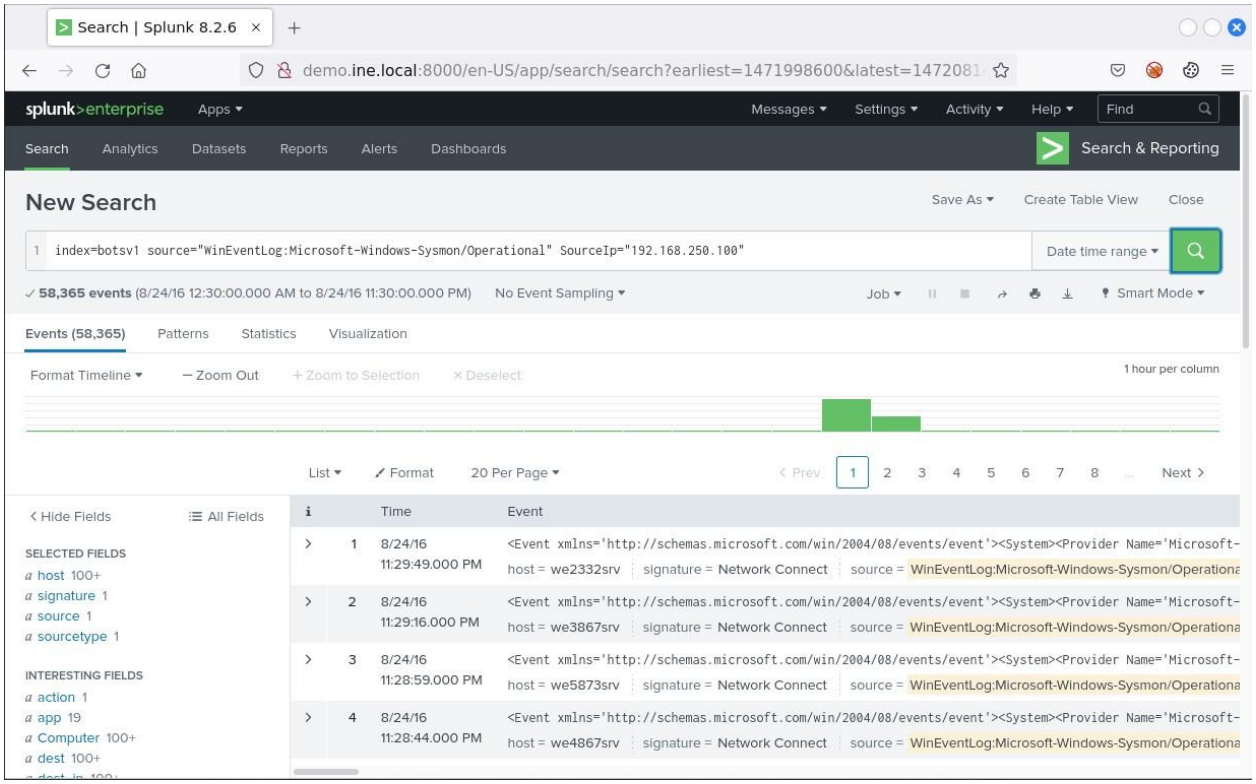
First, change the time range picker as follows and click **Apply**.



Then, submit the following search.

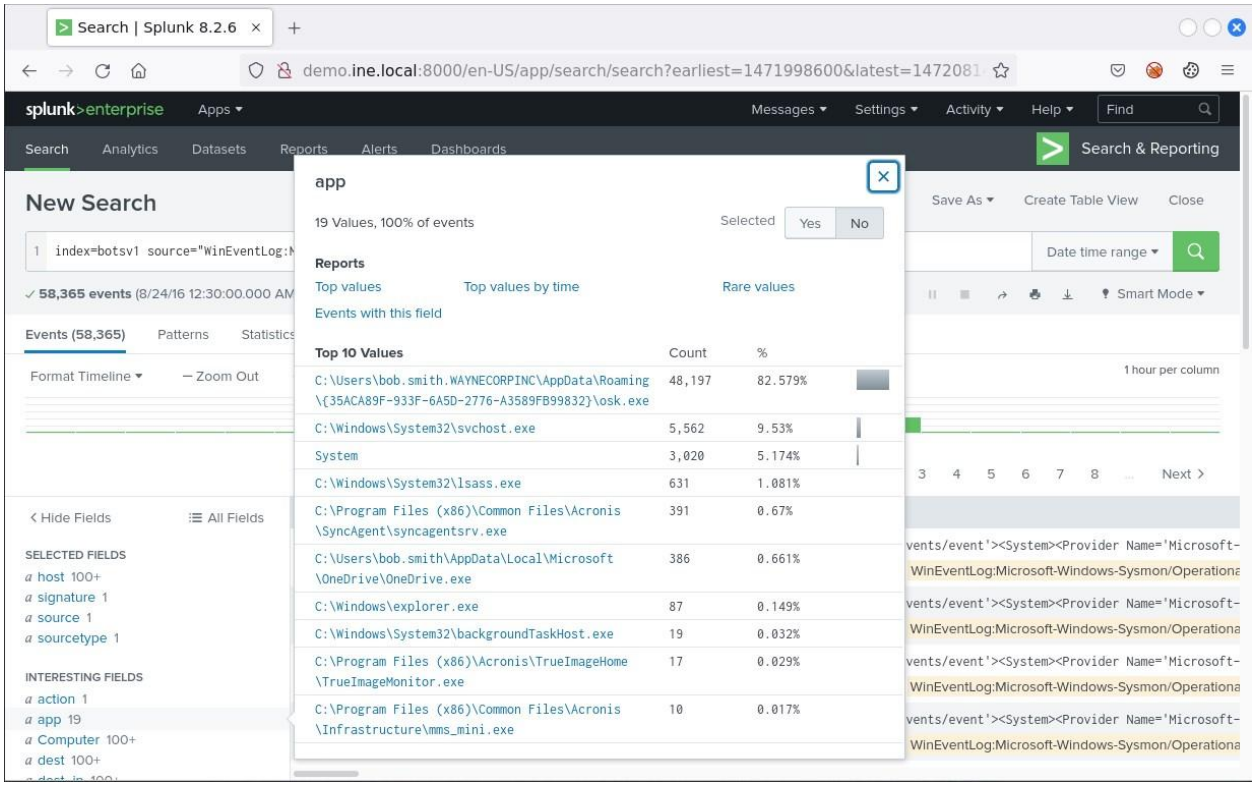
```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
SourceIp="192.168.250.100"
```

You should see something similar to the below.



Those two spikes are certainly suspicious.

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



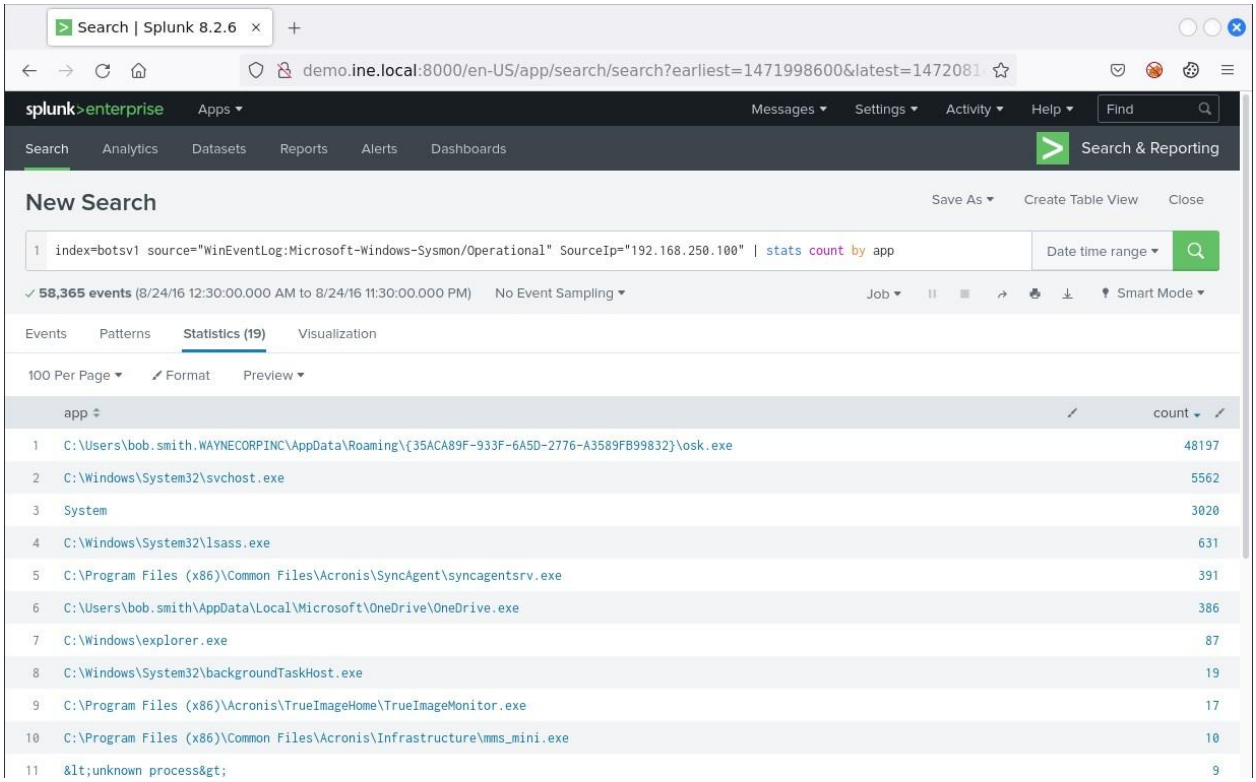
- The fact that *osk.exe* is in around 83% of the values is certainly suspicious

- *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

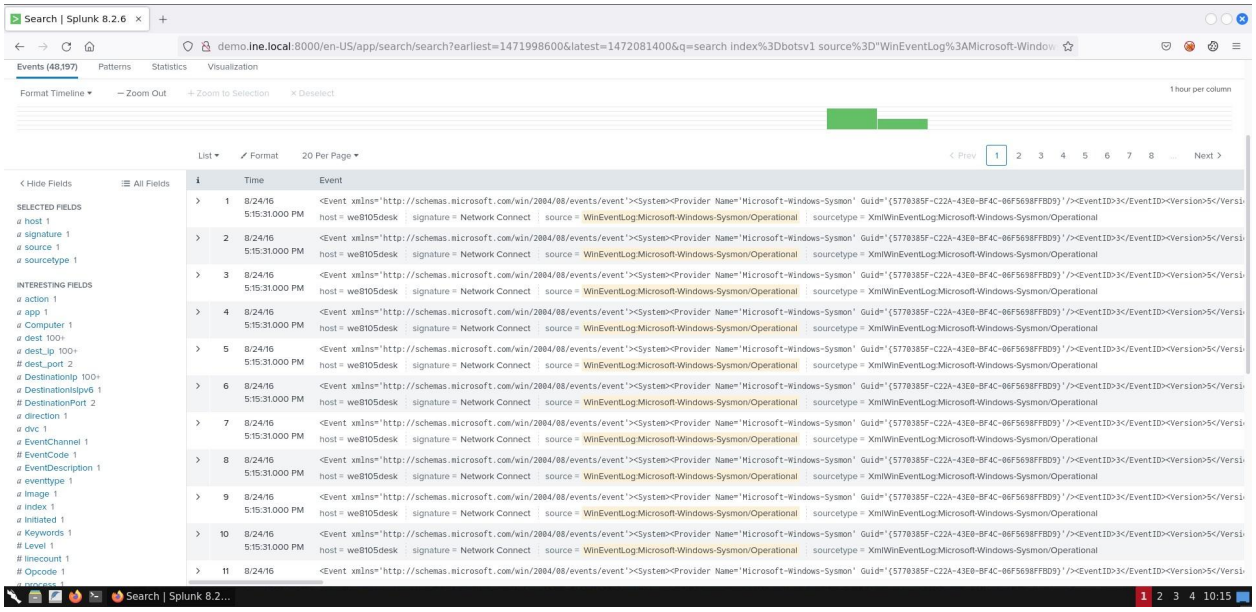
You could have also identified this application, as follows.

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

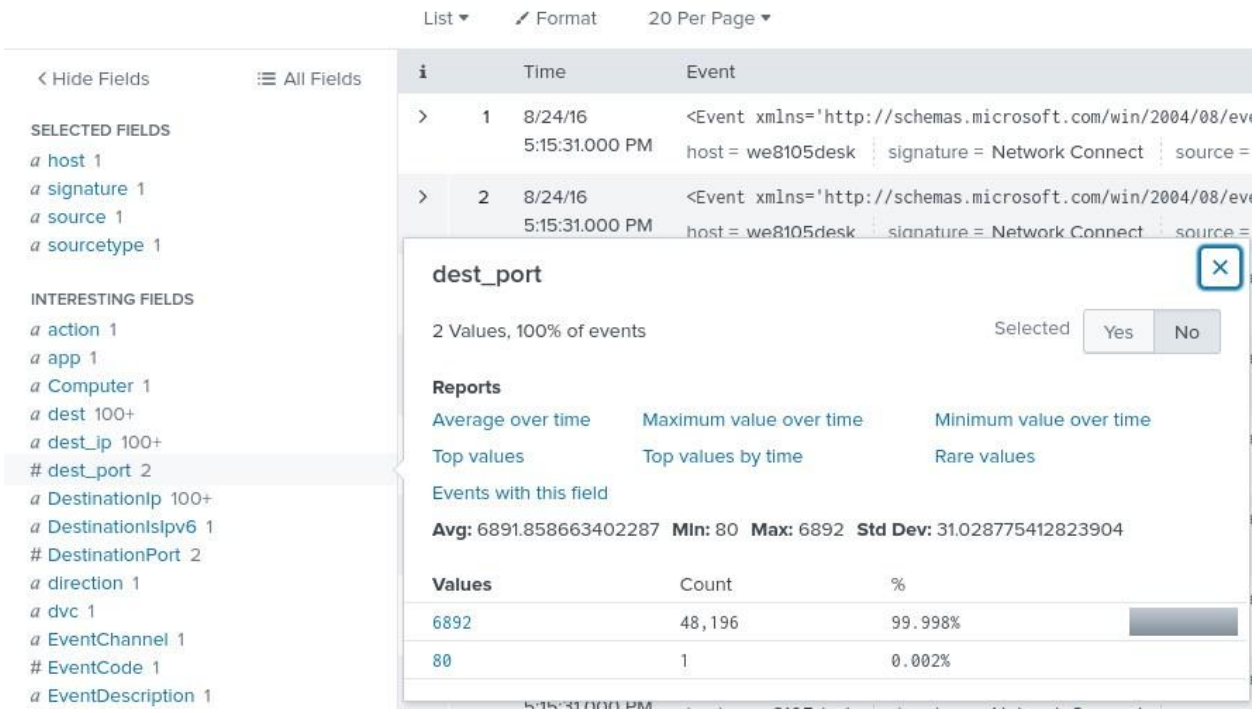


osk.exe definitely looks suspicious. So, give it a closer look by simply clicking on it.

You should see something similar to the below.



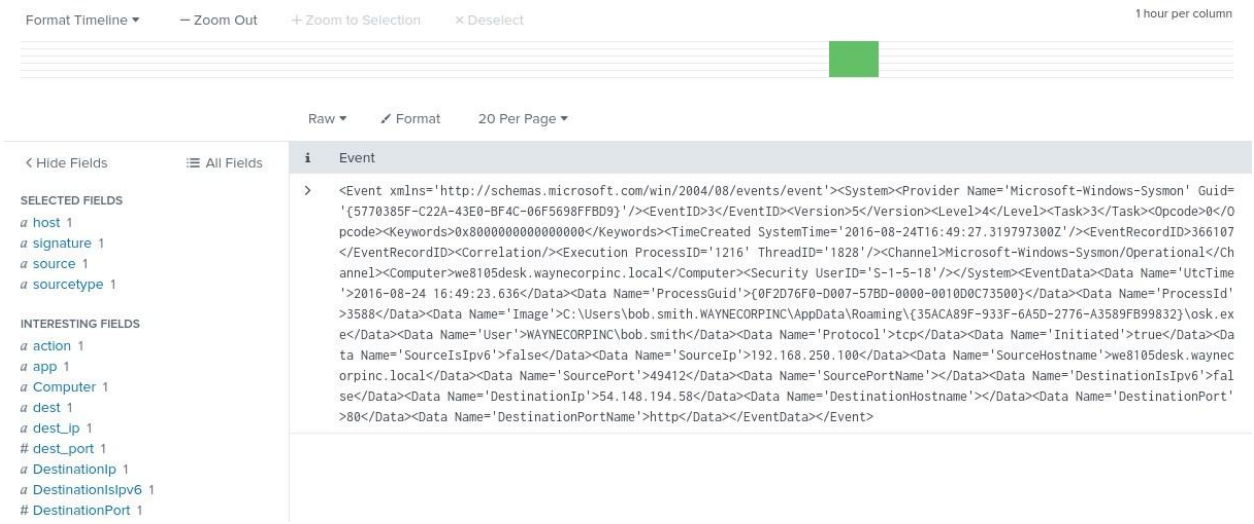
By inspecting the **dest_port** field. You will be presented with the below.



- 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.

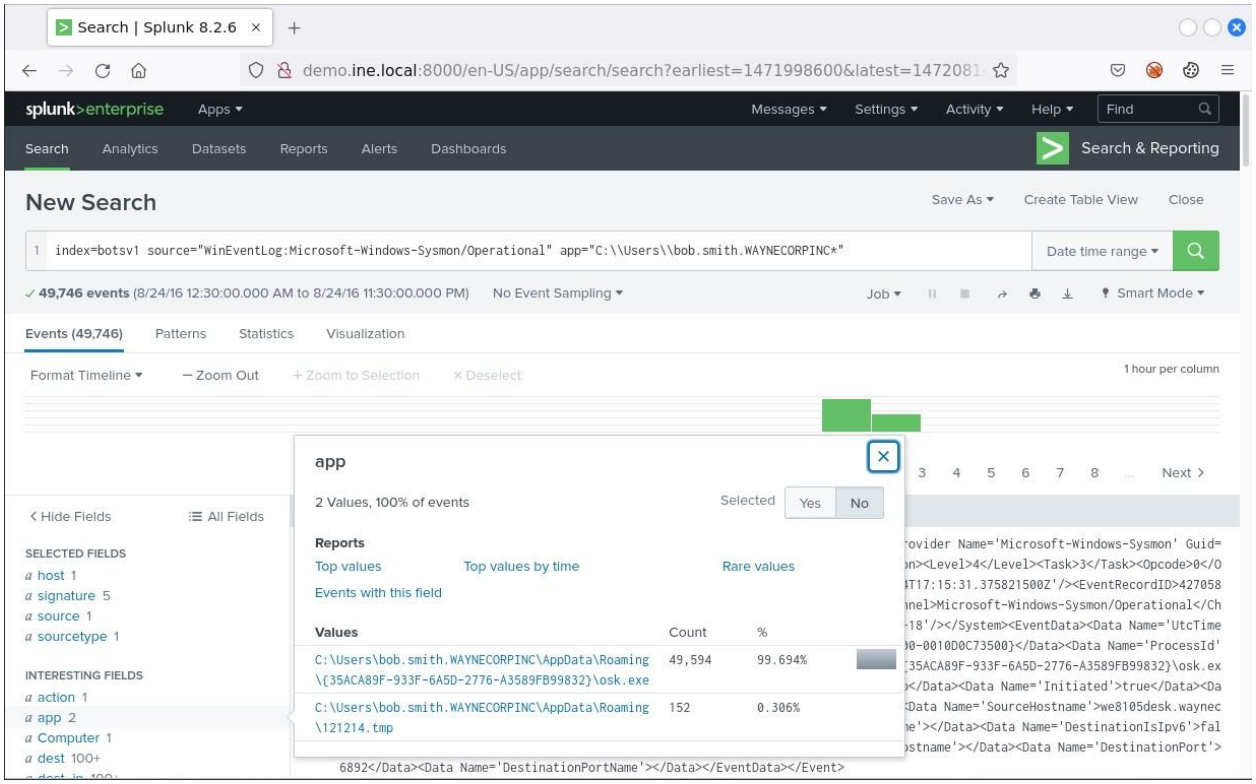
There is only one communication on port 80, click on it to learn more. You should see the below.



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.

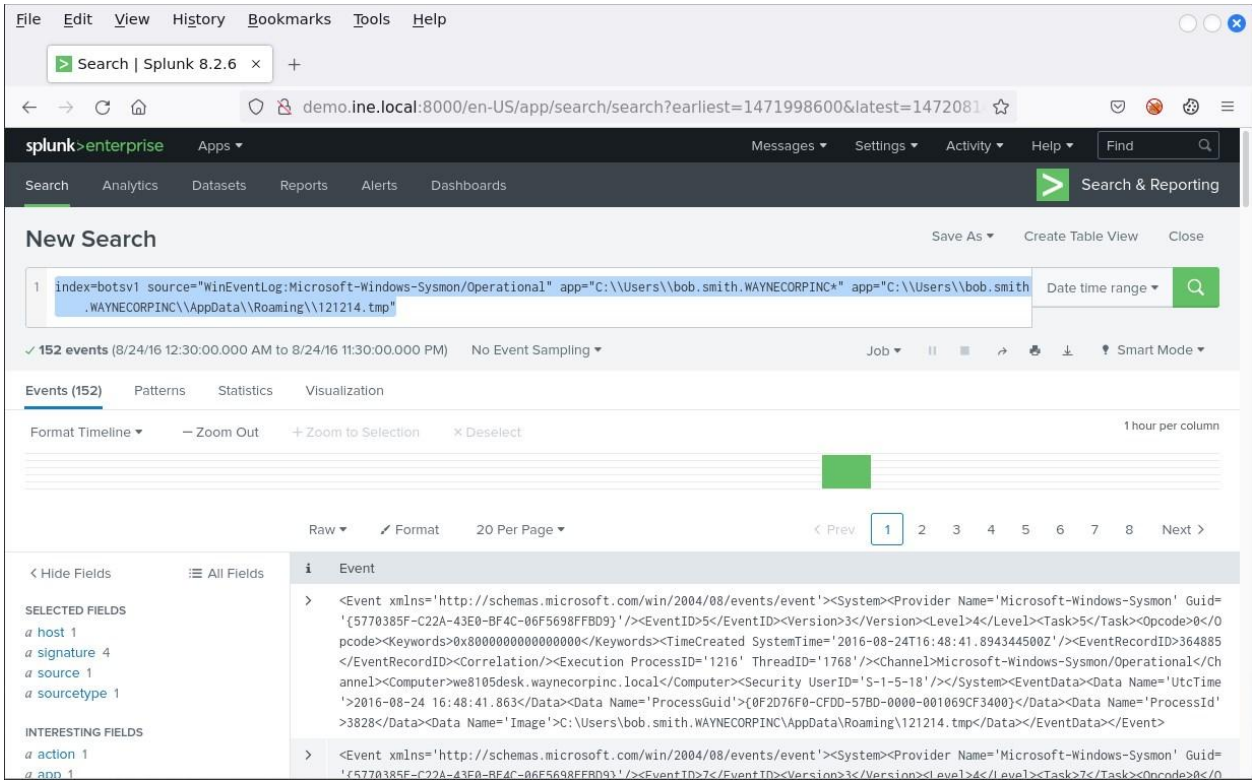
```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
app="C:\\Users\\bob.smith.WAYNECORPINC*"
```

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



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\\121
214.tmp"
```

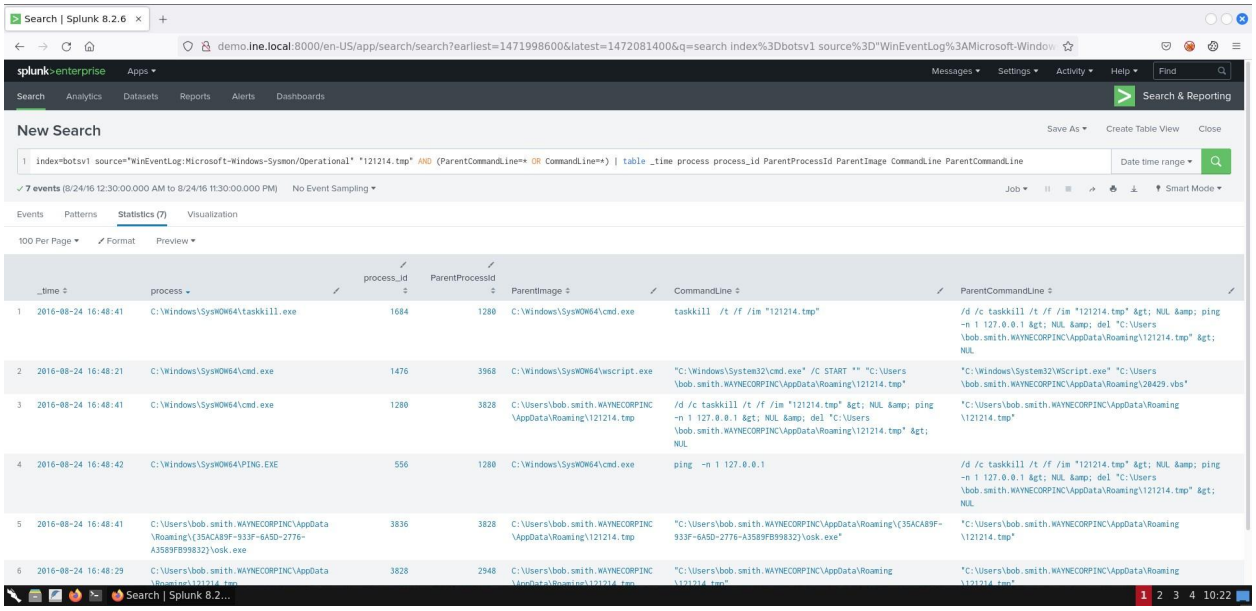


Nothing curious-looking in the results, but there are important fields that could be added to assist your investigation, such as the **CommandLine** or the **ParentCommandLine** one.

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.

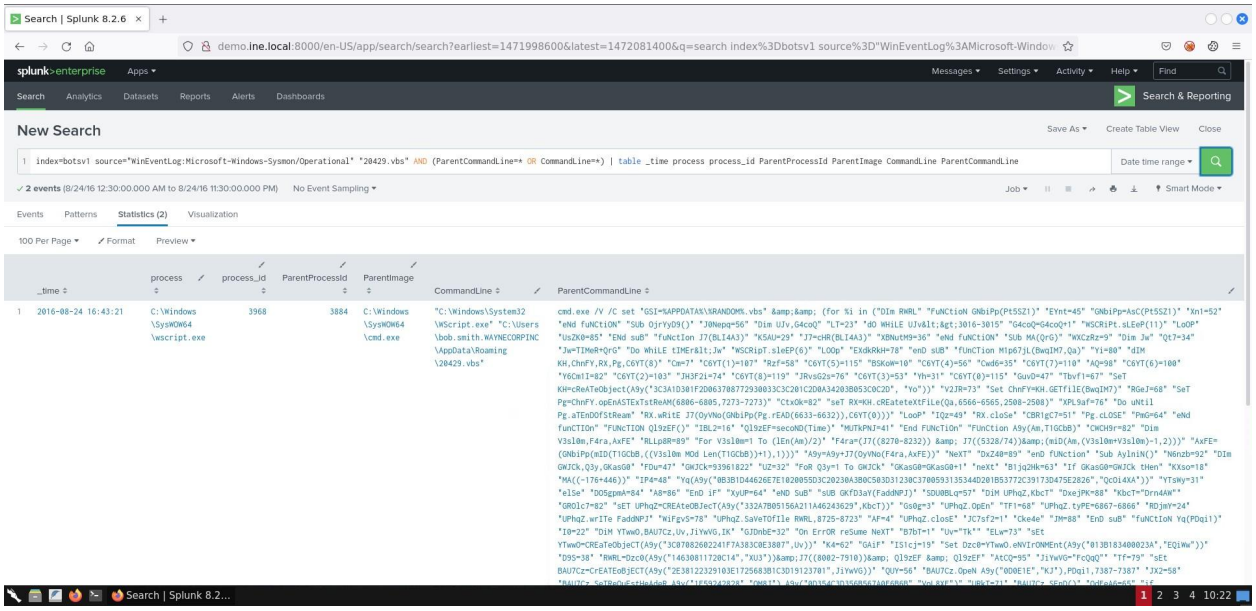


The earliest events are at the bottom of the table. If you start from the first (earliest) event you will see that *wscript.exe* (parent) called *cmd.exe* (child). In addition to that, you can see from **ParentCommandLine** that *wscript.exe* executed *20429.vbs*.

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
```

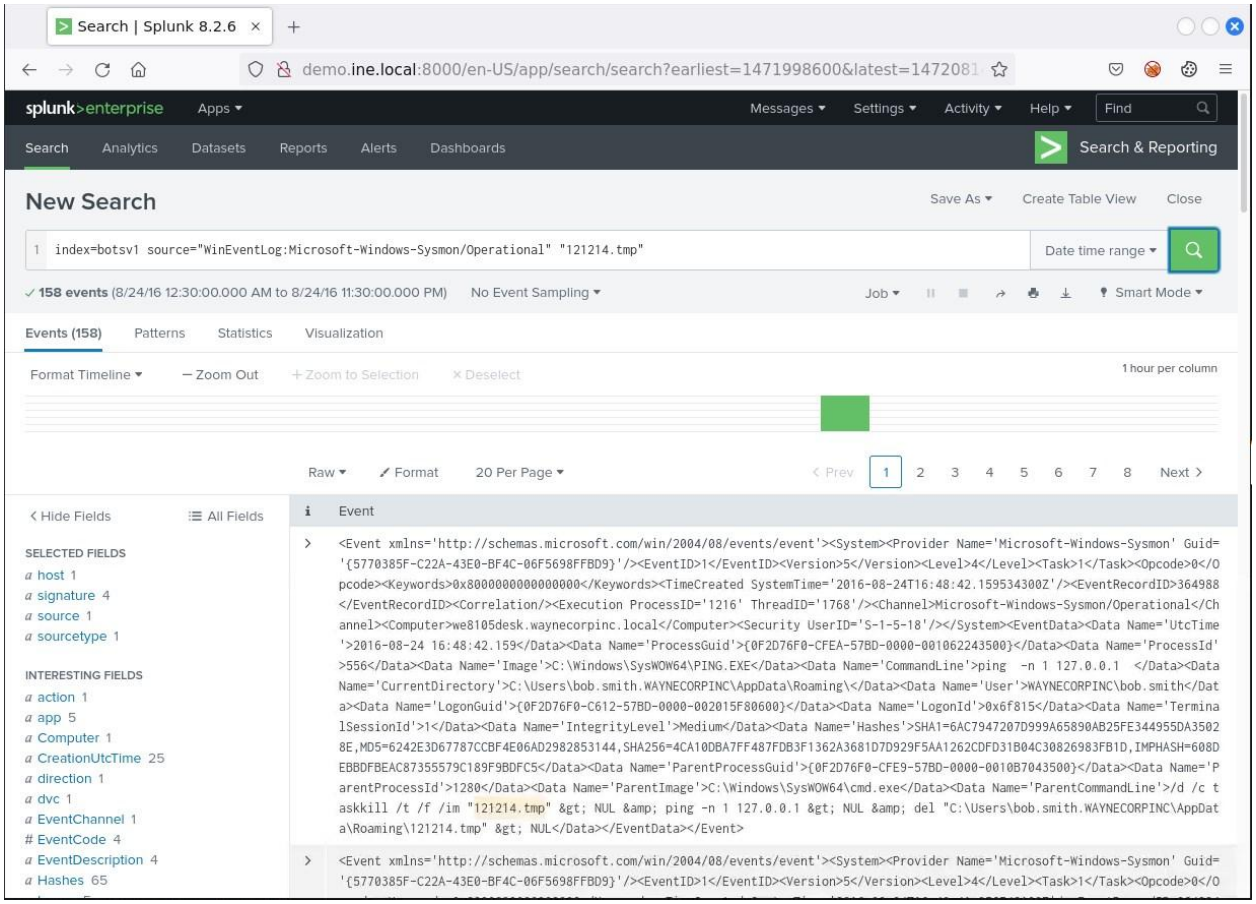
You should see the following.



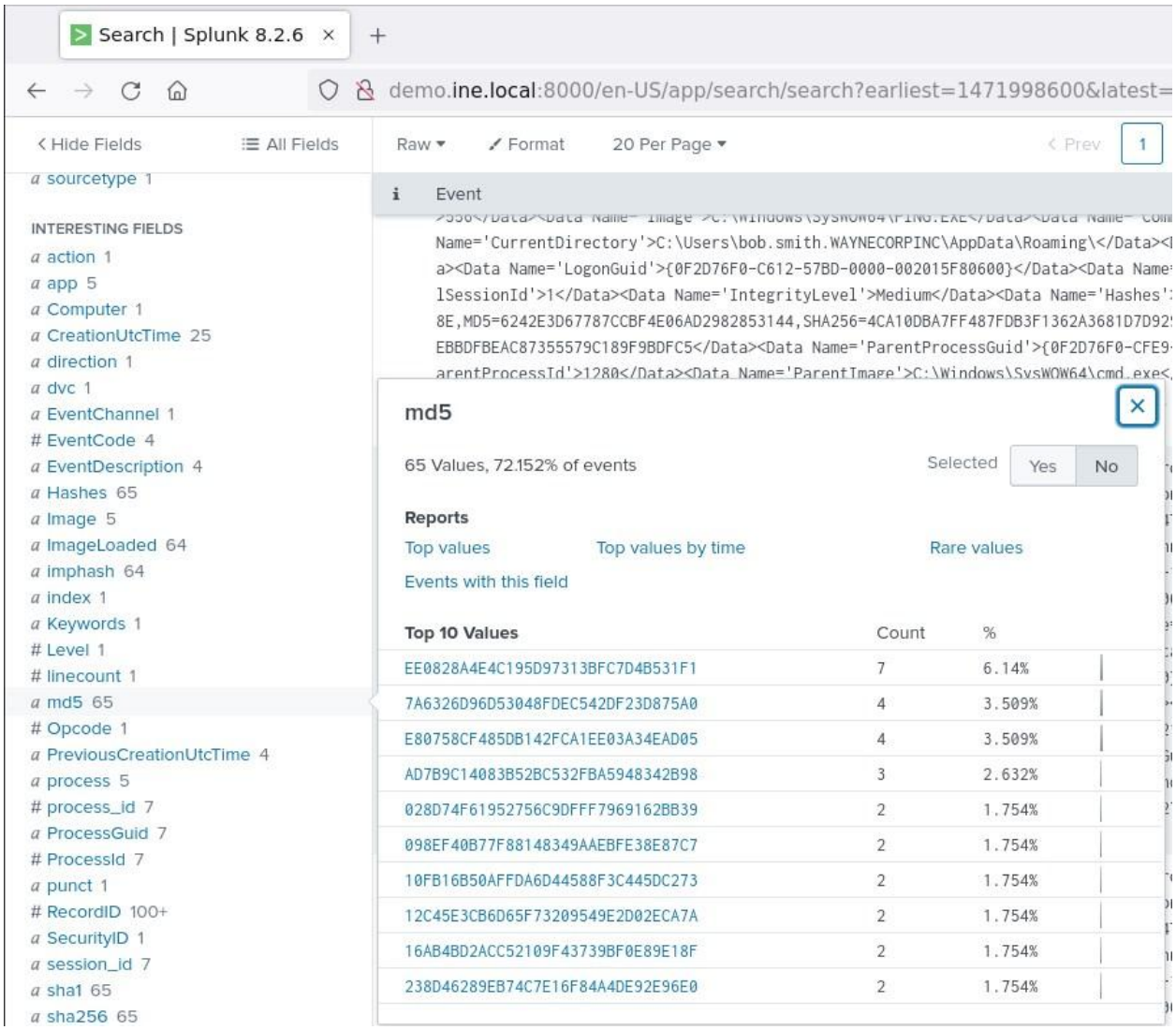
This is clearly obfuscated code. User Bob Smith is definitely victim of an attack.

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"
```



You will come across the following.



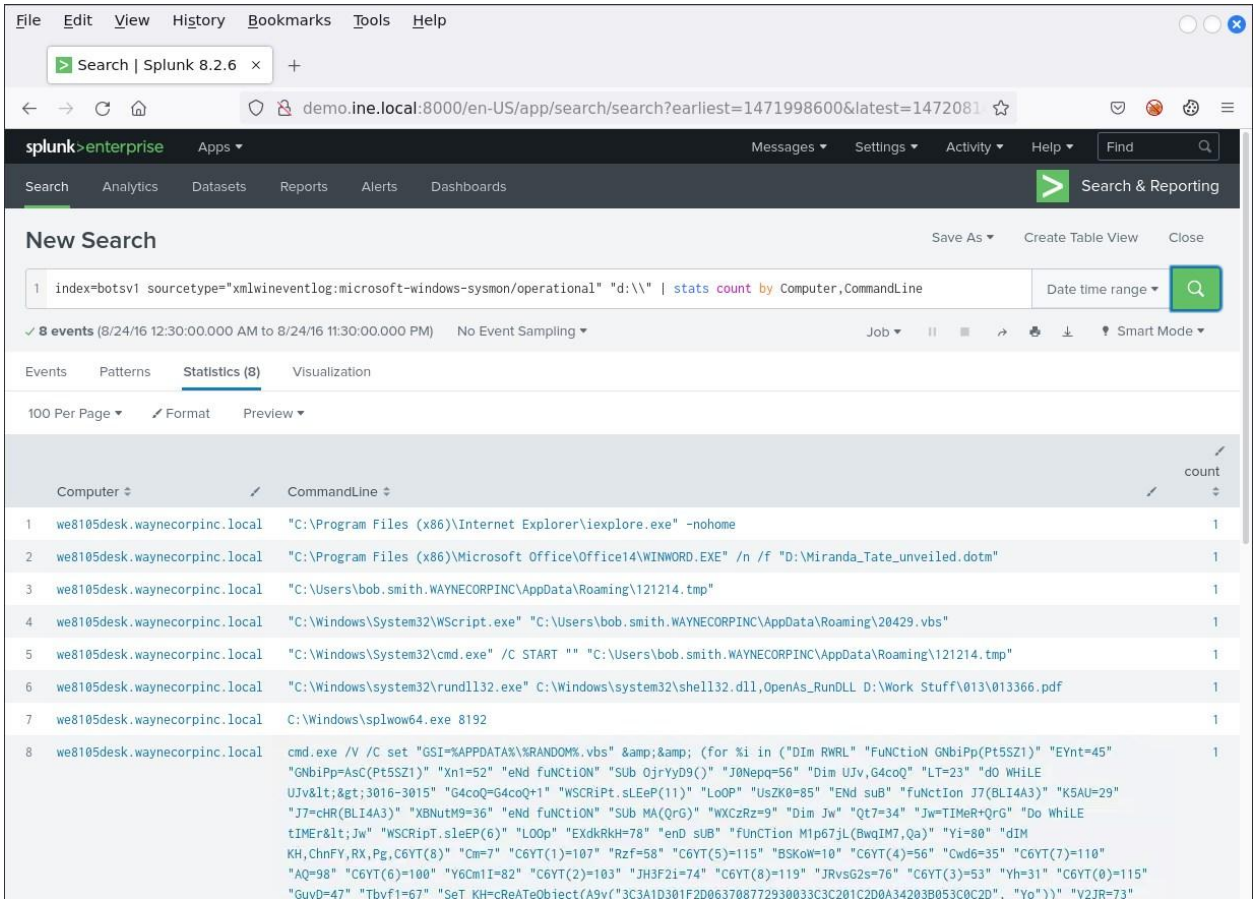
If you submit the **EE0828A4E4C195D97313BFC7D4B531F1** hash on a search engine, you will identify that you are dealing with Cerber ransomware.

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

1. Identify a malicious USB

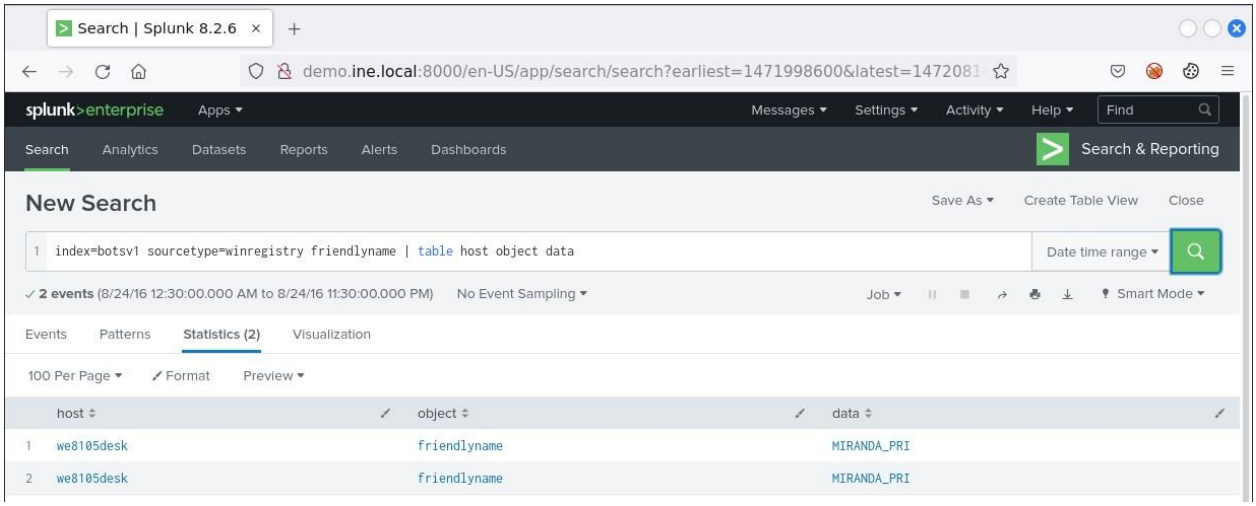
Removable media can be identified through the following Splunk searches.

```
index=botsv1
sourcetype="xmlwineventlog:microsoft-windows-sysmon/operation
al" "d:\\" | stats count by Computer,CommandLine
```



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

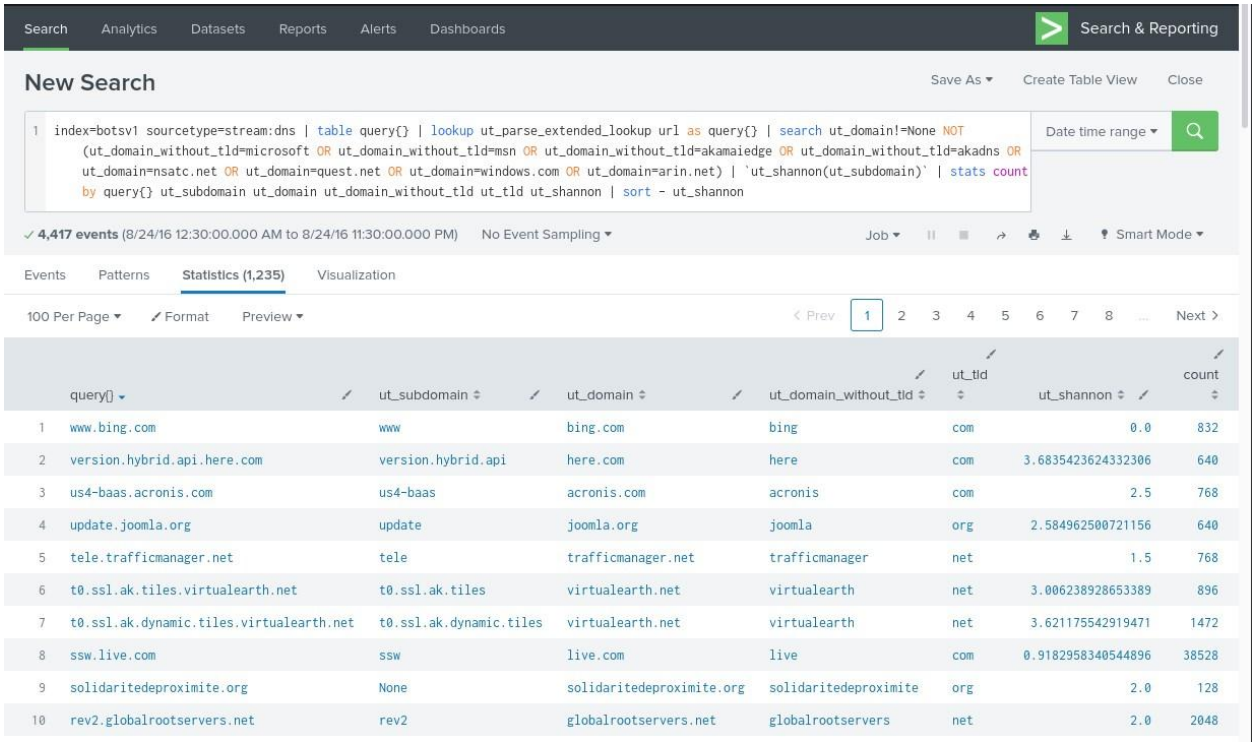
```
index=botsv1 sourcetype=winregistry friendlyname | table host
object data
```



1. Identify computer-generated domain names

The following search may uncover computer-generated domain names.

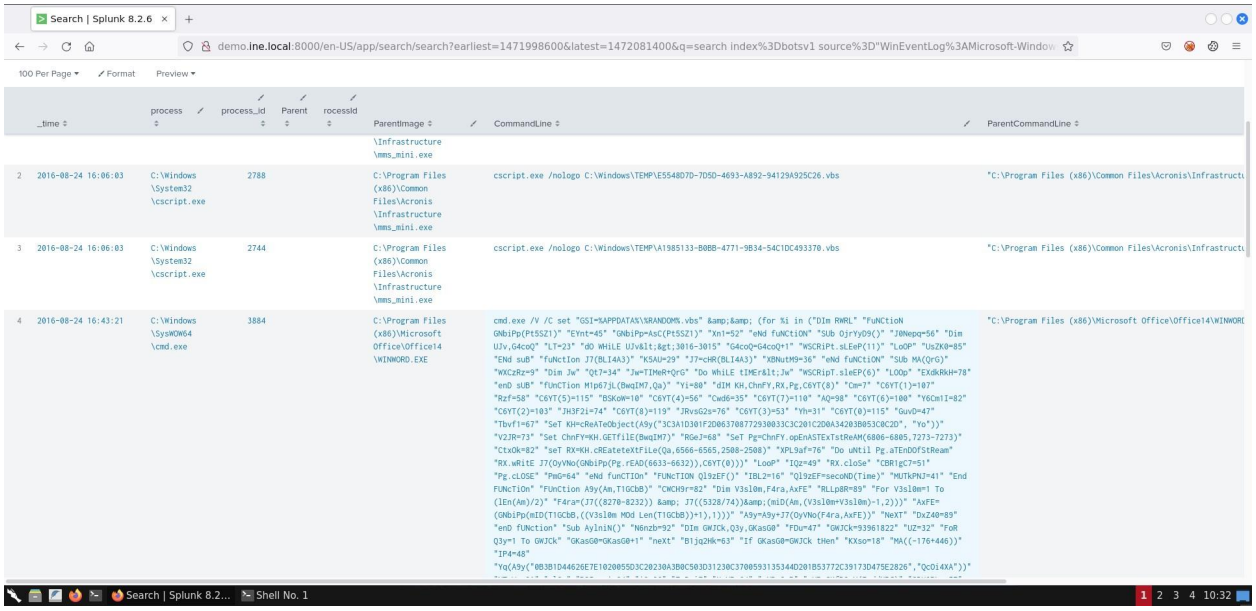
```
index=botsv1 sourcetype=stream:dns | table query{} | lookup ut_parse_extended_lookup url as query{} | search ut_domain!=None NOT (ut_domain_without_tld=microsoft OR ut_domain_without_tld=msn OR ut_domain_without_tld=akamaiedge OR ut_domain_without_tld=akadns OR ut_domain=nsatc.net OR ut_domain=quest.net 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
```



1. Identify malicious VBS

The following search may identify malicious VBS files

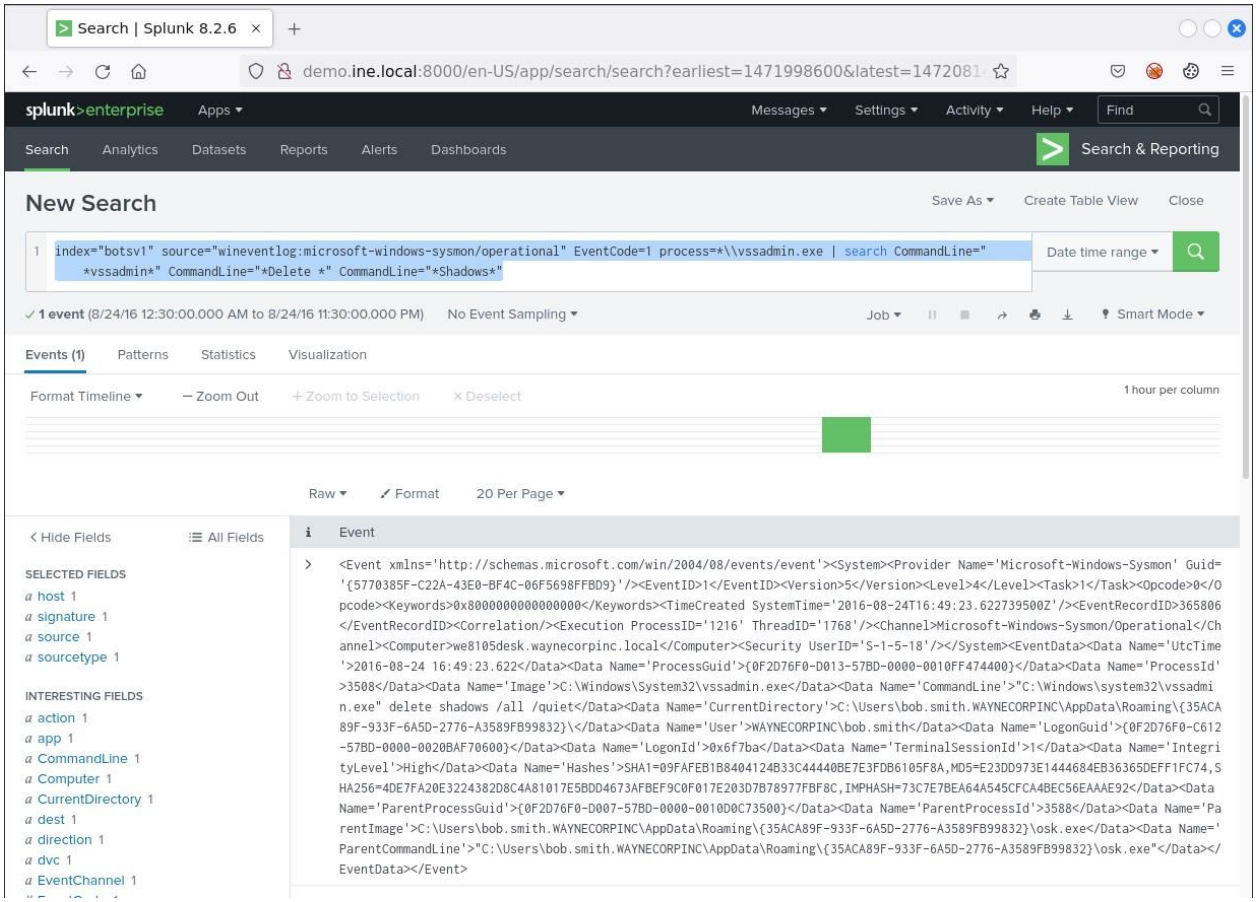
```
index=botsv1
source="WinEventLog:Microsoft-Windows-Sysmon/Operational"
"*.vbs" AND (ParentCommandLine=* OR CommandLine=*) | table
_time process process_id Parent rocessId ParentImage
CommandLine ParentCommandLine
```



1. Identify mature ransomware activity

The following search can possibly identify mature ransomware activity.

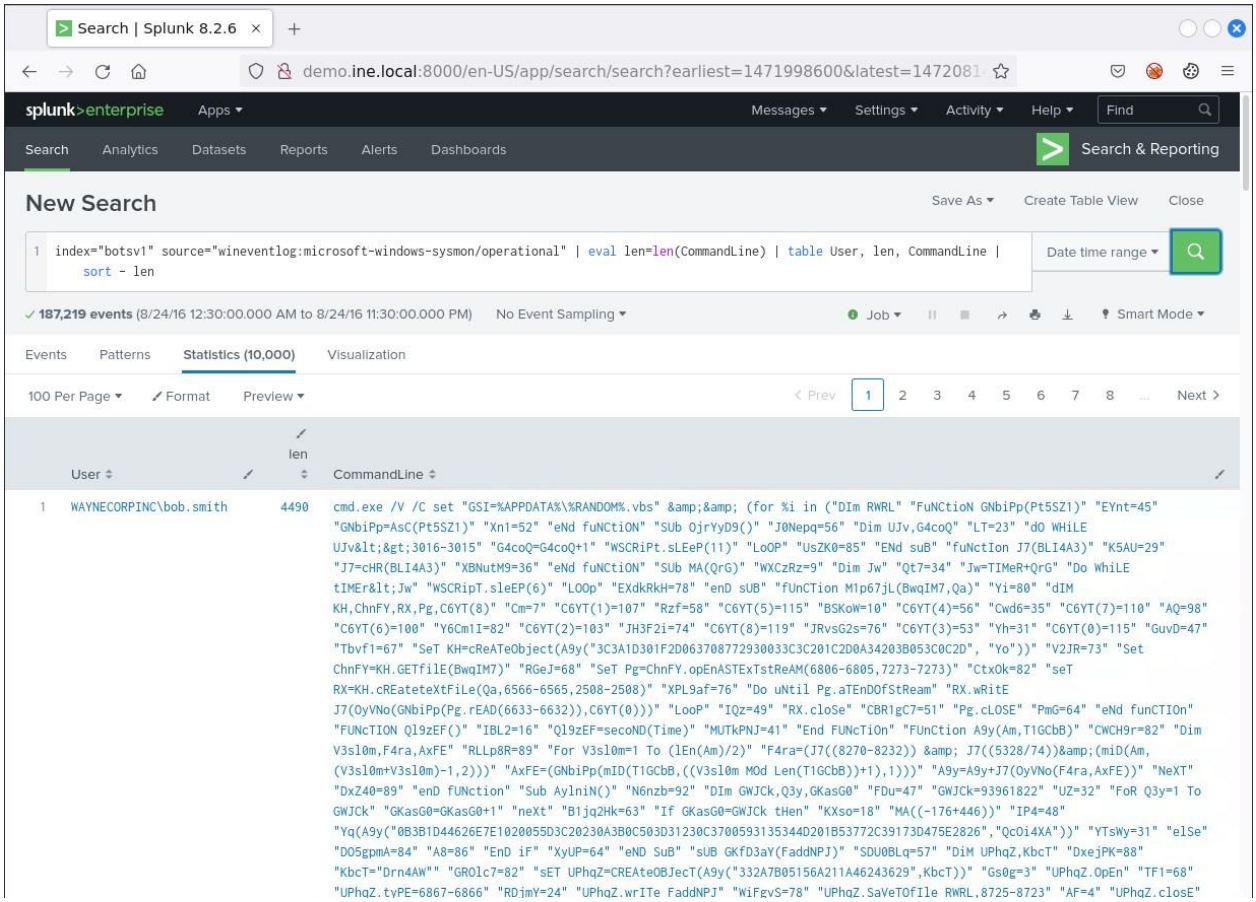
```
index="botsv1"
source="wineventlog:microsoft-windows-sysmon/operational"
EventCode=1 process=*\\vssadmin.exe | search
CommandLine="*vssadmin*" CommandLine="*Delete *"
CommandLine="*Shadows*"
```



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
```



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.

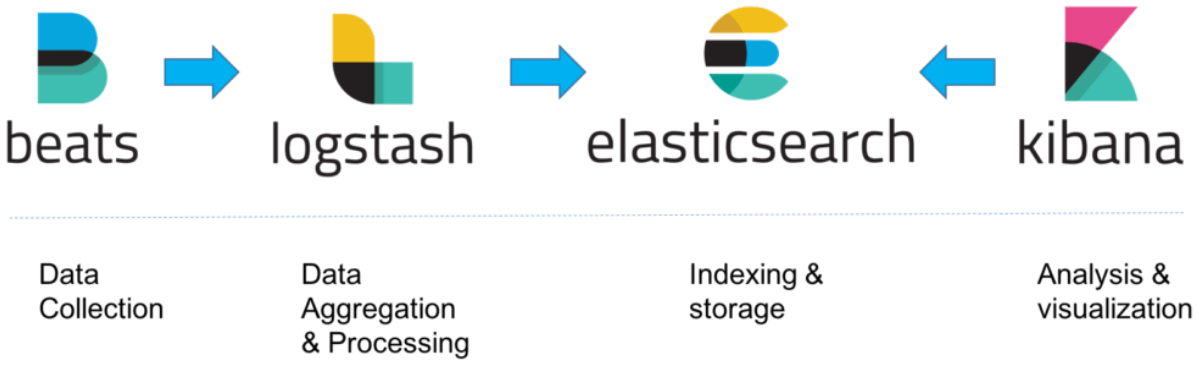
Learning Objectives

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

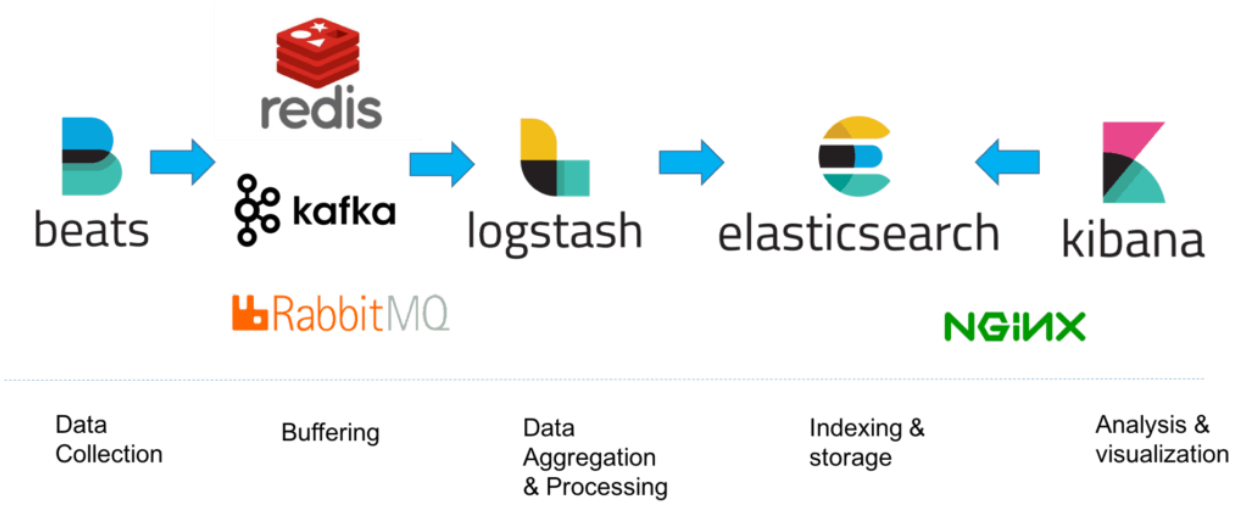
Introduction To ELK

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 ngnix 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.

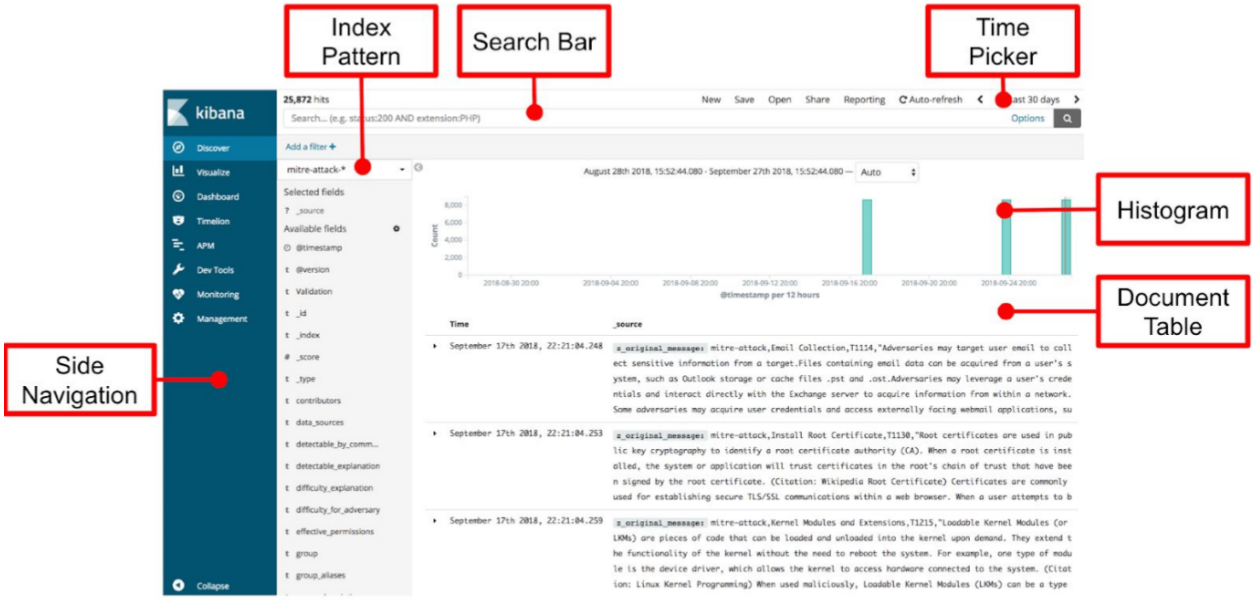
- **Send log records** to Elasticsearch by utilizing any of the output plugins.

- **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.

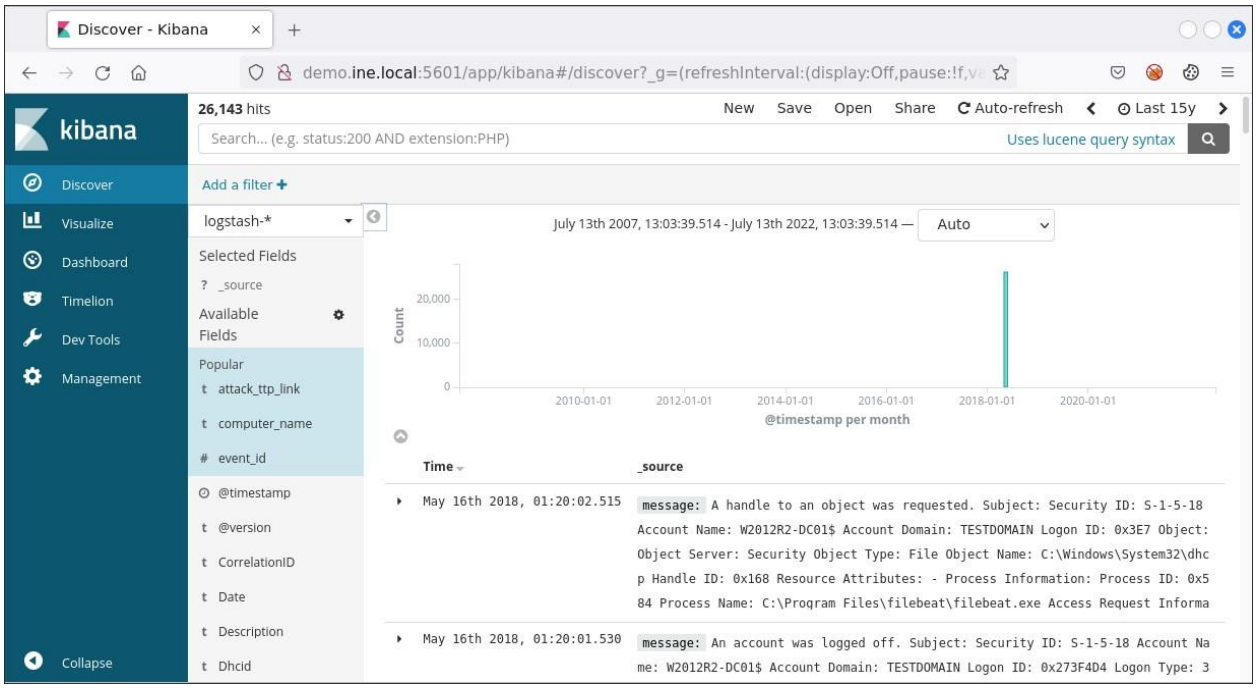
SOLUTIONS

Kali Machine

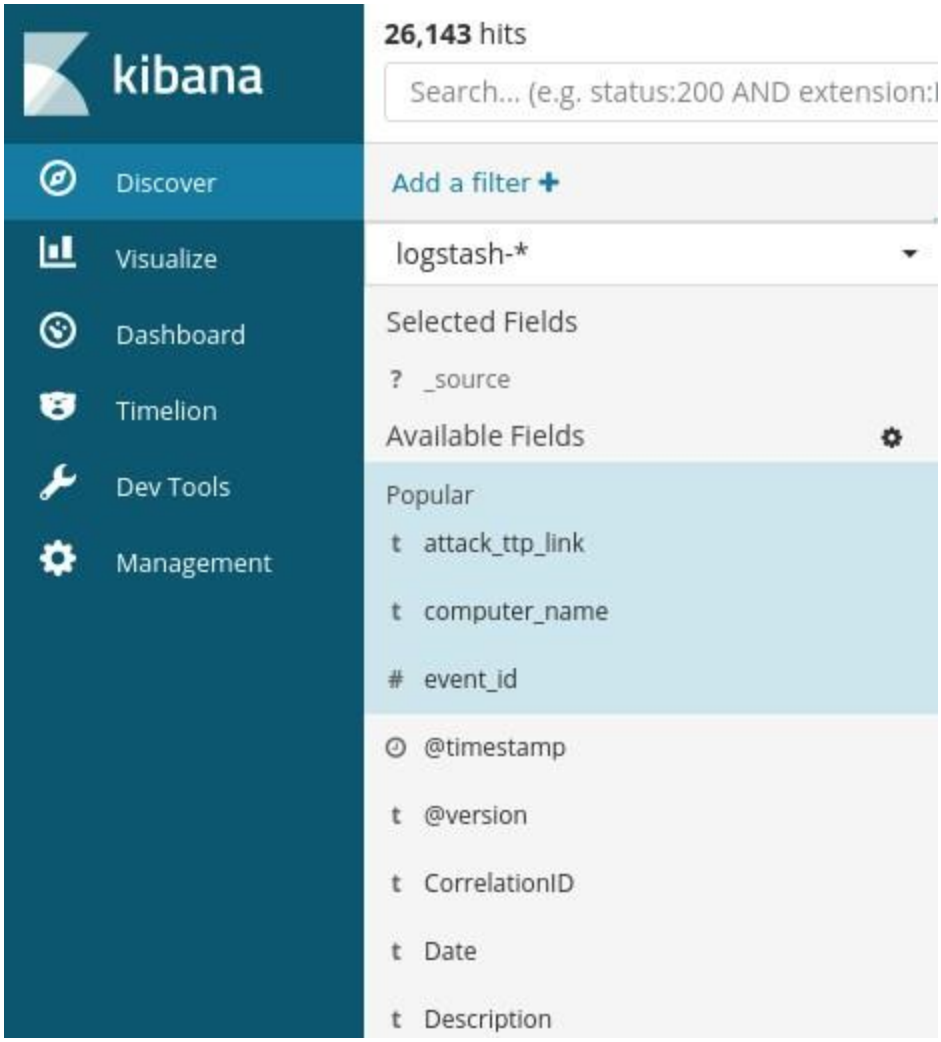
Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab.

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.

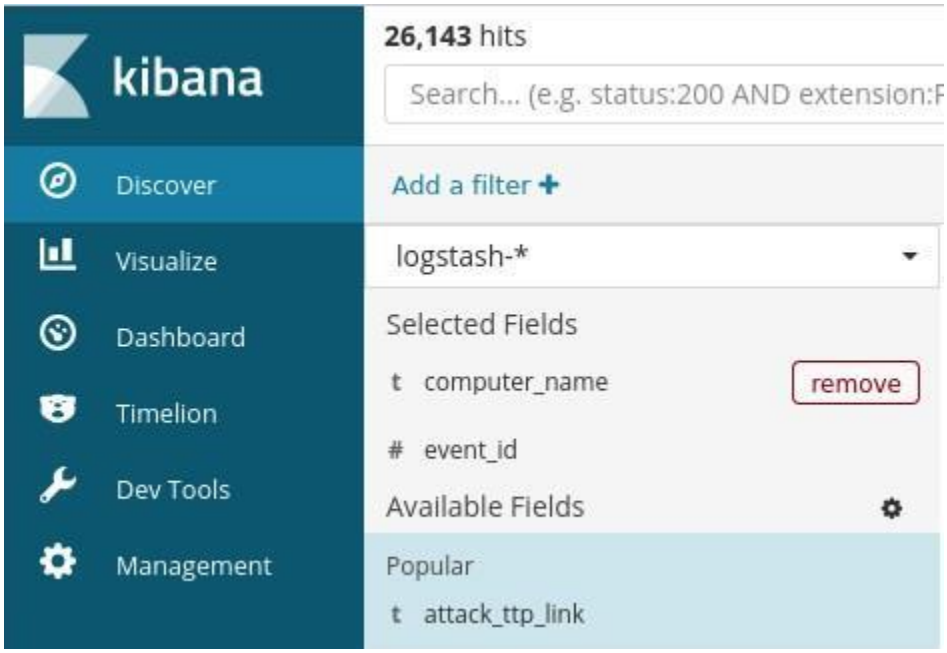


A good start would be adding the **event_id** field and the **computer_name** field. Then, the results will look like the ones depicted below.

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

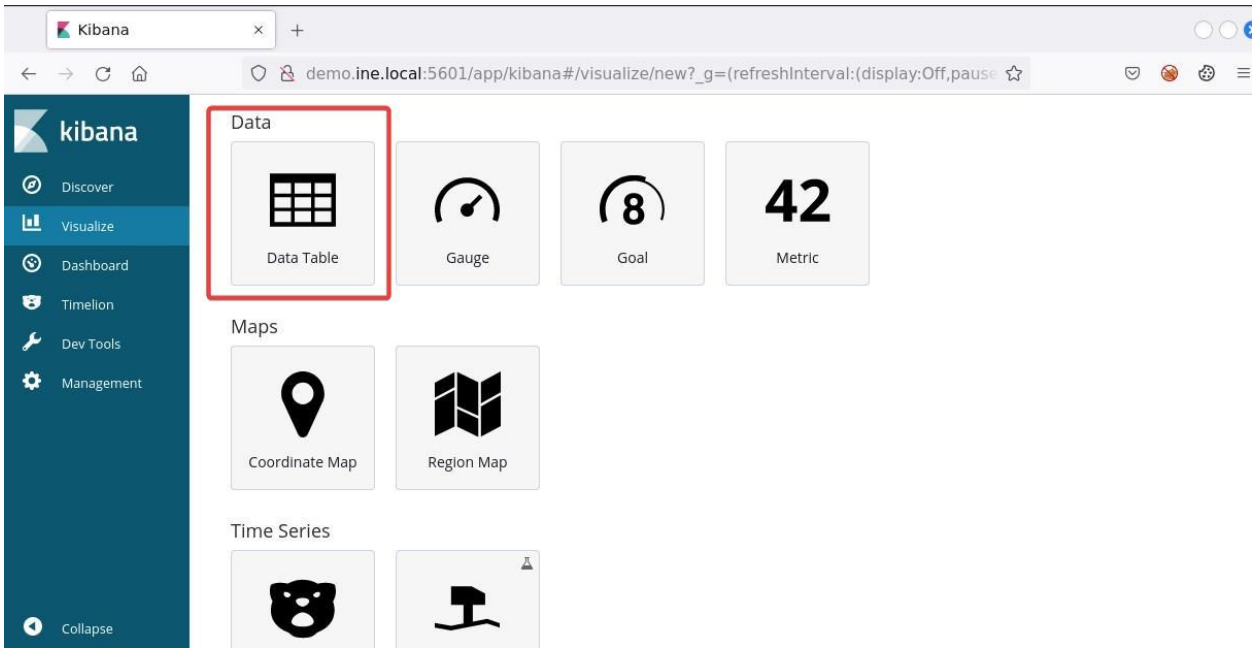
To identify all users included in the dataset you can start by submitting an empty search, expanding **Available Fields** and then inspecting the **event_data.User** field. If you do you so will come across the below.

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

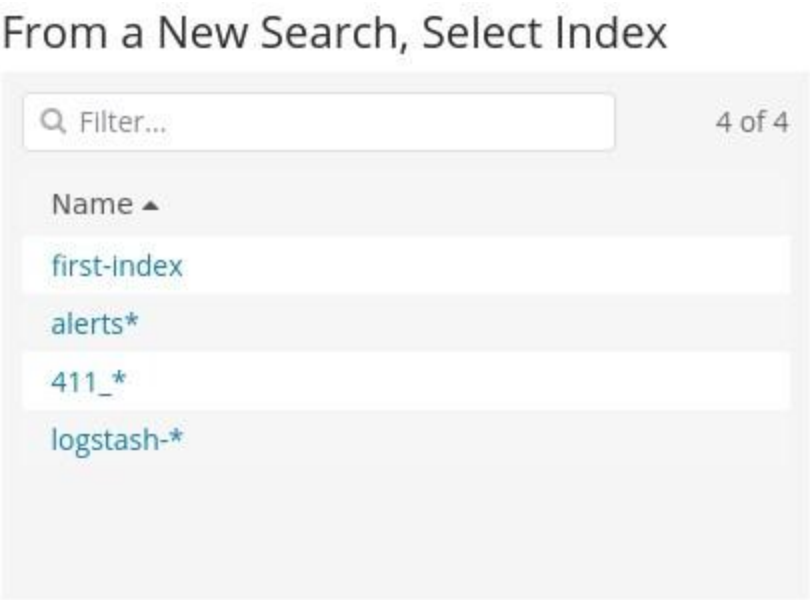
that by going to the **Management** tab and then clicking **Advanced Settings**, but let's create a visualization instead.

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

Then, click on **Data Table**.



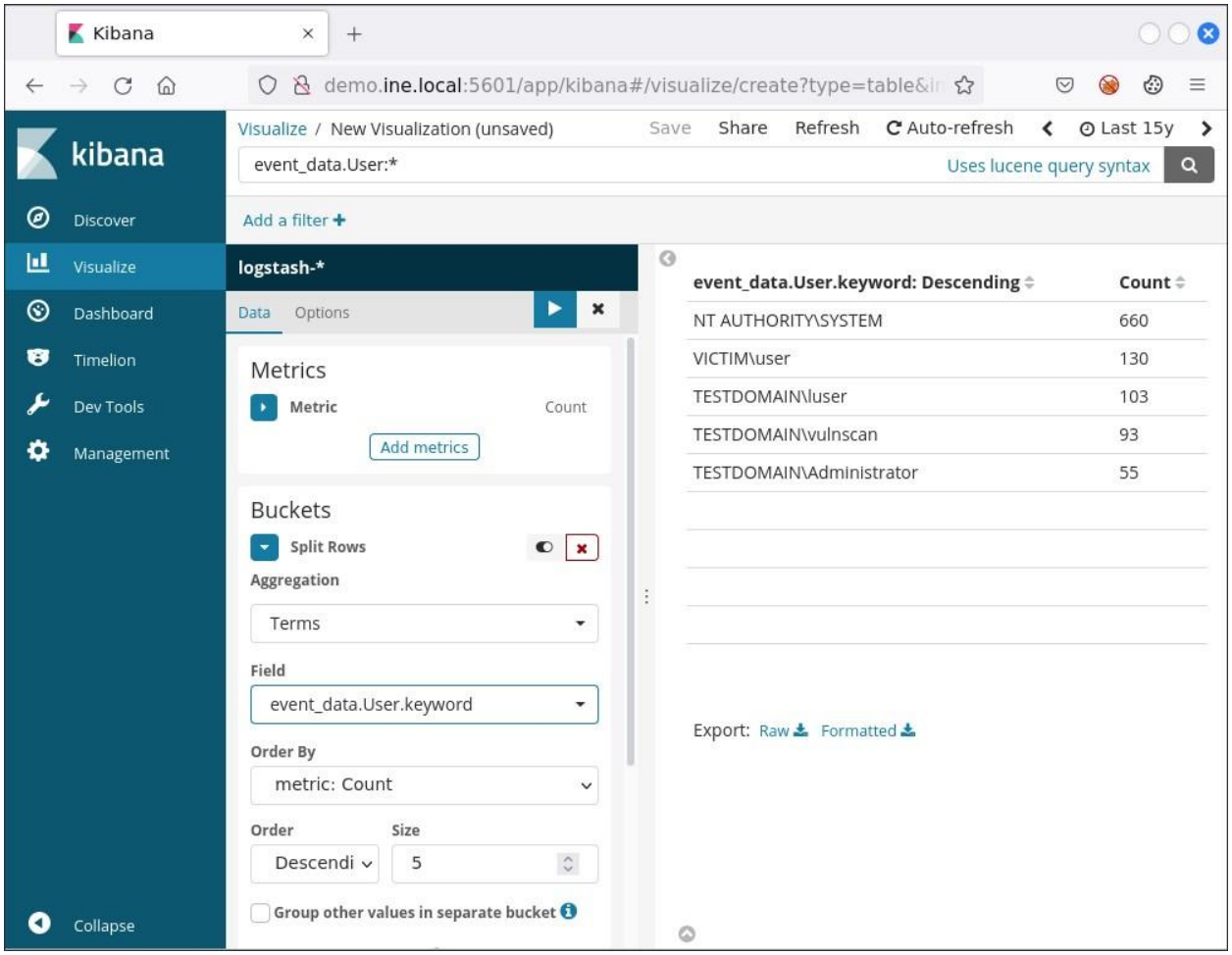
Now, click on the **logstash-*** index...



and submit the search below.

event_data.User:*

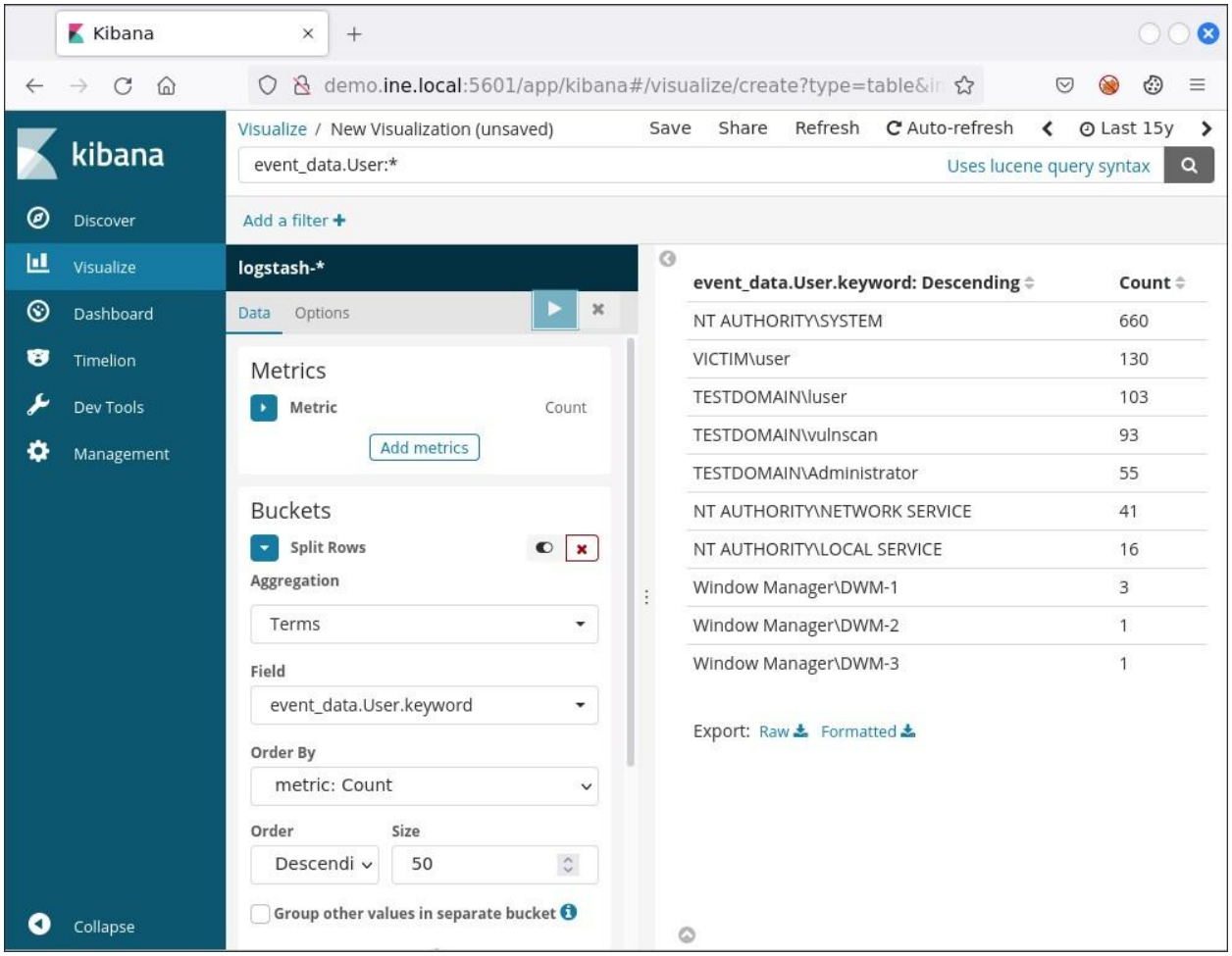
The search above can be translated as **winlog.event_data.User:"exists"**. This means that it will provide us with all the documents that contain the specified field. To identify all users, we need to create an aggregation. To create one, click the **Split Rows** tab and choose **Terms** from the Aggregation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



You should now see the below

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.

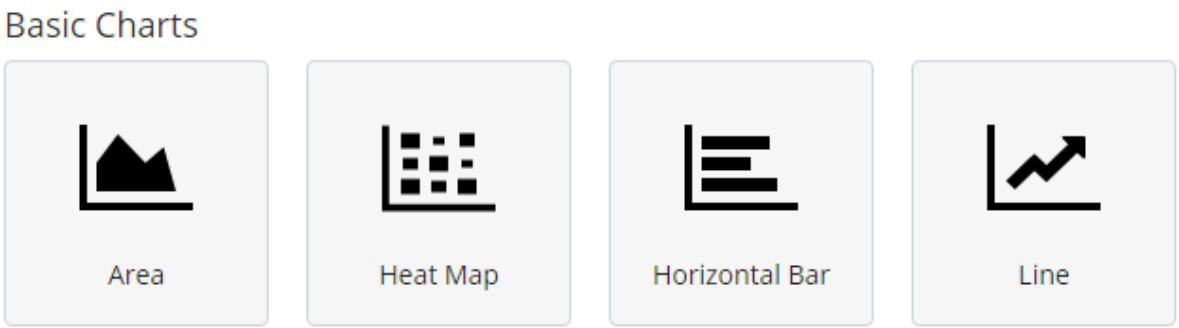
(event_id:4776 AND -keywords:"Audit Success") OR event_id:4625

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.

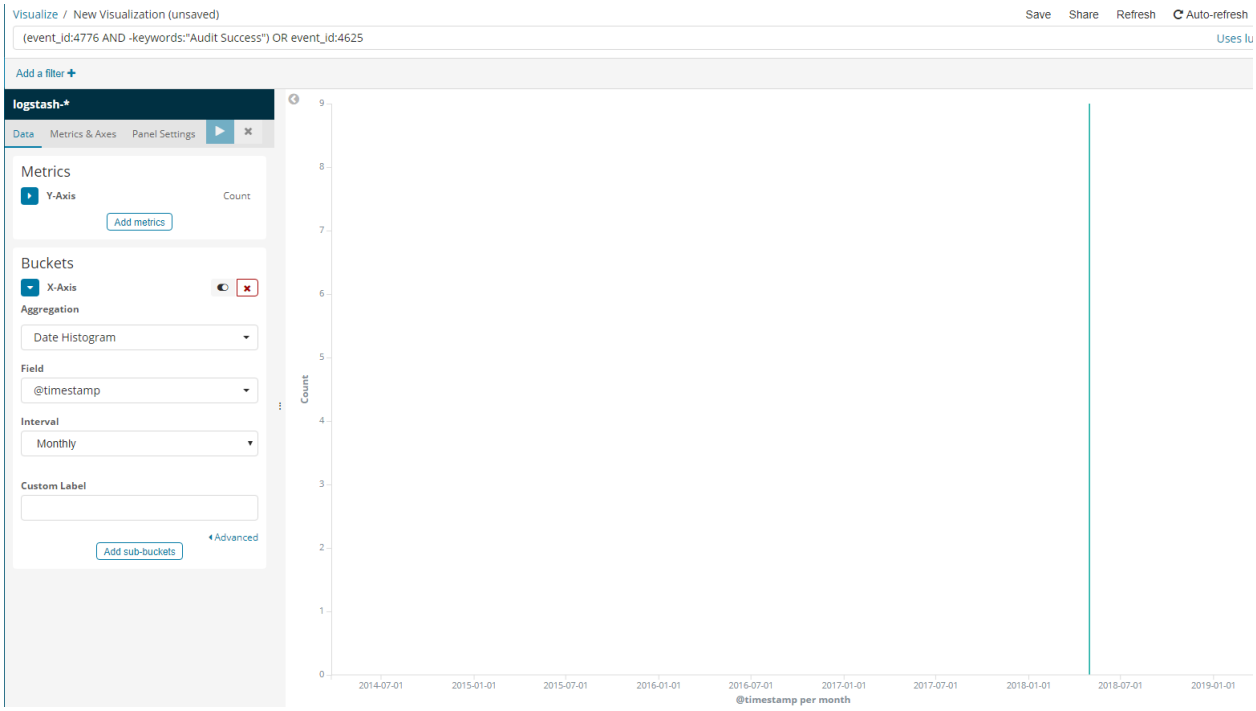
To create such a visualization we click **Visualize**, we press the button with the cross and we choose **Area**



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"
"*\\wuauclt.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"
"*\\wuauclt.exe" "*\\taskhost.exe" "*\\taskhostw.exe"
"*\\fontdrvhost.exe" "*\\searchindexer.exe"
"*\\searchprotocolhost.exe" "*\\searchfilterhost.exe"
"*\\sihost.exe") AND
-event_data.TargetFilename:("*\\system32\\*" "*\\syswow64\\*"
"*\\winsxs\\*") )
```

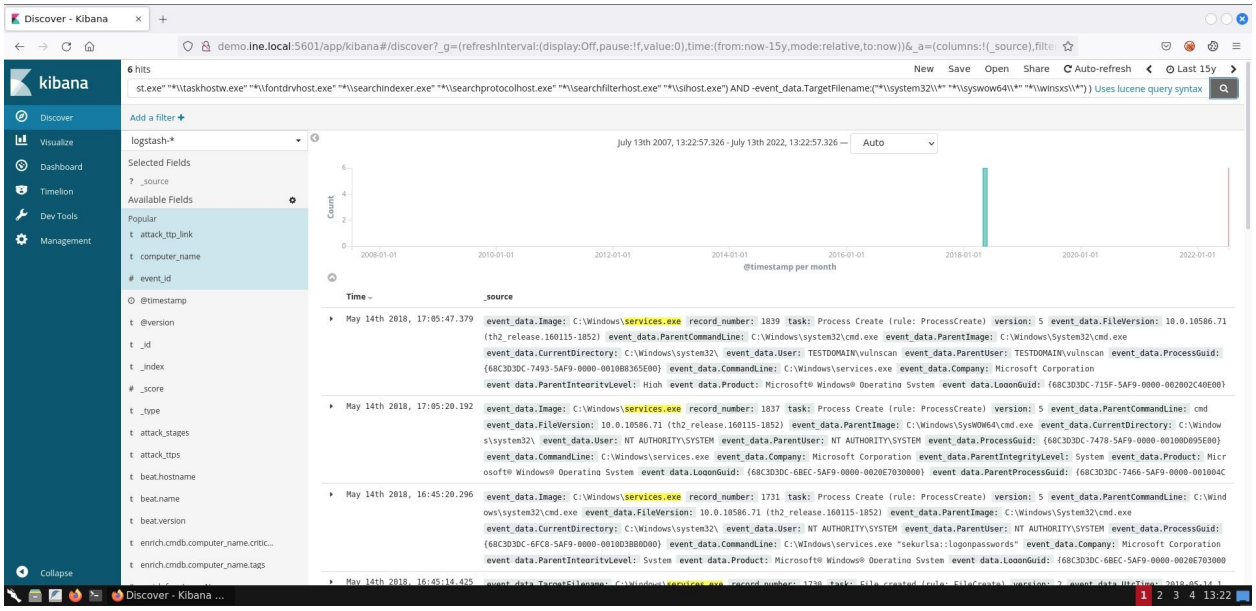
AND -event_data.Image is excluding the expected paths.

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.

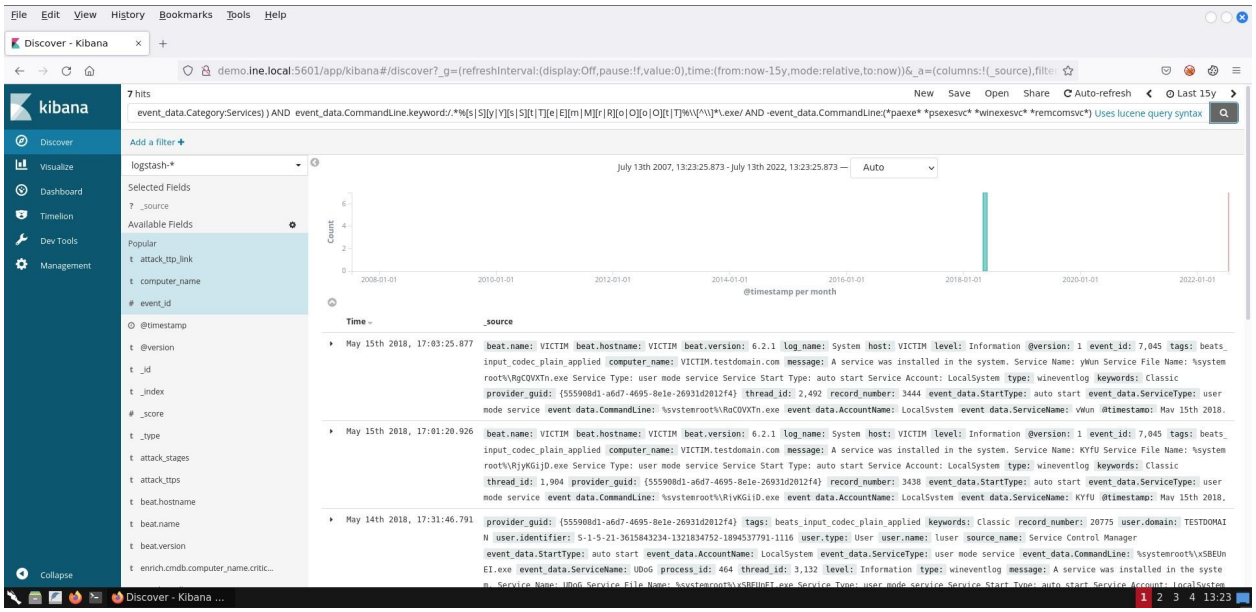
```
(event_id:("4697" "7045") OR (log_name:Autoruns AND
event_data.Category:Services) ) AND
event_data.CommandLine.keyword:/.*%[s|S][y|Y][s|S][t|T][e|E][
m|M][r|R][o|O][o|O][t|T]%\\[^\\]*\.exe/ AND
-event_data.CommandLine:(*paexe* *psexesvc* *winexesvc*
*remcomsvc*)
```

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]

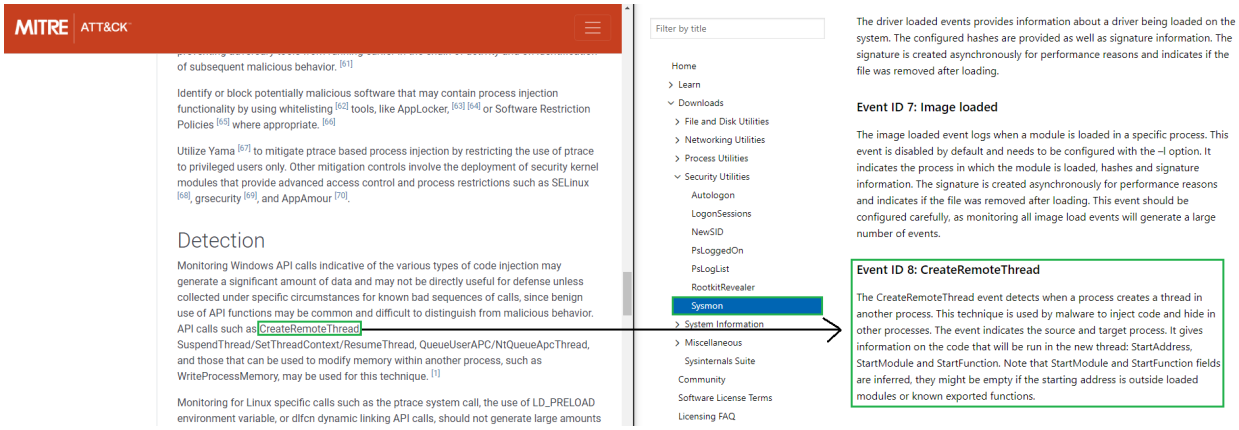
-event_data.CommandLine:(*paexe* *psexesvc* *winexesvc* *remcomsvc*) excludes services that interact with expected Windows executables inside the Windows folder.

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.

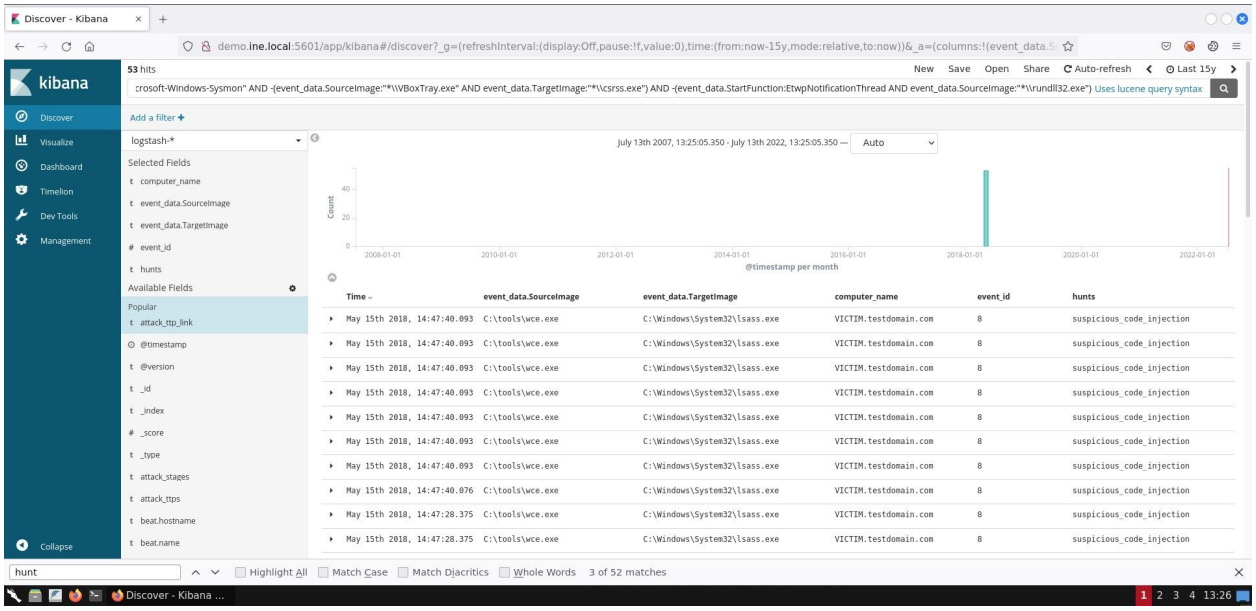


It should be noted that remote threads can be created on a Windows system for legitimate purposes as well. Such an example is *EtwpNotificationThread*, which is related to threads (thread entry points actually) being "created" in the context of a process, so that certain tasks can be performed on behalf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

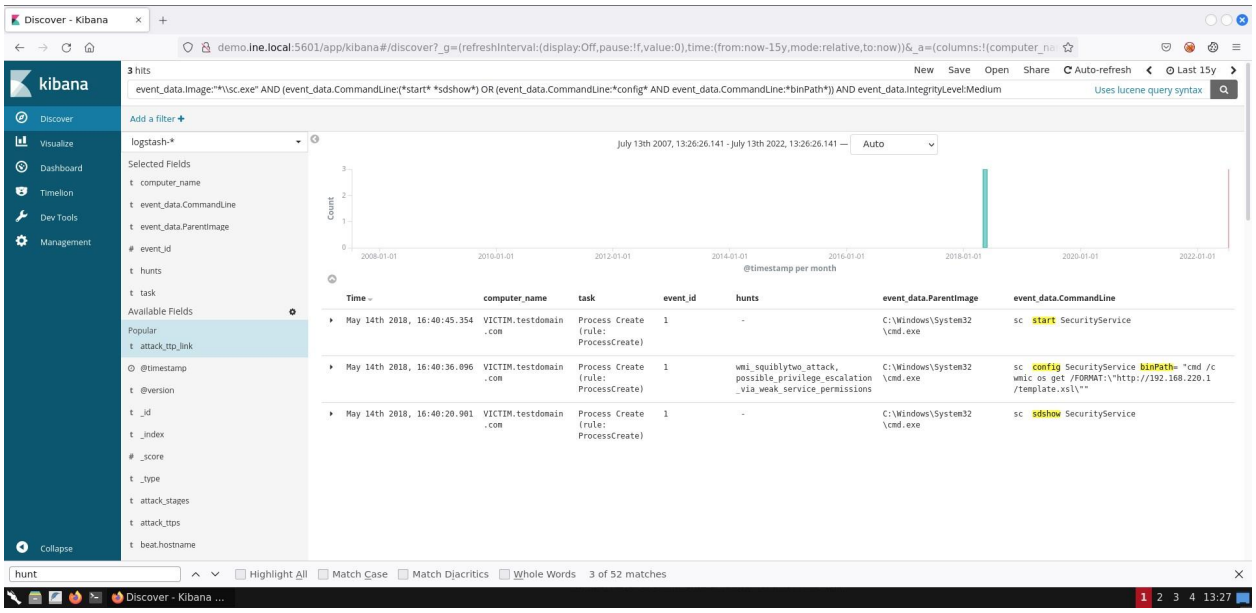
If they have enough privileges, attackers may also attempt to specify an executable of their own to be executed by the insufficiently secure service. This can be done again through the **sc** executable and the **config** option (binPath = will go next).

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.

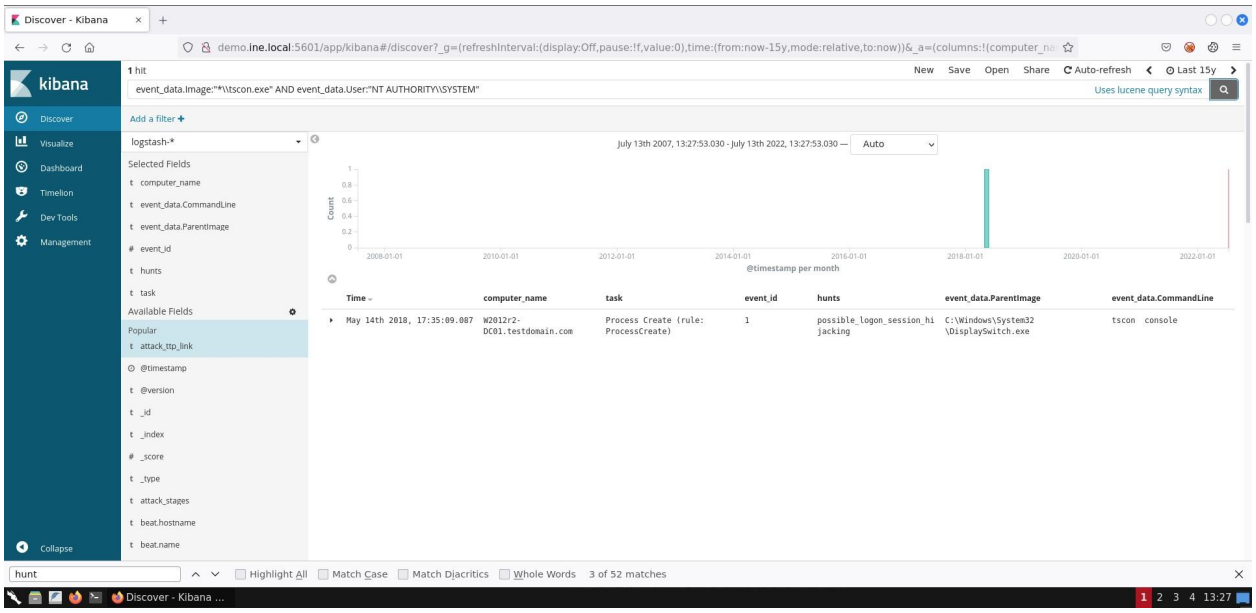
A viable search to identify possible Windows session hijacking via the described attacker technique is the below.

```
event_data.Image:"*\\tscon.exe" AND event_data.User:"NT AUTHORITY\\SYSTEM"
```

Alternatively, you can use the following search.

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

If you submit any of the searches above, you should see 1 hit.



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

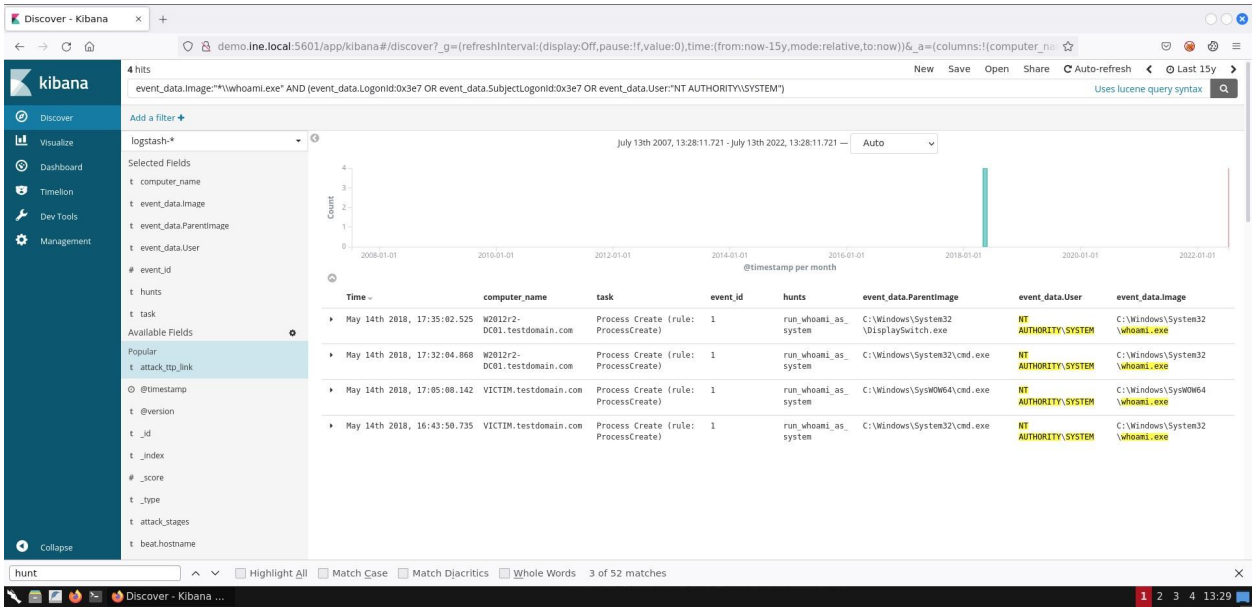
It is quite trivial to create a search to detect the whoami command being executed with SYSTEM-level privileges.

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.

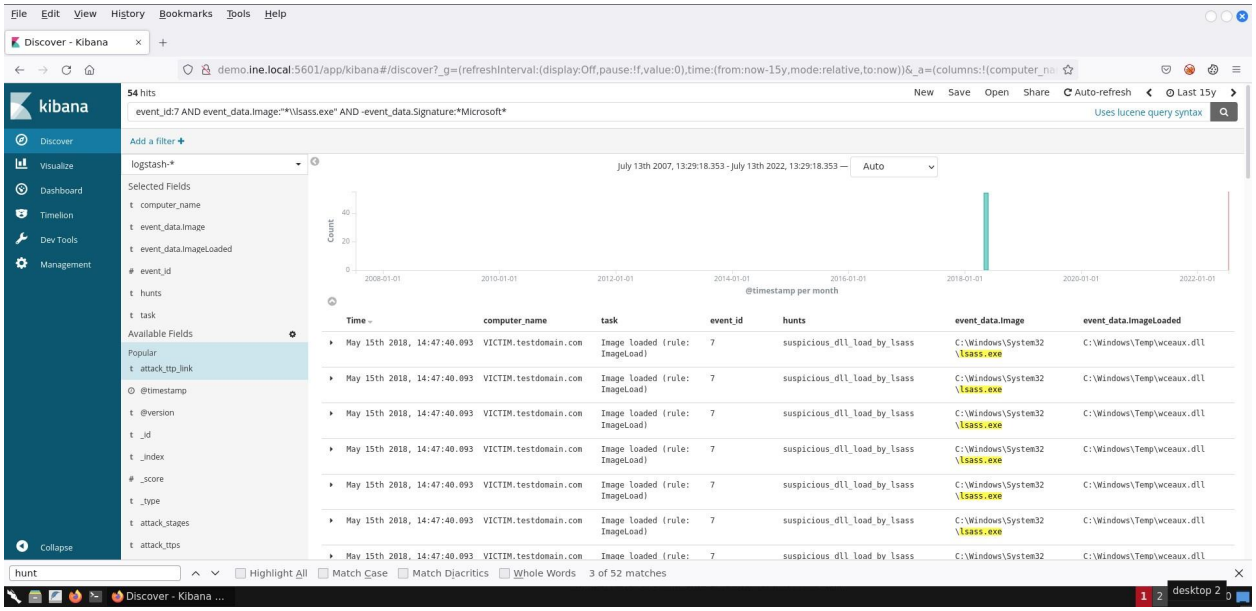
Task 9: Create a search to identify LSASS loading a library not signed by Microsoft

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_IOa0MiOHWndxWFRiUHNoNW8/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.

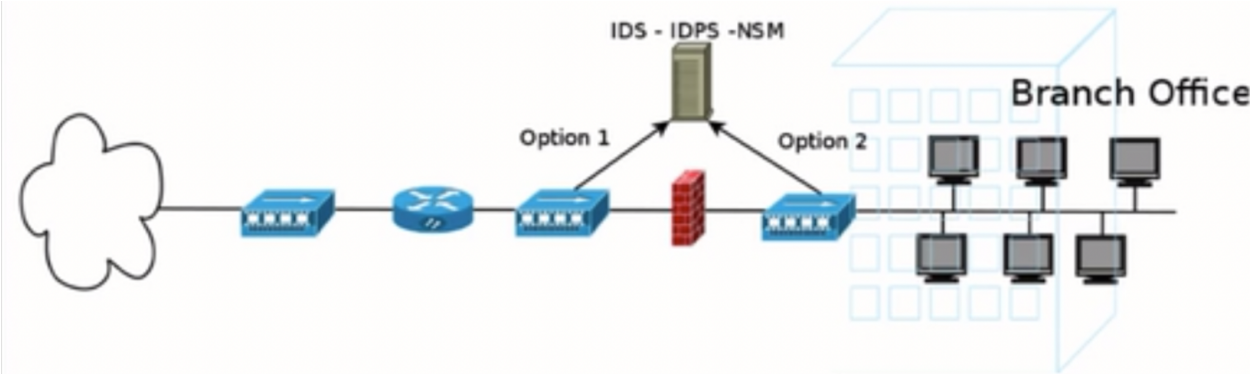
The most important Suricata features and capabilities are:

- 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

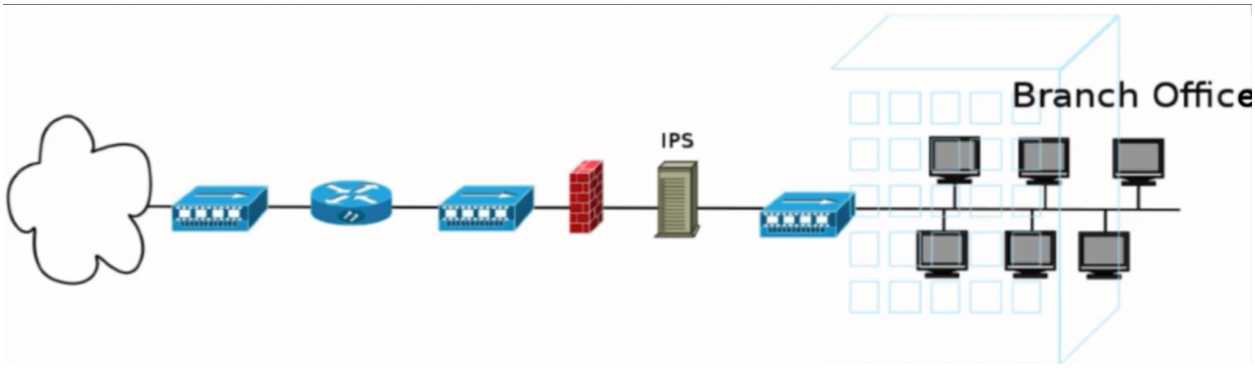
- IP Reputation
- JSON logging output
- Advanced protocol inspection
- Multi-threading

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

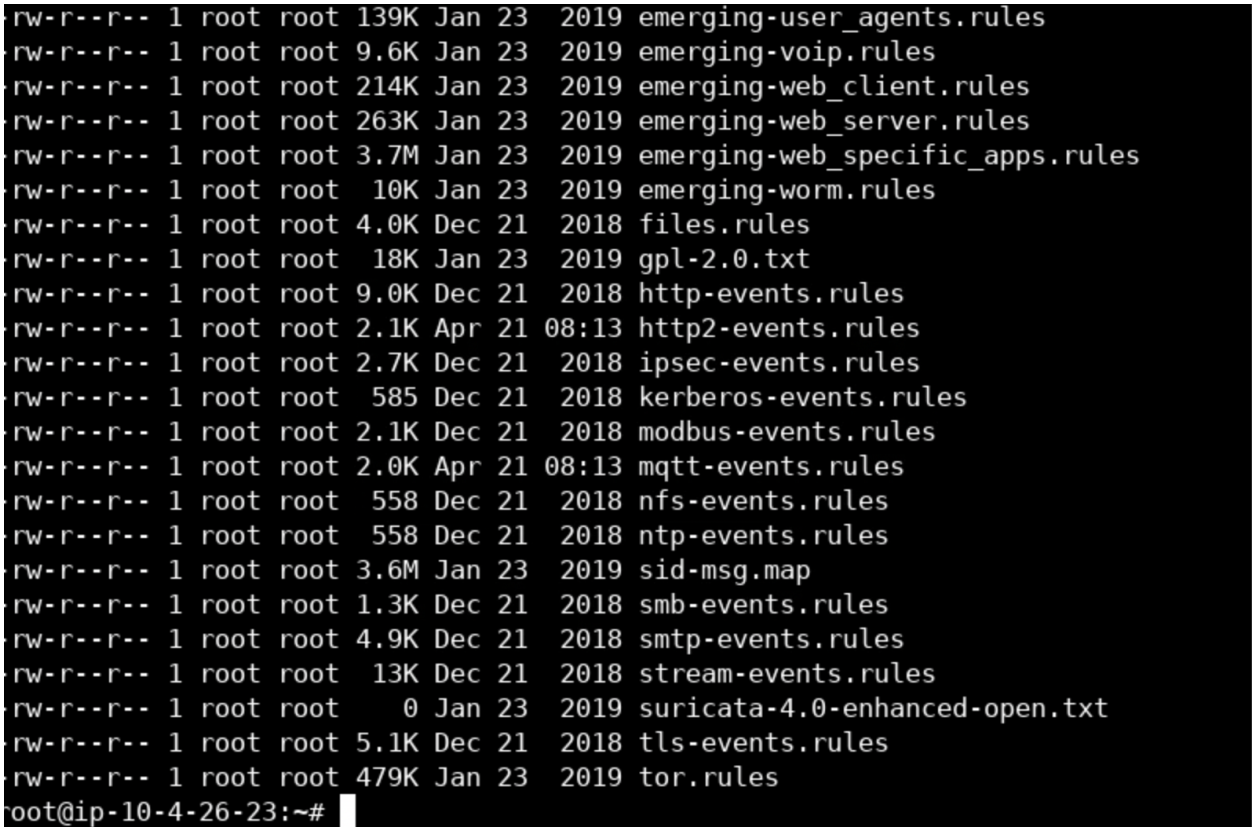
Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab.

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.



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 packets, 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 txcsum offloading
15/6/2022 -- 17:39:15 - <Info> - running in 'auto' checksum mode. Detection of i
nterface 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,
LRO: unset, TSO: unset, GSO: 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.
```

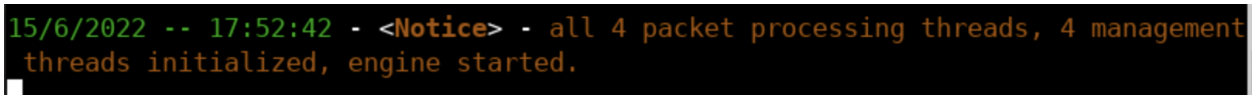
The -i option of Suricata chooses the best input option (for Linux the best input option is AF_PACKET). If you need pcap mode, it is better to use the --pcap option.

(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.



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
```

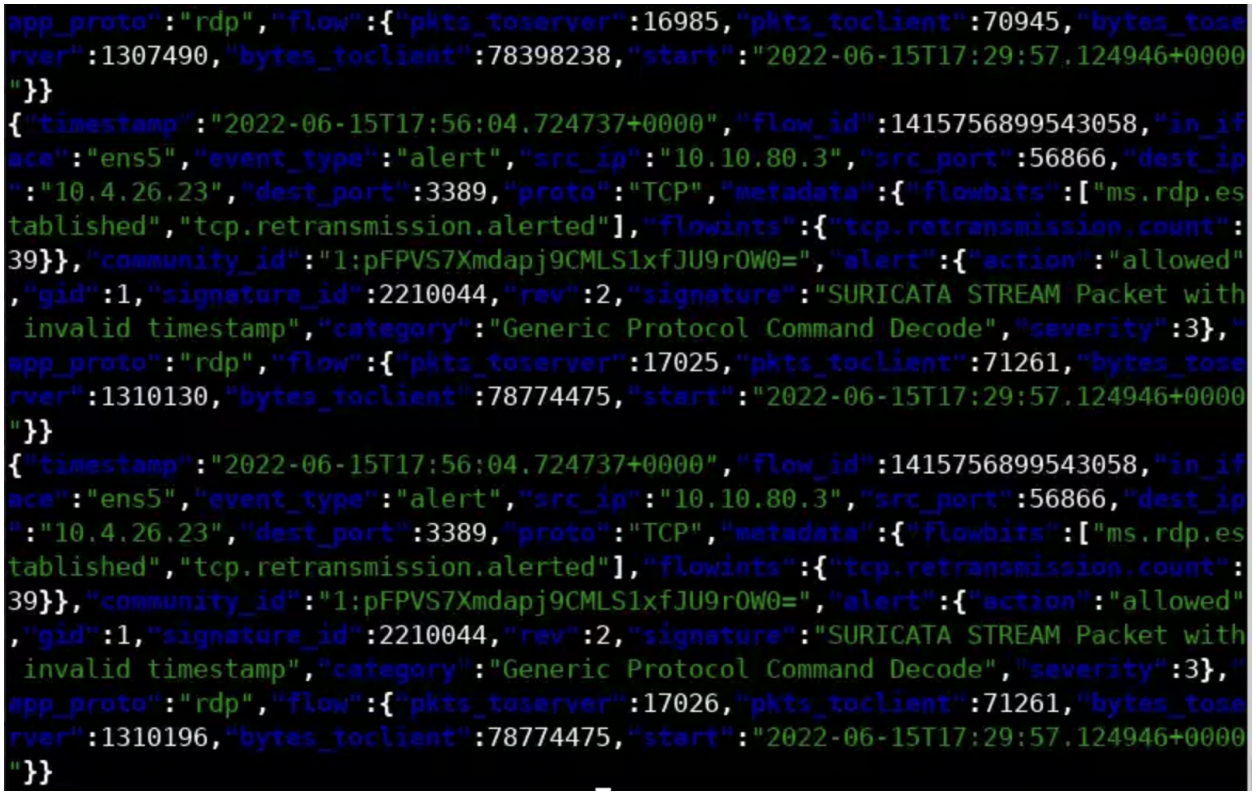
You should see something similar to the below.

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.



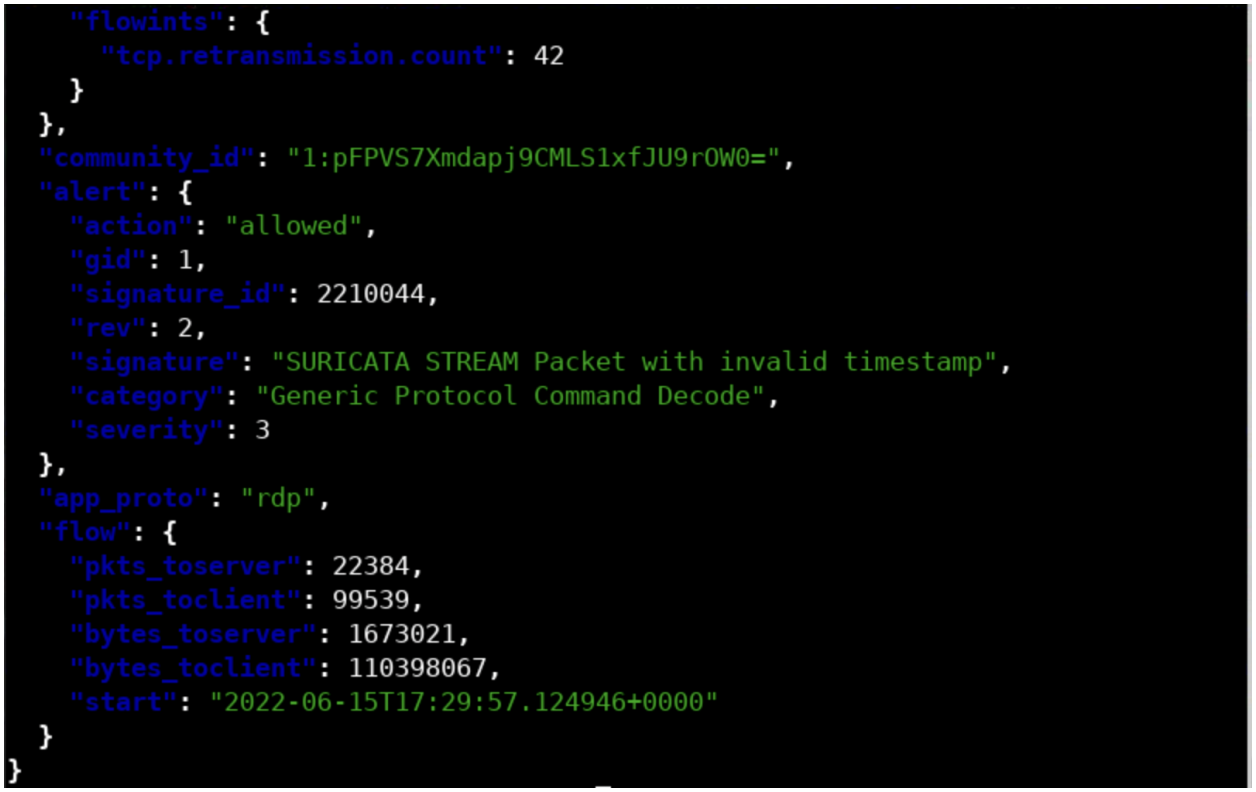
Notes:

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.



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

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

You should see the below.

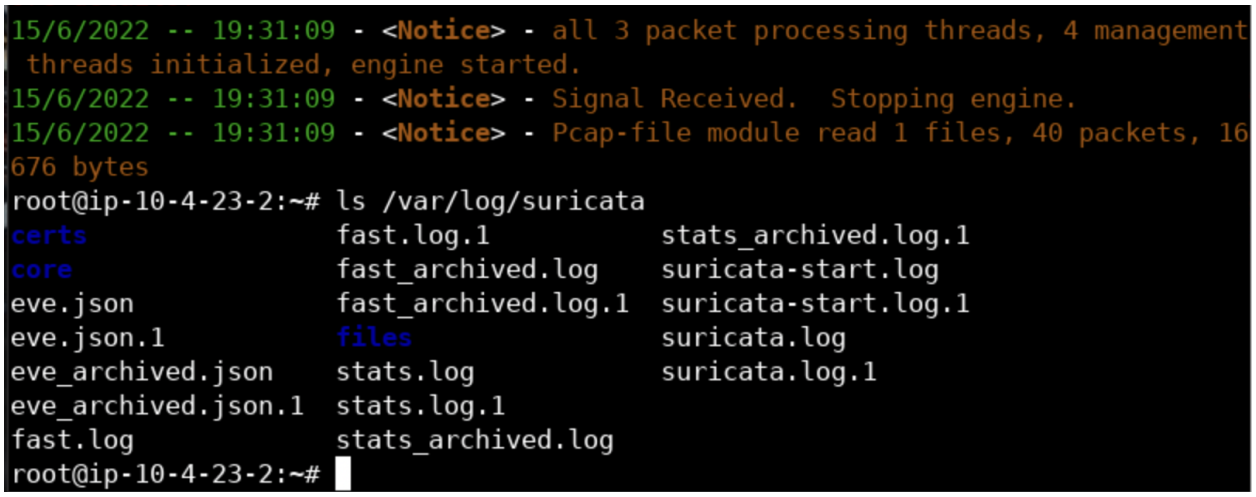
It should be noted that *eve.json* logs most of the event types. If you want a more targeted approach, you can disable EVE and enable specific outputs. Take for example HTTP events. The *http-log* has become obsolete with the introduction of EVE. You can still enable it though, as follows.

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.



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                 | Valu
e
-------------------------------------------------------------------------------
----
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 :

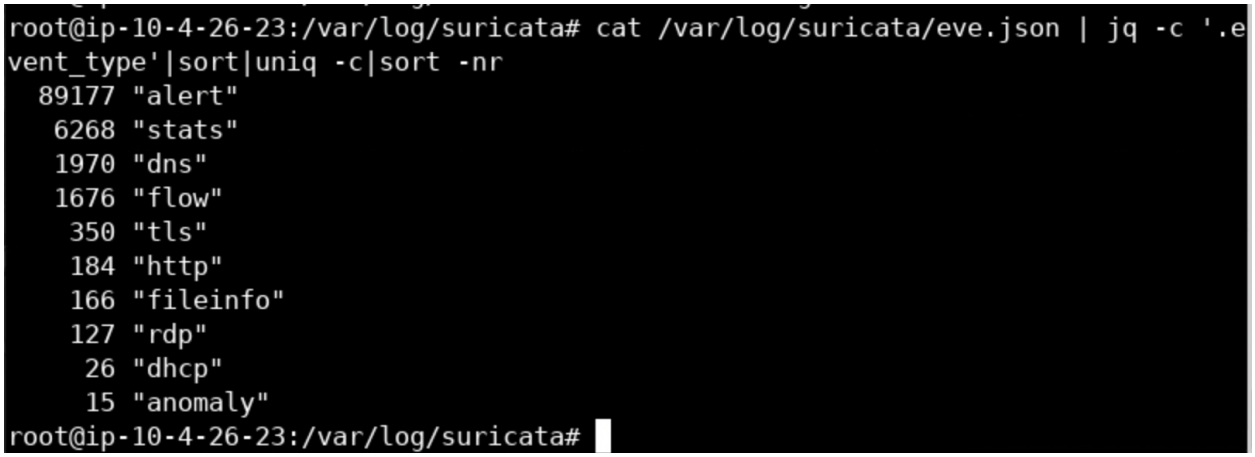
- pcap-log <- Logs all packets to PCAP files (full packet capture)
- alert-debug

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
```



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 .
```

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 'se
lect(.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.

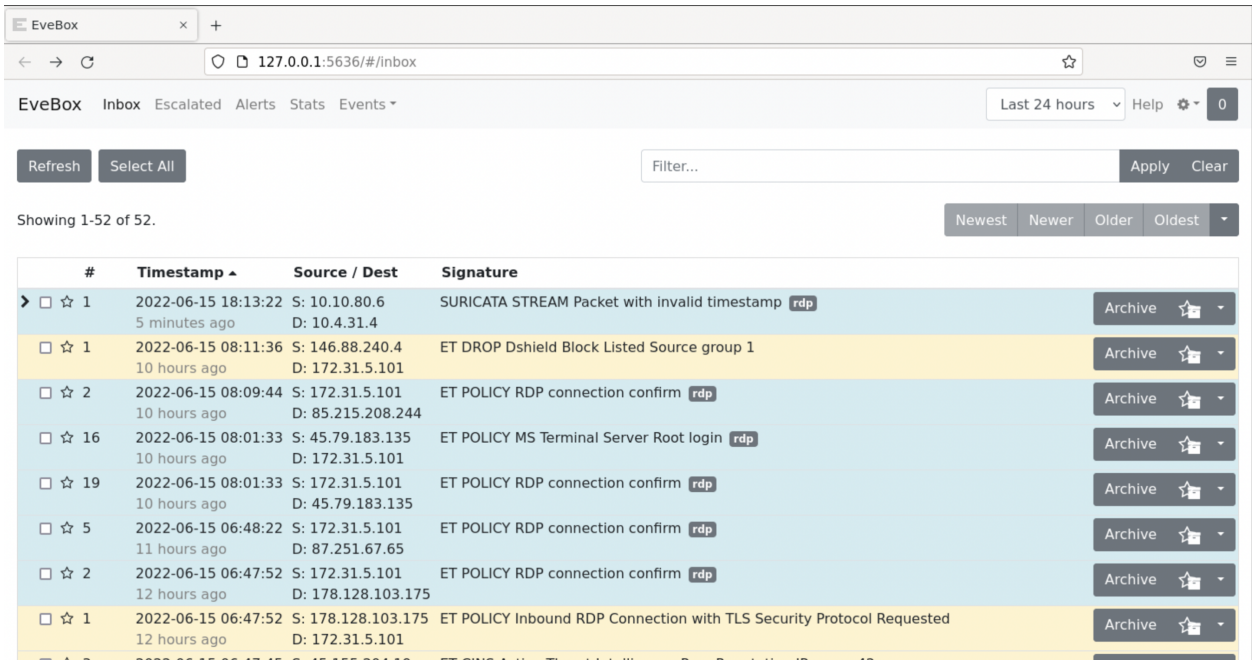
To analyze *eve.json* through EveBox, execute the below.

```
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 config
uration 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__I
nitial
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://12
7.0.0.1:5636
```

You will be presented with a web browser as shown.

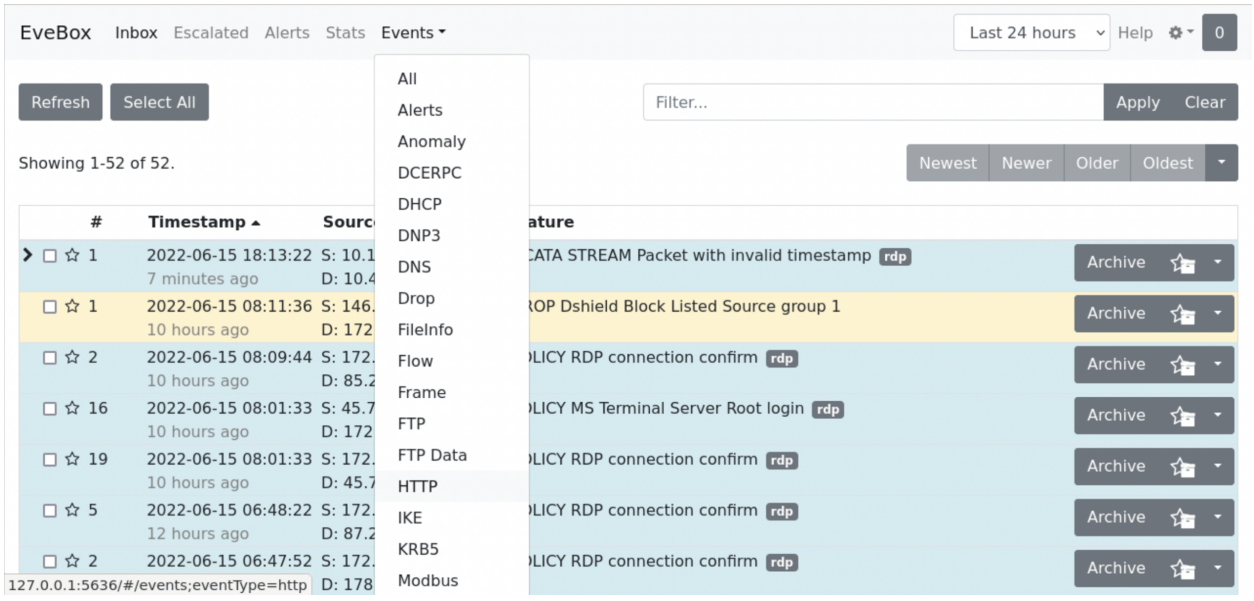


You will notice that EveBox parsed the whole *eve.json* and presented it to you in a much more organized way. Feel free

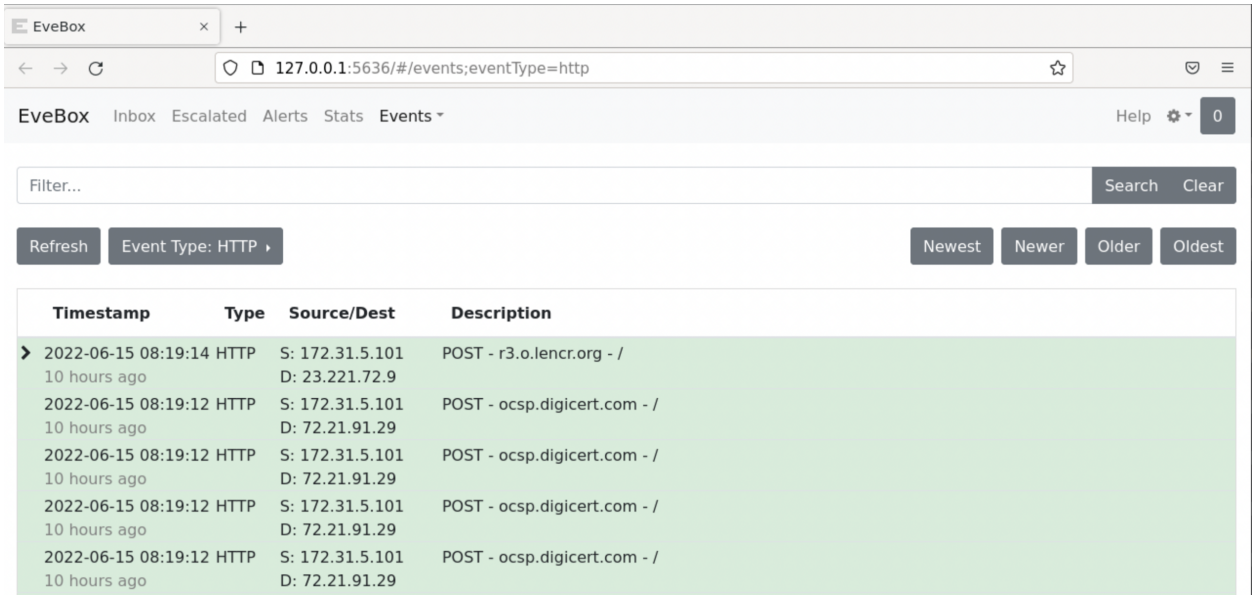
to look around and get familiar with EveBox's layout and capabilities.

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

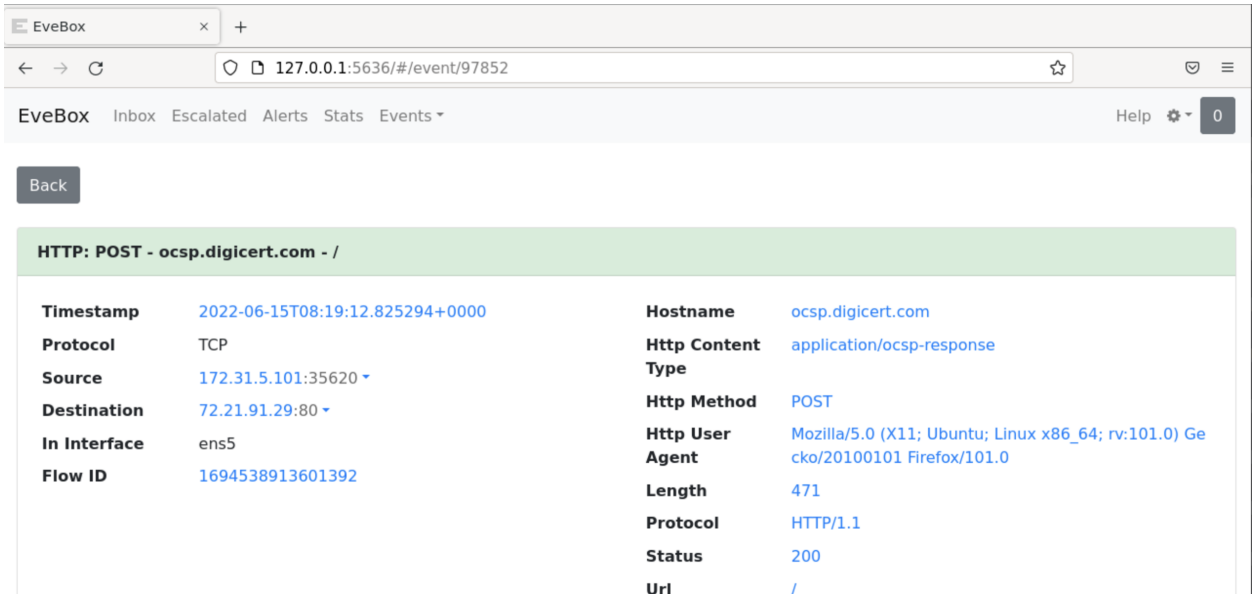
Try the following... First, list all HTTP events.



Then, choose the third entry.



Finally, click on the Flow ID.



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

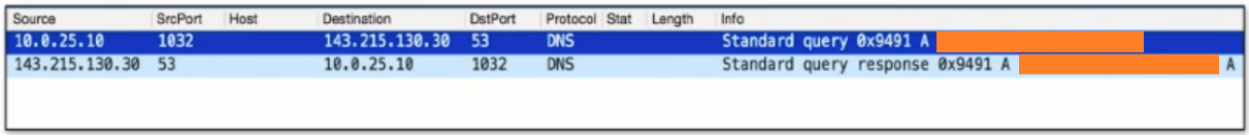
Then, we need a way to declare the **directionality of the traffic**; this can be done through:

- **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 -> $EXTERNAL_NET any*
    - Inbound traffic *$EXTENRAL_NET any -> $HOME_NET any*
    - Bidirectional traffic *$EXTENRAL_NET any <> $HOME_NET any*

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

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

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



Source	SrcPort	Host	Destination	DstPort	Protocol	Stat	Length	Info
10.0.25.10	1032		143.215.130.30	53	DNS			Standard query 0x9491 A
143.215.130.30	53		10.0.25.10	1032	DNS			Standard query response 0x9491 A 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 is should start matching. This option is used in conjunction with "*depth*". Examples:
        - *content:"whatever"; offset:5;*
        - *content:"|00 03|"; offset:3; depth:2;*
            - Content of hex |00 03| will be found 3 bytes in and 2 bytes deep
- **distance.** Informs the Suricata engine to look for the specified content n bytes relative to the previous match. This option is used in conjunction with "*within*". Examples:

    - *content:"whatever"; content:"something"; distance:5;*
    - *content:"whatever"; content:"something"; distance:0;*
    - *content:"|00 03|"; offset:3; depth:2;*
- *content:"whatever"; distance:0; nocase; within:30;*

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

- *content:"whatever"; nocase; depth:9; content:"some|20|thing"; distance:0;*
- We are looking for the string "whatever" (with nocase enabled) in the first 9 bytes of the packet. An offset of 0 is implied. Then, we are looking for the string "some thing" occurring anywhere after the first match of "whatever".

- *alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Malicious"; flow:established,to_server; dsize:45; content:"suspicious"; offset:33;*

- We are looking for outbound TCP traffic on any port with a size of 45 bytes. The string "suspicious" should start 33 bytes in the packet. 33+12=45. Essentially, we are looking for the string at the last 12 bytes.

Finally, let's dive into Rule metadata.

- **reference**. This is the first rule metadata field we usually come across. It indicates where the initial information to develop this rule came from. Example:

    - *reference:url,securingtomorrow.mcafee.com/2017-11-20-dridex;*
- **sid.** This is the signature ID number. It is a unique number given by the rule writer. Example:
    - sid:10000000;
- **revision.** This field informs about the version of the rule. Example:
    - rev:5;

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

- *pcre:"/something/flags";*
- *pcre:"/^\/[a-z0-9]{6}\.php$/Ui";*
- We are looking for 6 alphanumeric characters followed by *.php* and nothing after (the lowercase i indicates a case insensitive match, ^ indicates the start and $ indicates the end)

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

Managing Suricata Rules

- Suricata rules can be downloaded from:
- 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
- Wiresharkhttp://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.gooogle.com";
isdataat:!1,relative;
reference:url,threatintelprovider.com/trojanx;
classtype:trojan-activity; sid:1; rev:1;)
```

Rule 2:

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

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

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

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

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

SOUTIONS

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

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

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

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

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

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

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

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

- The main content of the rule starts with the *dns_query* keyword. *dns_query* is essentially a sticky buffer used to inspect the value of a DNS request. Being a sticky

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

- The remaining part can be easily comprehended.

Let's continue with **Rule 2**.

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

- This rule specifies the TLS protocol. *tls* is a Suricata protocol keyword that allows the engine to detect TLS traffic based on the protocol and not the port. By looking further down the rule, we are obviously checking for the delivery of a TLS certificate from an external server, hence the *$EXTERNAL_NET any -> $HOME_NET any* directionality. Once again, *any* is set for both ports since the *tls* keyword is utilized.

- Then, there is a typical message that will inform us about the detected activity. *flow:established,to_client; established* is used due to TCP and *to_client* due to the inbound traffic.

- *tls_cert_subject* is once again a buffer, and the rule content is a unique string included in the TLS certificate's CN portion.

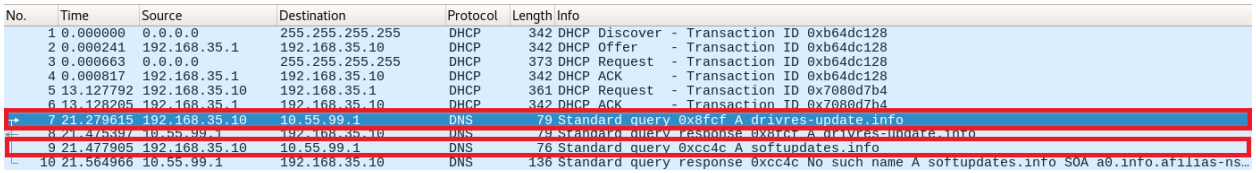
- The remaining part can be easily comprehended

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

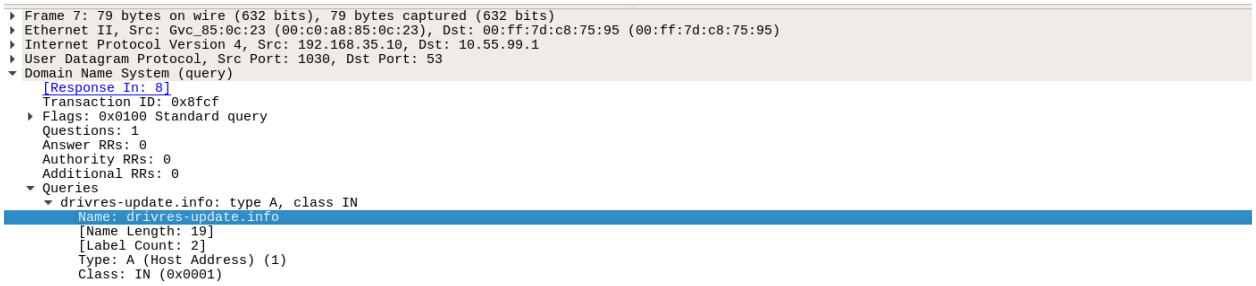
Let's start with *Sofacy.pcap*

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

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



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



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

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

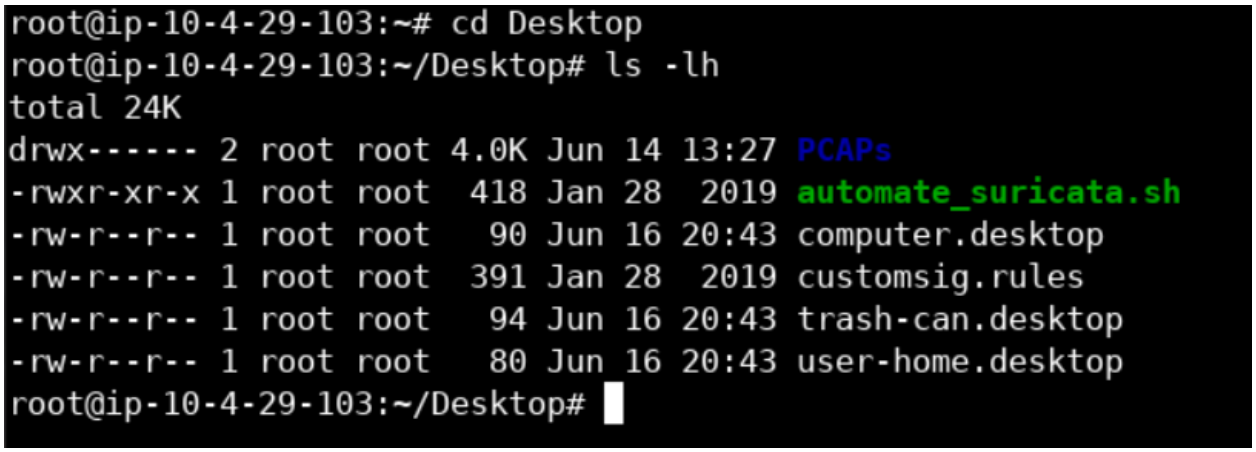
alert dns $HOME_NET any -> any any (msg:"TROJAN Activity Detected DNS Query to Known Sofacy Domain 1"; dns_query;

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

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

nano /etc/suricata/rules/local.rules

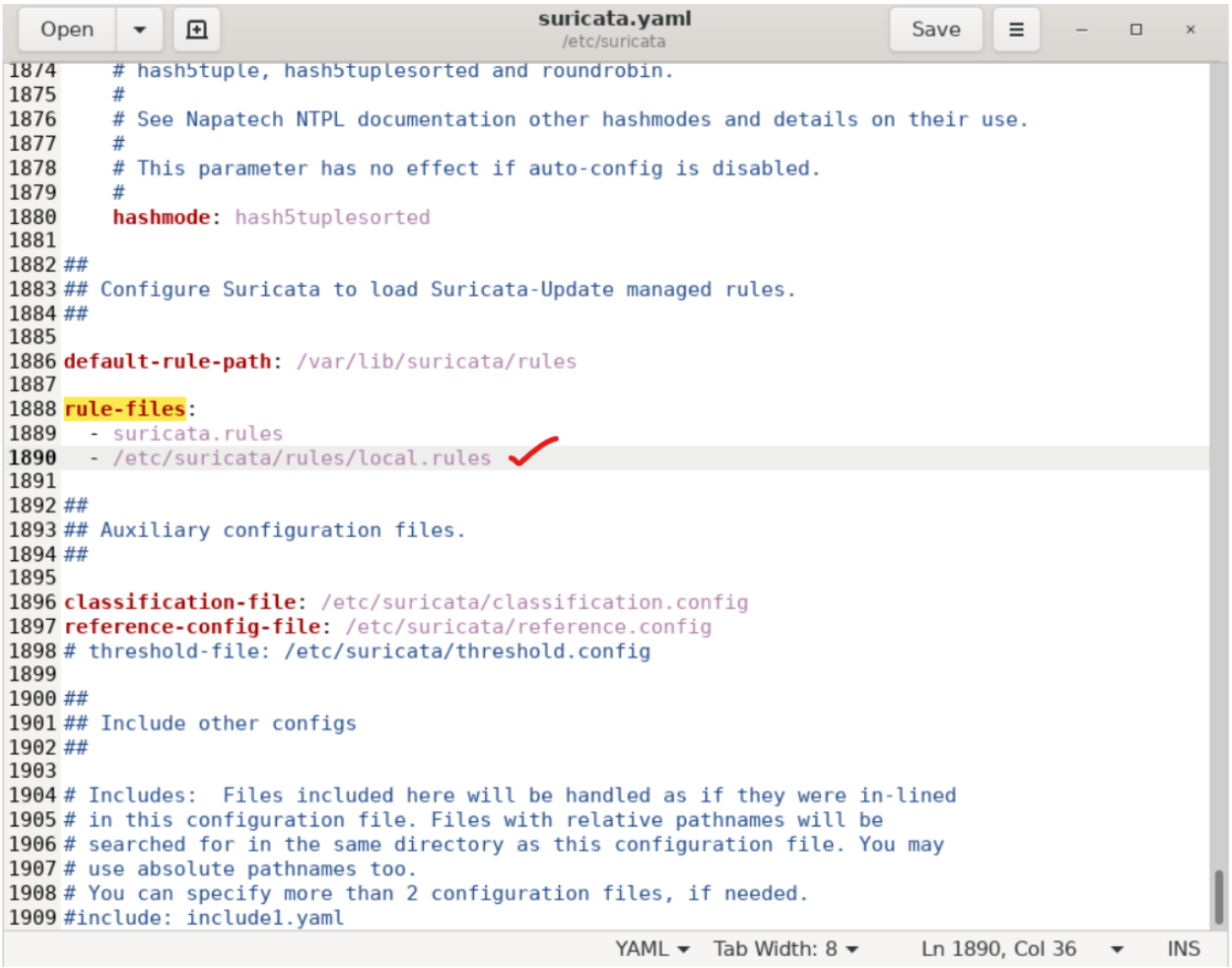
On the machine's Desktop, you will find a bash script (*automate_suricata.sh*), by OISF's J Williams, that will automate running Suricata with the *local.rules* file, logging in a different location (*/tmp/suricata*), echoing the contents of *fast.log* and also cleaning previous logs before each new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



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.



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

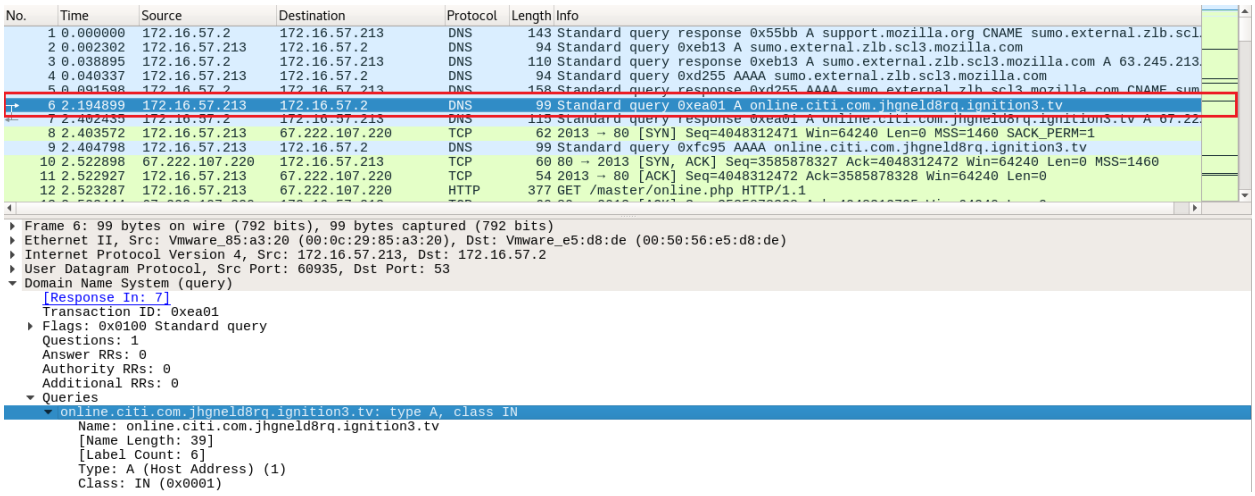
```
cd Desktop
./automate_suricata.sh ./PCAPs/Sofacy.pcap
```

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 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 -- 10:59:26 - <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 -- 10:59:26 - <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 -- 11:00:09 - <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 [**] [Classific
ation: (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 [**] [Classific
ation: (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).

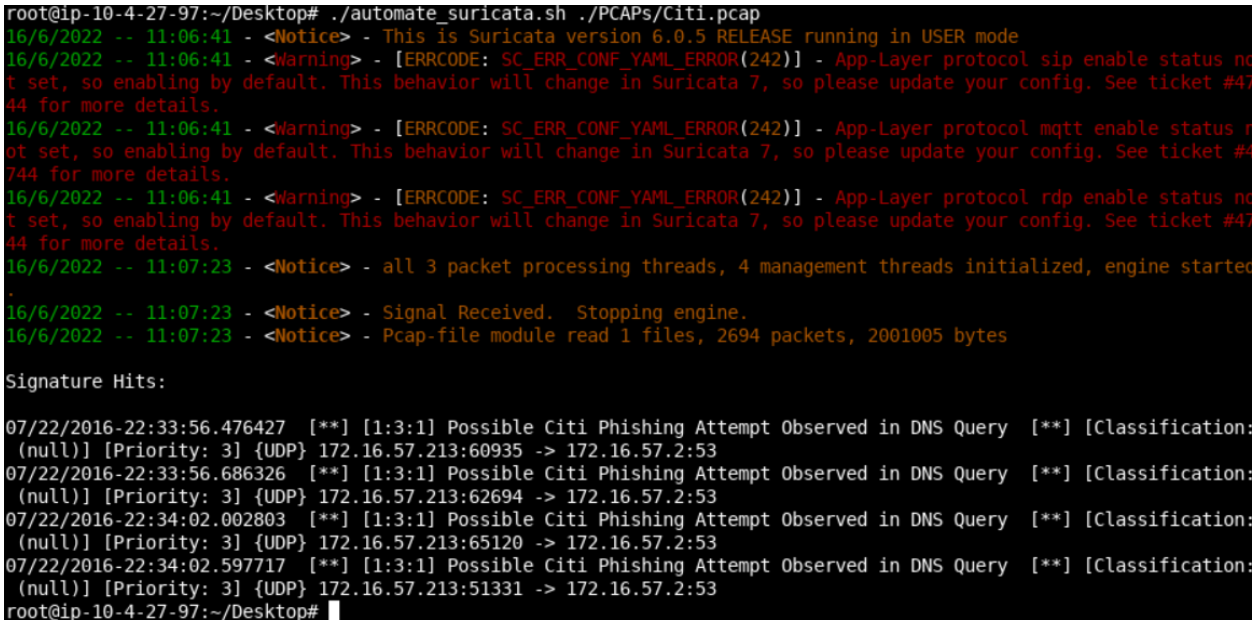


Let's create a Suricata rule to detect such phishing-related DNS traffic against online.citi.com.

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.



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

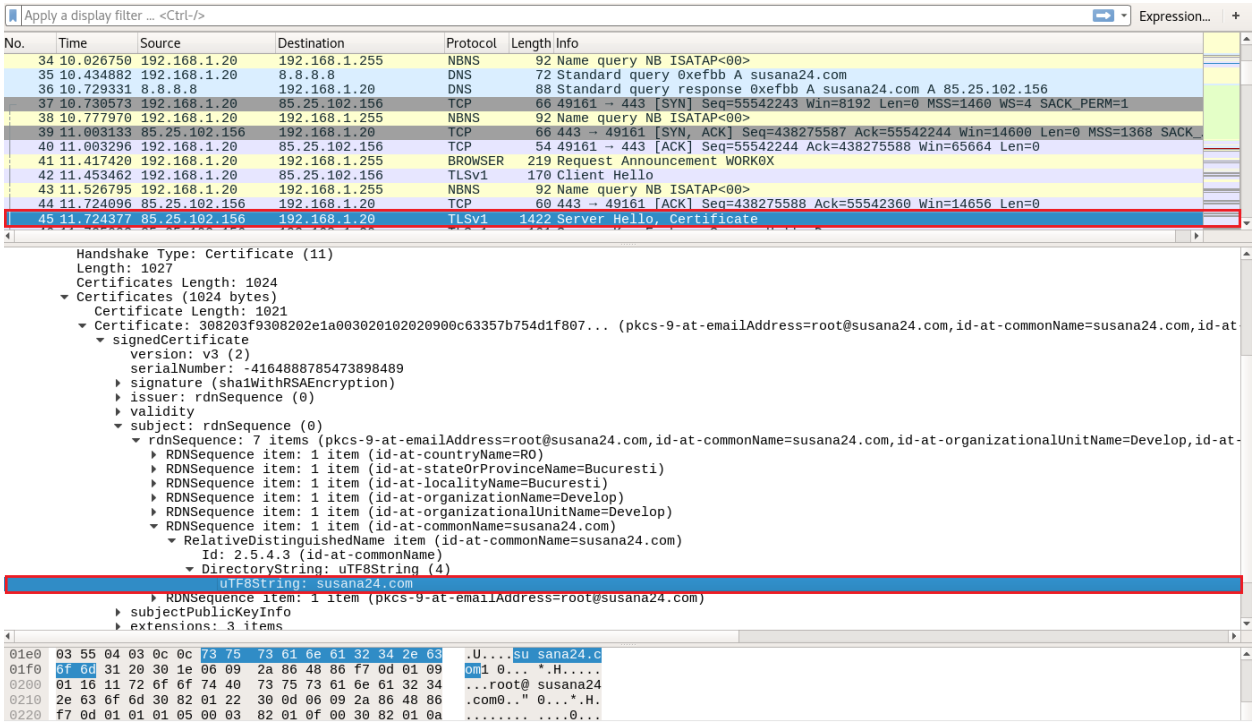
To learn more about Qadars, refer to the following links:

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

- https://sslbl.abuse.ch/ssl-certificates/sha1/1862c777babf298fe5a93406e4dc8456d718abcf/

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

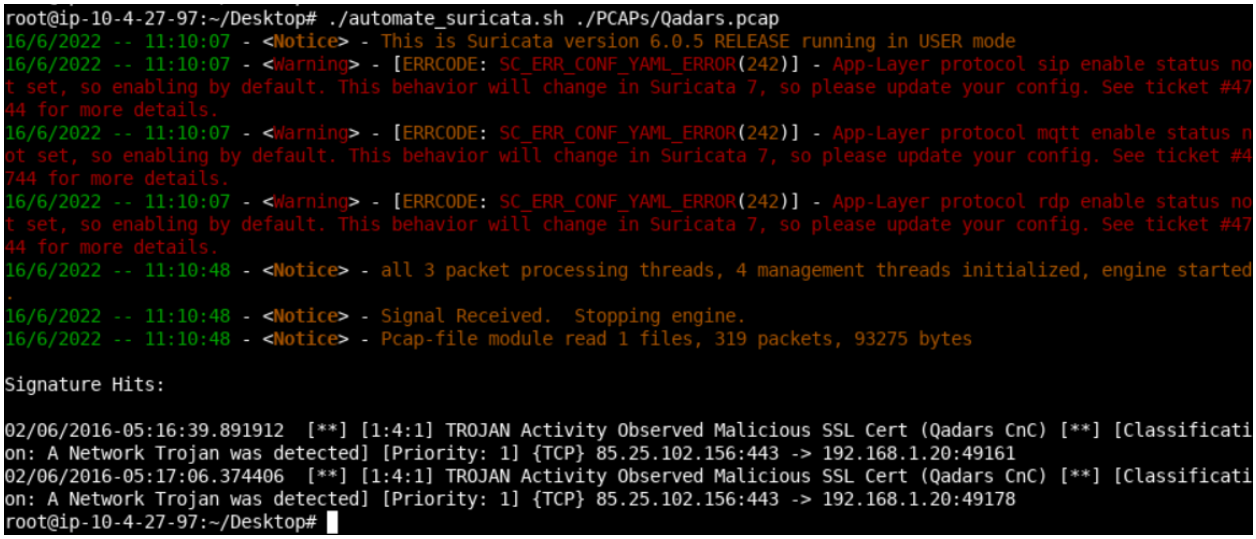
By opening *Qadars.pcap* in Wireshark, the first thing we notice is a DNS query and a DNS response regarding susana24.com. Then, we notice TLS traffic initiating.



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.



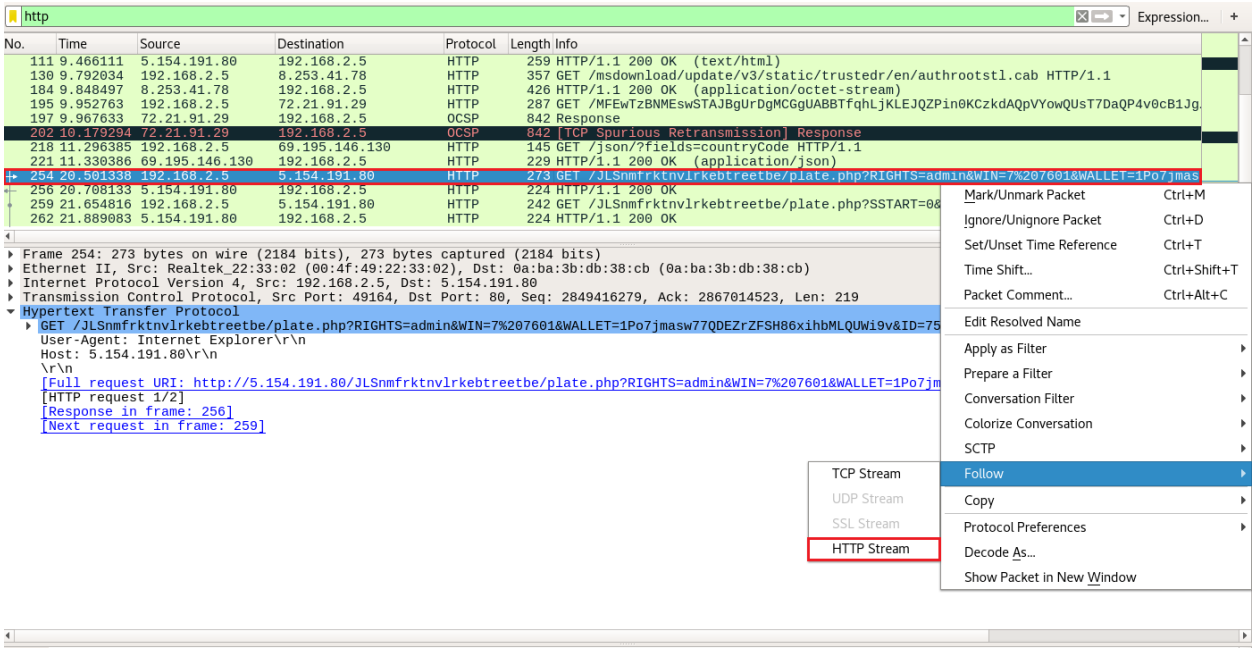
Now, it's time to analyze the *7ev3n.pcap*

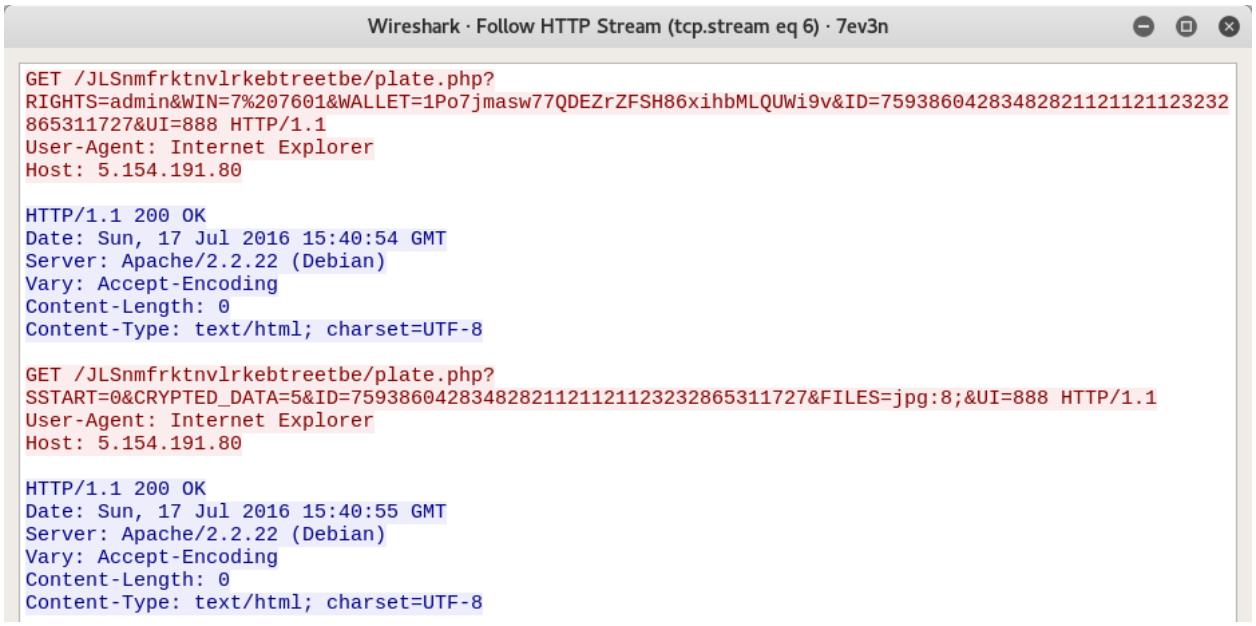
To learn more about the 7ev3n ransomware, refer to the following link https://www.vmray.com/cyber-security-blog/7ev3n-honet-ransomware-rest-us/.

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

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

Let's follow the whole stream.





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
```

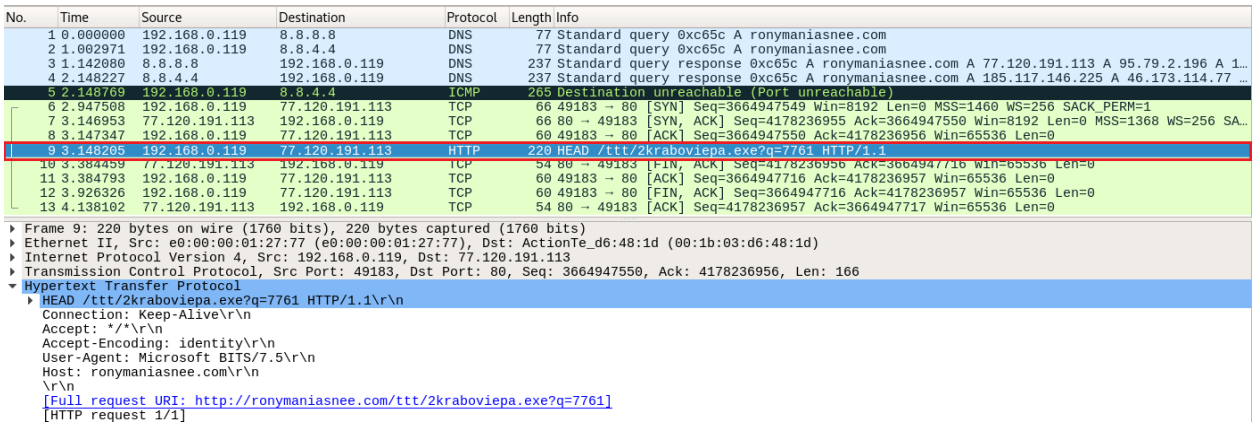
You should see the following.

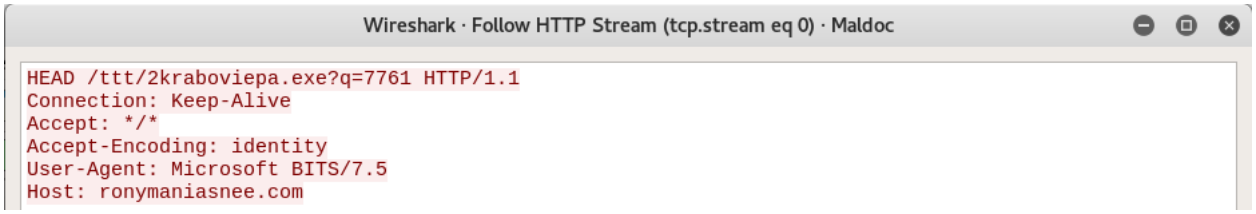
```
16/6/2022 -- 11:15:48 - <Notice> - Pcap-file module read 1 files, 296 packets, 97119 bytes

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 [**] [Classificatio
n: 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)] [Priorit
y: 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 [**] [Classificatio
n: 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.





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.

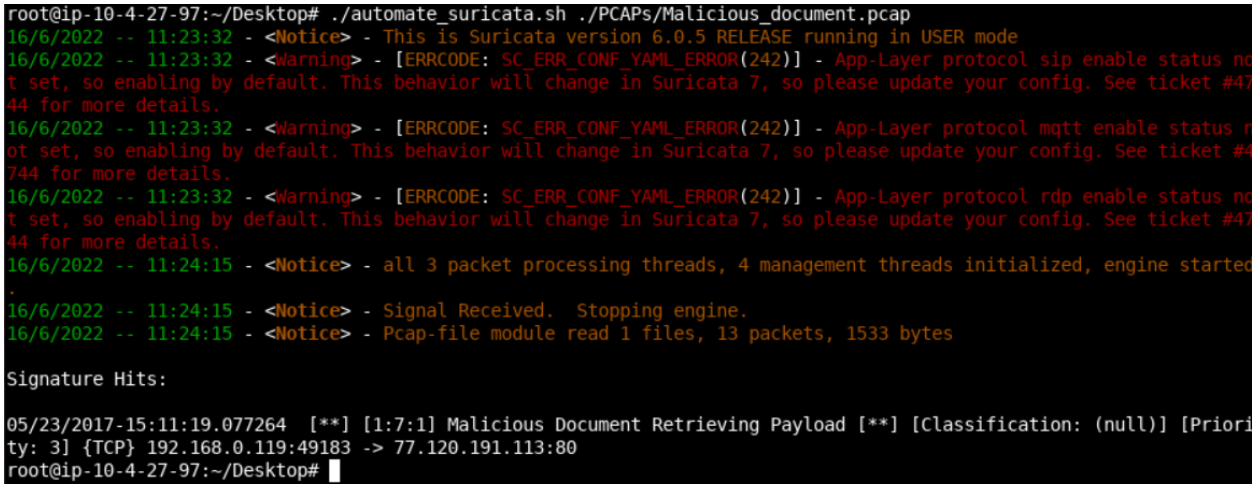
```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"Malicious Document Retrieving Payload"; flow:established,to_server; content:".exe?q="; fast_pattern; content:"Microsoft BITS/"; http_user_agent; depth:15; pcre:"/\.exe\?q=[0-9]{3,5}$/U"; http_header_names; content:!"Referer"; sid:7; rev:1;)
```

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.

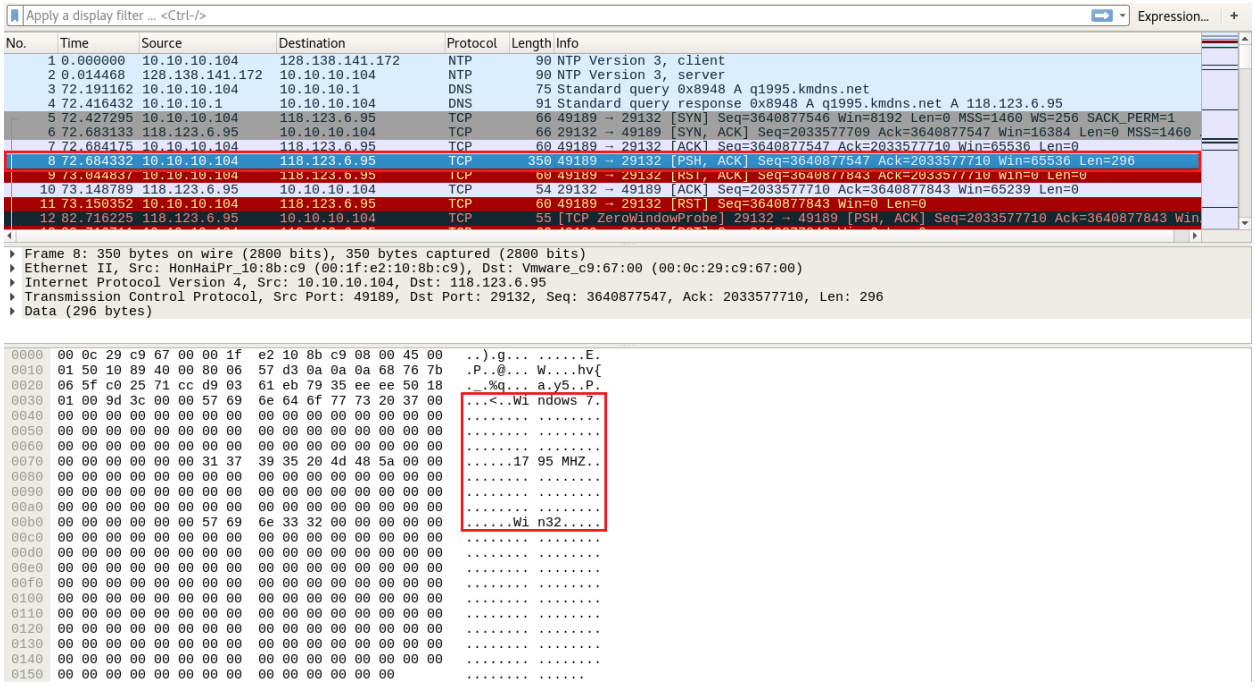


Task 3: Analyze the provided PCAP files and develop your own rules (Level: Intermediate)

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.



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

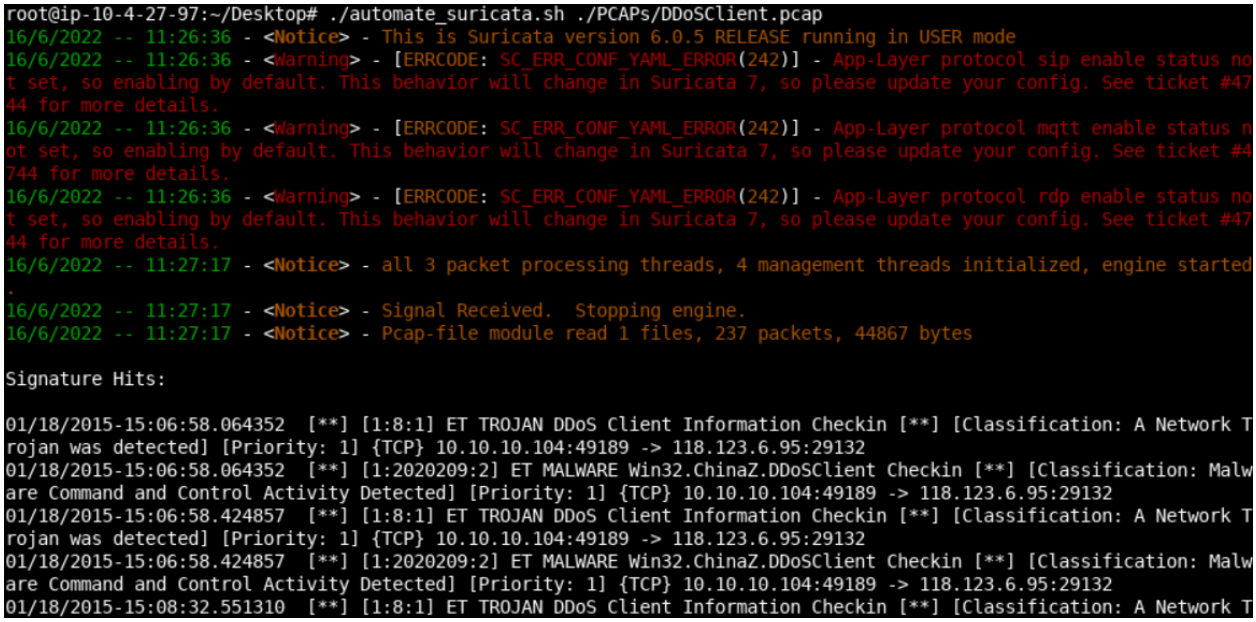
```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN DDoS Client Information Checkin"; flow:established,to_server; content:"Windows"; nocase; depth:7; content:"MHZ|00 00 00 00 00 00|"; distance:0; nocase; content:"|00 00 00 00 00 00|Win"; distance:0; nocase; classtype:trojan-activity; sid:8;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

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

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

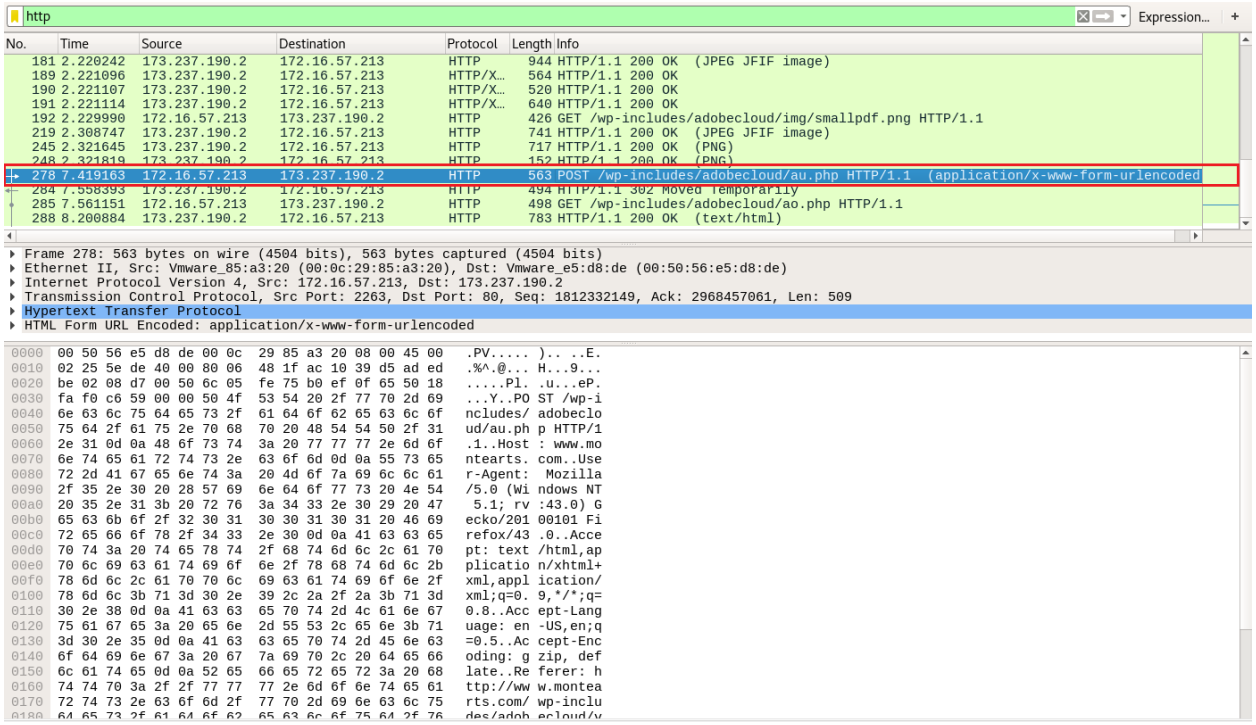
```
cd Desktop
./automate_suricata.sh ./PCAPs/DDoSClient.pcap
```

You should see the following.

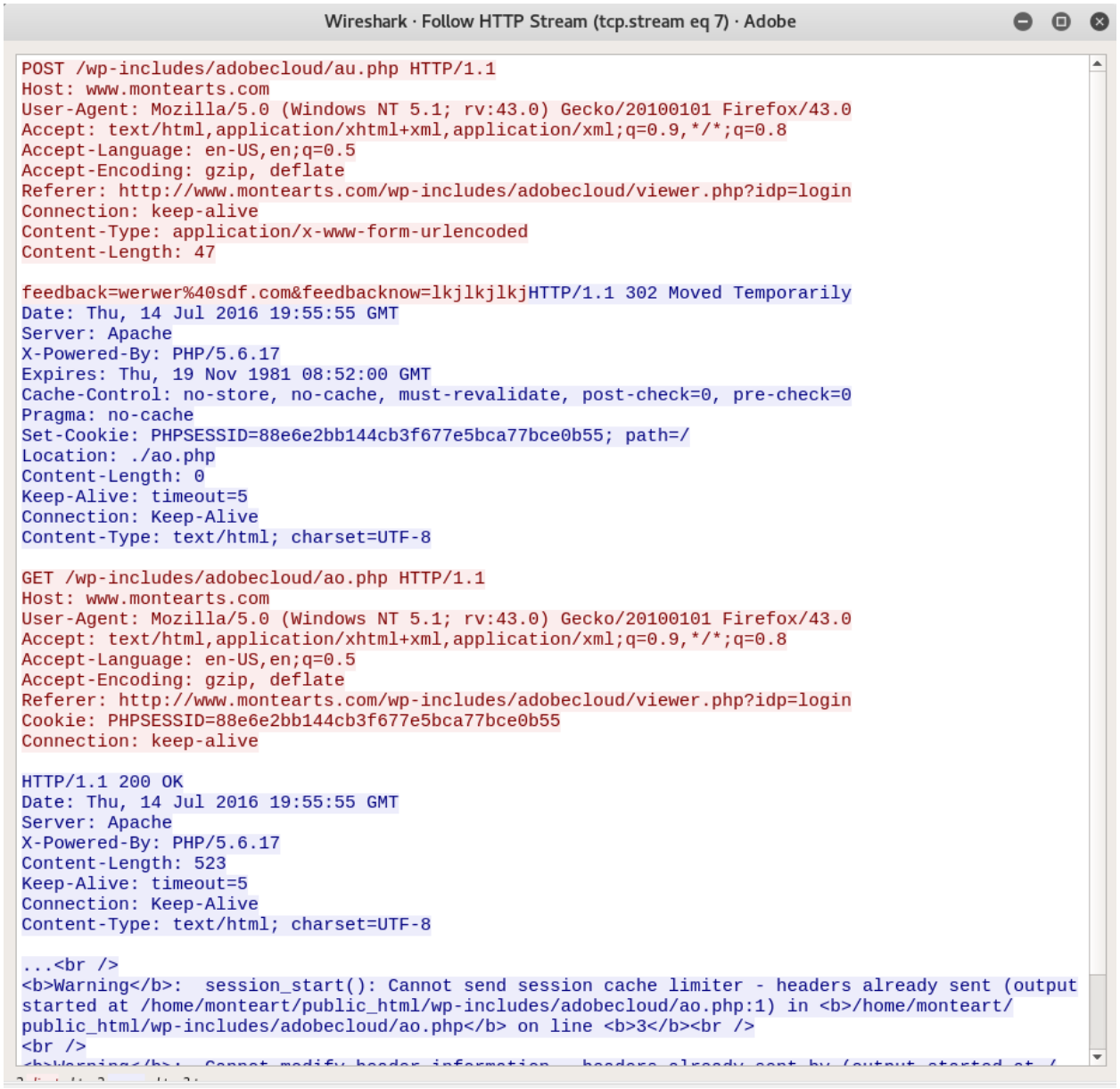


Now, it's time to analyze the *Adobe.pcap*

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

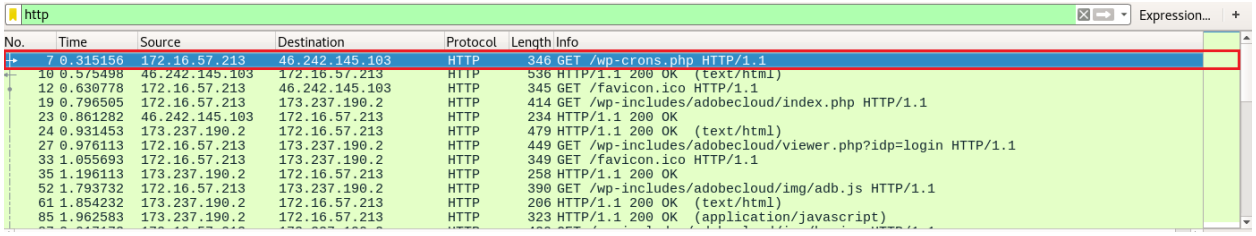


By opening *Adobe.pcap* in Wireshark and filtering so that we can see HTTP traffic only, the first thing we notice is this curious-looking POST request. Let's follow the whole stream.



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.



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

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

```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"Adobe
Phising Attempt"; flow:established,to_client; content:"200";
```

http_stat_code; http_content_type; content:"text/html"; nocase; file_data; content:"<META HTTP-EQUIV="; nocase; within:100; content:"refresh"; distance:1; nocase; within:7; content:"self.location.replace"; nocase; distance:0; content:"window.location"; nocase; distance:0; classtype:bad-unknown; sid:9; rev:1;)

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

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

cd Desktop
./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 Phising 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 Phising 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:

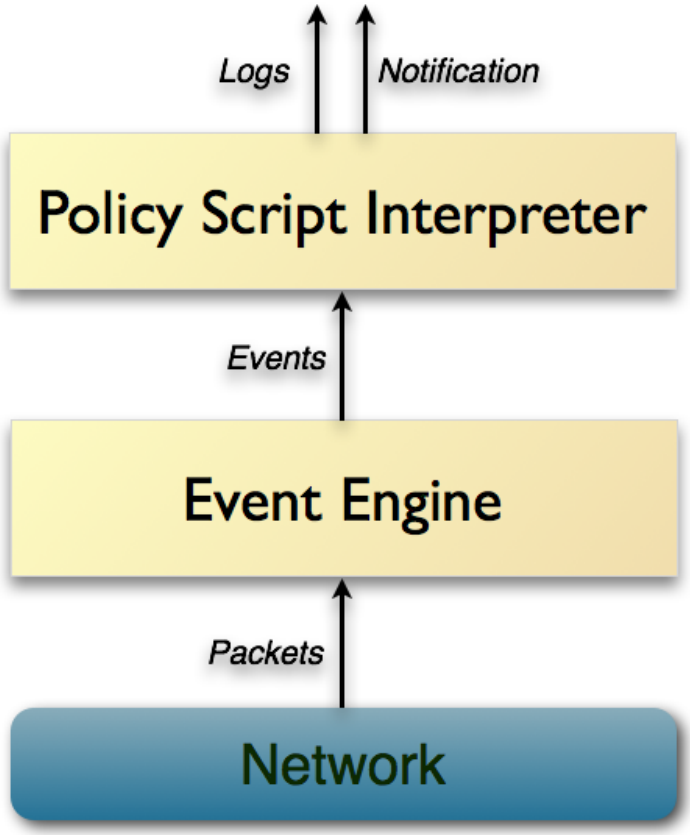
- Comprehensive logging of network activity
- Analysis of application-layer protocols (regardless of the port, covering protocols like HTTP, DNS, FTP, SMTP, SSH, SSL, etc.)
- Ability to look into file content exchanged over application-layer protocols
- IPv6 support
- Tunnel detection and analysis
- 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.

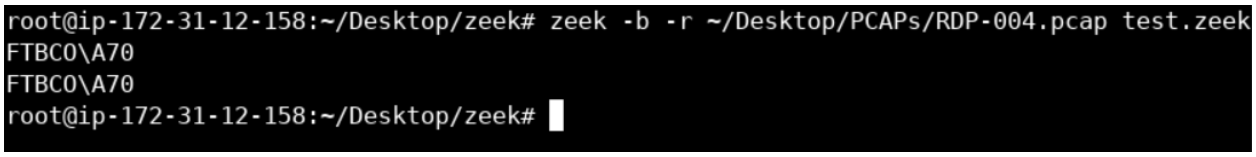
```
zeek -b -r ~/Desktop/PCAPs/RDP-004.pcap test.zeek
```

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.



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$start_time );
    print fmt("New RDP Requst from: %s at %s by user %s",
c$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 Requst from: 172.21.128.16 at 2007-10-26 03:36:34 by user FTBCO\A70
New RDP Requst from: 172.21.128.16 at 2007-10-26 03:36:37 by user FTBCO\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.

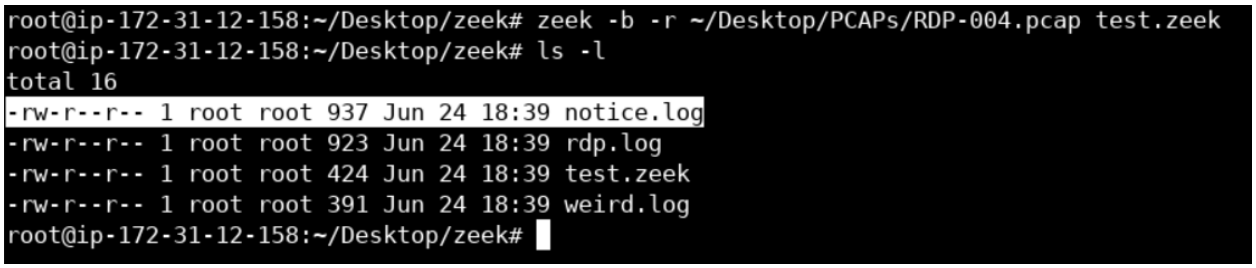
```
@load base/protocols/rdp
@load base/frameworks/notice
export { redef enum Notice::Type += {Suspicious_RDP}; }
event rdp_connect_request(c: connection, cookie:string){
    if (cookie != "admin"){
```

```
        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.



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

- Run the following command -
  cat notice.log

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

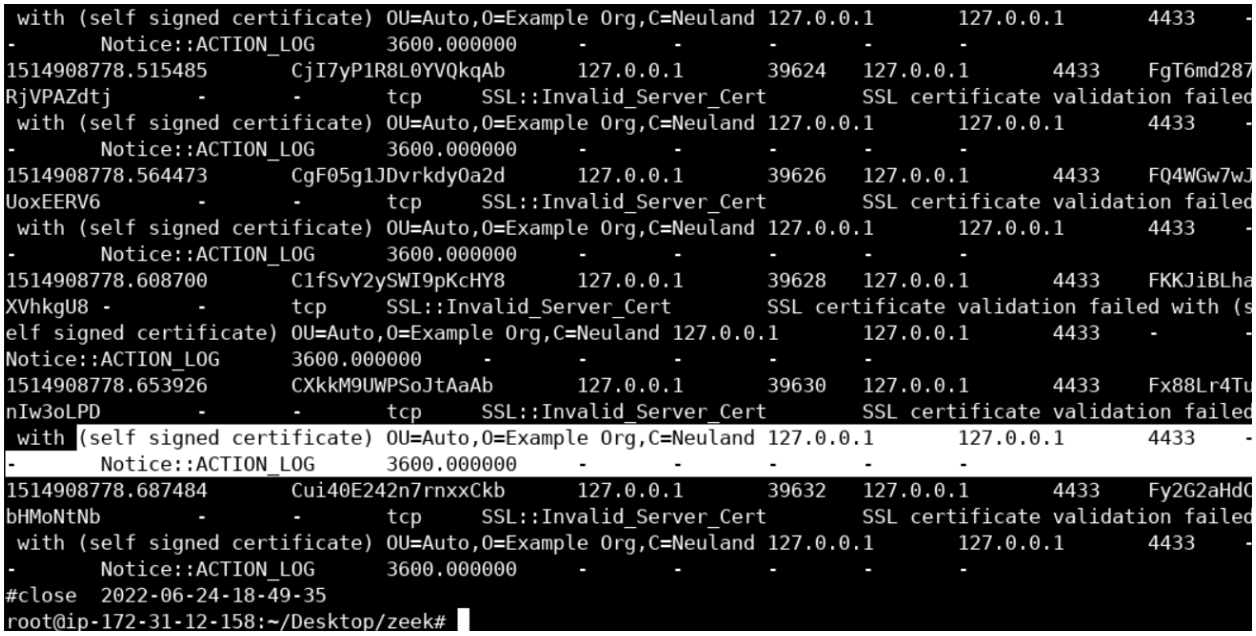
You should see something similar to the below.

```
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       CGAnmN1cZDF8Wz58Og      127.0.0.1       39500   127.0.0.1       4433    data_befo
re_established  -       F       zeek    TCP
1514908776.731924       C28Uj31oiTXEXX3lWg      127.0.0.1       4433    127.0.0.1       39552   connectio
n_originator_SYN_ack    -       F       zeek    TCP
1514908776.760200       C28Uj31oiTXEXX3lWg      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.



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.)

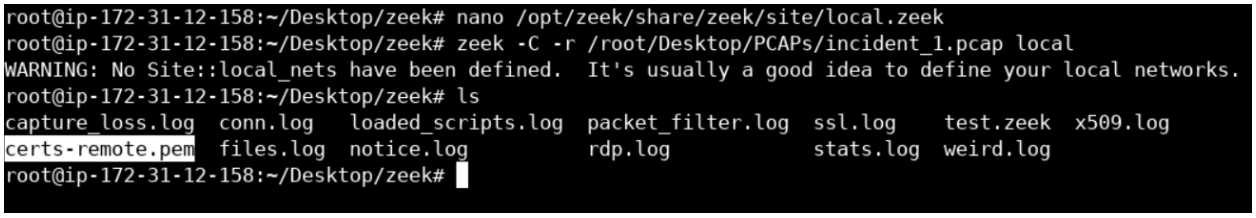
```
@load protocols/ssl/extract-certs-pem
redef SSL::extract_certs_pem = ALL_HOSTS;
```

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
```



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  3.pem   36.pem  42.pem  49.pem  55.pem  61.pem  68.pem  74.pem
11.pem  18.pem  24.pem  30.pem  37.pem  43.pem  5.pem   56.pem  62.pem  69.pem  75.pem
12.pem  19.pem  25.pem  31.pem  38.pem  44.pem  50.pem  57.pem  63.pem  7.pem   76.pem
13.pem  2.pem   26.pem  32.pem  39.pem  45.pem  51.pem  58.pem  64.pem  70.pem  77.pem
14.pem  20.pem  27.pem  33.pem  4.pem   46.pem  52.pem  59.pem  65.pem  71.pem  8.pem
15.pem  21.pem  28.pem  34.pem  40.pem  47.pem  53.pem  6.pem   66.pem  72.pem  9.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:
```

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.

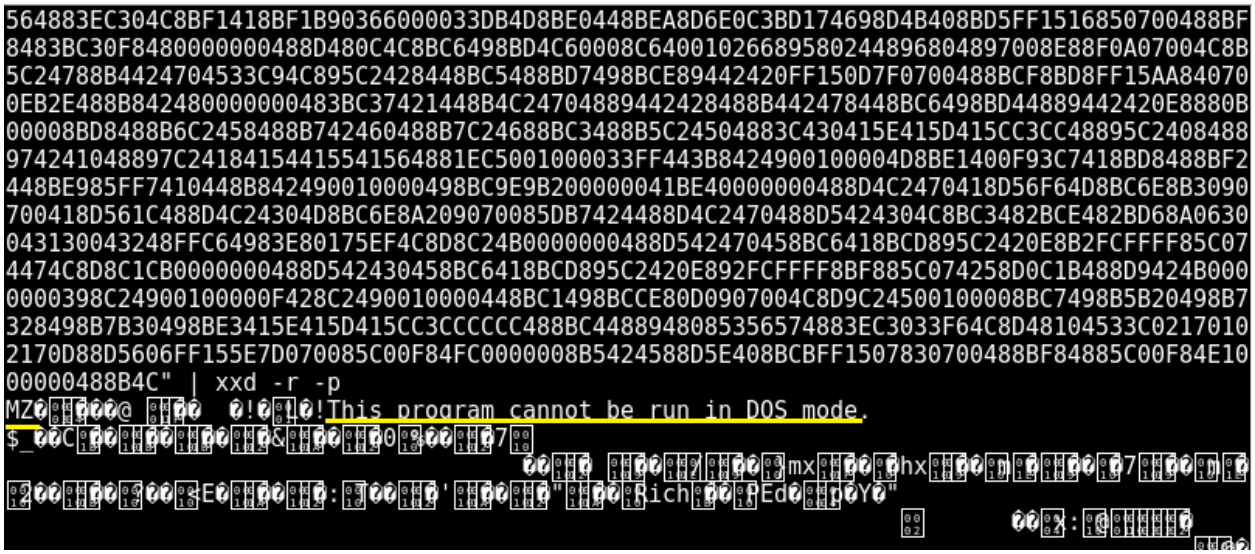
```
echo "copy-paste the whole X509v3 Subject Key Identifier section here" | sed 's/://g'
```

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.



The above means that a binary was stealthily being transferred, masked as an *X509v3 Subject Key Identifier*.

PCAP was taken from: https://github.com/fideliscyber/x509/

For more on the attacking technique being used, check out the following link: https://www.fidelissecurity.com/threatgeek/threat-intelligence/x509-vulnerabilities>

Task 3: Analyze the provided incident_2.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 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   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       CPlPJ24tgcnSmEU0c       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 pro
to 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 hos
t uri referrer
172.16.88.10    49493   172.16.88.135   80      f52pwerp32iweqa57k37lwp22erl48g63m39n60ou.net   /       -
172.16.88.10    49495   172.16.88.135   80      h54jtbqmuj56hwb48e41p42g33h34c29grbqfxm29.ru    /       -
172.16.88.10    49511   172.16.88.135   80      iqcqmrn30iuoubuo11crfydvkylrbtmtev.info /       -
172.16.88.10    49512   172.16.88.135   80      ezdsaqbulsgzh44m59p42eqmrkxa57n40brcq.com       /       -
172.16.88.10    49513   172.16.88.135   80      o41lwmqnqarmxiyi35iyftpzaye21osjyjq.ru  /       -
172.16.88.10    49516   172.16.88.135   80      n30arh24frisbslqmqoxgvpvk47o11pritev.biz        /       -
172.16.88.10    49517   172.16.88.135   80      jsa57n20hyisjxcre11fwl58gta37i65ovf32o51.info   /       -
172.16.88.10    49518   172.16.88.135   80      j36lxf52hsj56itc49lqayoveymwfzosi15jw.org       /       -
172.16.88.10    49519   172.16.88.135   80      g53lvo61ayoucrm49kzgvm69irhwl58erjwfu.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
m69e31iwiscth14c49hwcylxbyotiqgxoxlu.info
kvm49mynrd60l48lynre21hqfun20a47hyn20kq.org
htj56h34ewmzh44izn30nwcvg23bsb58irg63b18.net
asi55f32nyernygxjsbqk27pyewcygzo21ps.com
mydvhxdvh54i35ayc69mroyh54drmqcvpzoz.ru
lubqlze11bvovgub68jrazhxaqmwhrkqj46.com
gqe21muf32evntdvasd10j26k27pqlrbtosgx.net
kyoqpxg53nuf42g43oqo21l48a17d40o31k67j16h44.org
teredo.ipv6.microsoft.com
nxhyosg43a47exhum19g23f52fro21byayk57fs.info
dsmxgygrmud50pzj36hwpqazdrg43eyl38f12.biz
axgql48mql28h34k67fvnylwo51csetj16gzcx.ru
c69erb28g53ctdvkxk47cwixbqczcyc59dvk27.com
esmudsa57lul48o41f12mre41bxeygzo11f52pr.info
n50owhwguj66evkug33ewntn10n40puhtlxay.org
azn50i35btlsl48g33nre41g43ism39exc49lwn30.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    -
1502840909.945242       CYJQSIicJI6EovGTd       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       C8X1Us6sXGTAChqPk       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       CLOgXp2PSNU36sjIOf      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       CbRJWv41bx0KFuIBw1      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       CFe8vX1HJeXd6Gi2z6      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       CYRZPO10W273yFP2se      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       CZbdVD2vRLQFtUWB9i      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       CWKzCr4WW2pOrVHZhd      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       CvxSnK3bAKp5WN1Sm4      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       Cea6Np1zK4IY836HEd      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       CcXGGv2b6tzPQDEOy2      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    C=US    -       -       self signed certificate
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       FFNW1ARFu6xdtI9Nb       (empty) C=US    C=US    -       -       self signed certificate
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       FGX4Zy3lKccTXxDRkf      (empty) C=US    C=US    -       -       self signed certificate
1502840082.157036       CiZ3uCsuLN5YOykx4       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       FU9eex4e6KDOqXQ99c      (empty) C=US    C=US    -       -       self signed certificate
1502840087.172807       CVSjN4oksOMTdEf33       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       FWqVna47aShxs3Hymf      (empty) C=US    C=US    -       -       self signed certificate
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       FSdGwL2uByHQiVaav8      (empty) C=US    C=US    -       -       self signed certificate
1502840097.212504       Cy5ovX36VFZSOJUFt2      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       F1HHiz13povy069N4       (empty) C=US    C=US    -       -       self signed certificate
1502840102.229003       CZFMCkuQGO65qDxD1       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       F384UA43Kgl0meRQh1      (empty) C=US    C=US    -       -       self signed certificate
1502840107.245449       CJCzhI3Si7DuzeG24d      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       F07a9O2TjrupoF8AN7      (empty) C=US    C=US    -       -       self signed certificate
1502840112.261634       CBrqCS3o9leljvTAI4      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       FHhuKK2jiIZn90lw98      (empty) C=US    C=US    -       -       self signed certificate
1502840117.278666       CSfX5N2PSWCvRhlMp9      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       F6e9LH1de85Idq8Stk      (empty) C=US    C=US    -       -       self signed certificate
1502840122.291001       CsvBWw4AXVPqoVq9eh      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       F2pdoo3kpQQeCd8wL7      (empty) C=US    C=US    -       -       self signed certificate
1502840127.307952       CPUW2U2KFbHJYr2uHe      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    C=US    -       -       self signed certificate
```

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_
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]
1501895346.262554       C8sWF11B6cuVrpsD1b      10.200.201.29   47859   91.189.91.157   123     udp     ntp     0.023366        48      48      SF      -       -       0       Dd      1       7
6       1       76      -
1501895474.512333       CjwOYotUMsxegvvih       10.200.201.29   49777   91.189.91.157   123     udp     ntp     0.023506        48      48      SF      -       -       0       Dd      1       7
6       1       76      -
1501895634.848284       CvT75Co8O6E5PdIe9       10.200.201.29   59312   10.200.201.2    53      udp     dns     0.039017        122     272     SF      -       -       0       Dd      2       1
78      2       328     -
1501895634.887384       CnfDo2DhENiPRjU5f       10.200.201.29   54892   10.200.201.2    53      udp     dns     0.066114        68      218     SF      -       -       0       Dd      2       1
24      2       274     -
1501895634.953636       CZpU74gNSVkrk4PB9       10.200.201.29   44441   10.200.201.2    53      udp     dns     0.000361        122     272     SF      -       -       0       Dd      2       1
78      2       328     -
1501895634.954019       CLjFa331td6smSHVh5      10.200.201.29   37253   10.200.201.2    53      udp     dns     0.000249        68      218     SF      -       -       0       Dd      2       1
24      2       274     -
1501895730.762402       ClOpMV2XLLJ2PDg2z4      10.200.201.29   48860   91.189.91.157   123     udp     ntp     0.024368        48      48      SF      -       -       0       Dd      1       7
6       1       76      -
1501895827.639504       C95w5132RwebkUFmE3      10.200.201.29   40947   10.200.201.2    53      udp     dns     0.018622        122     272     SF      -       -       0       Dd      2       1
78      2       328     -
1501895827.658182       CMopT4Z8nFjU6vjj7       10.200.201.29   45353   10.200.201.2    53      udp     dns     0.029225        68      218     SF      -       -       0       Dd      2       1
24      2       274     -
1501895827.687506       CEvJ96iUUhJG6xp25       10.200.201.29   37574   10.200.201.2    53      udp     dns     0.000321        122     272     SF      -       -       0       Dd      2       1
78      2       328     -
1501895827.687847       CZD54r2JOWfMrwKdv8      10.200.201.29   37385   10.200.201.2    53      udp     dns     0.000262        68      218     SF      -       -       0       Dd      2       1
24      2       274     -
1501895964.704770       CacMIl1mtWDOGOoVJj      10.200.201.29   44851   10.200.201.2    53      udp     dns     0.024651        122     272     SF      -       -       0       Dd      2       1
78      2       328     -
1501895964.729523       CvcIzj2R5XEkYuvSd9      10.200.201.29   48897   10.200.201.2    53      udp     dns     0.059462        68      218     SF      -       -       0       Dd      2       1
24      2       274     -
```

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;
};
```

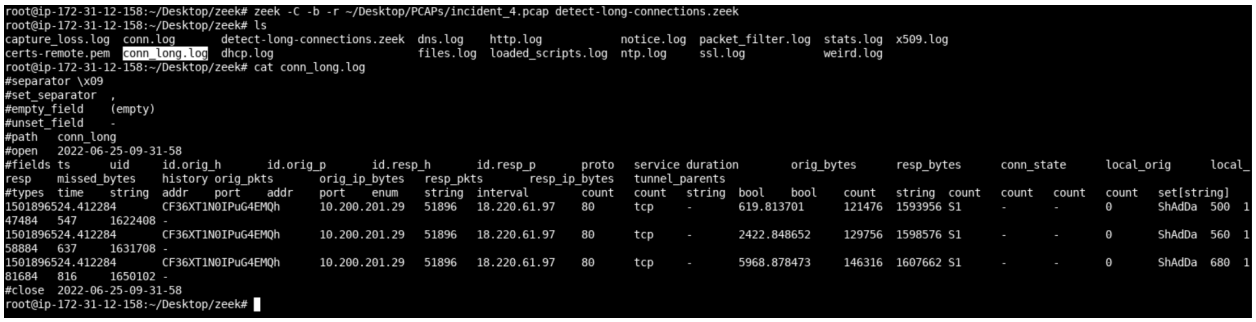
```
event zeek_init() &priority=5
    {
    Log::create_stream(LOG, [$columns=Conn::Info,
$path="conn_long"]);
    }
function get_durations(c: connection): Durations
    {
    local check_it: Durations;
    if ( c$id$orig_h in special_cases )
        check_it = special_cases[c$id$orig_h];
    else if ( c$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
```

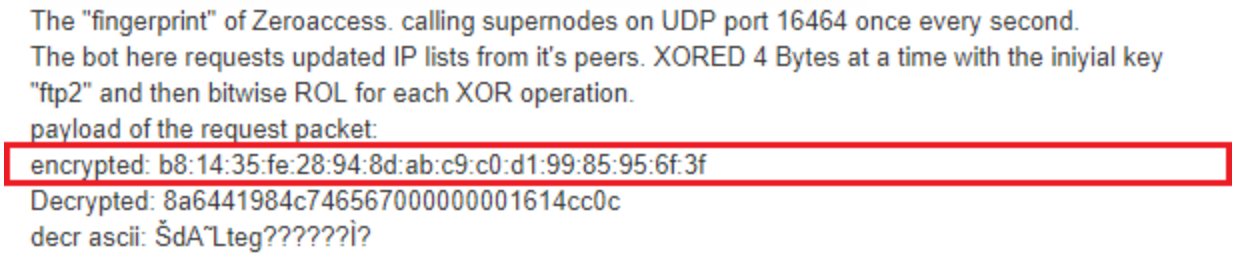
You should now be able to see a new log **conn_long.log** created in the current directory, with the below contents.



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.



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/utils/addrs
@load base/utils/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)]);
    }
    }
```

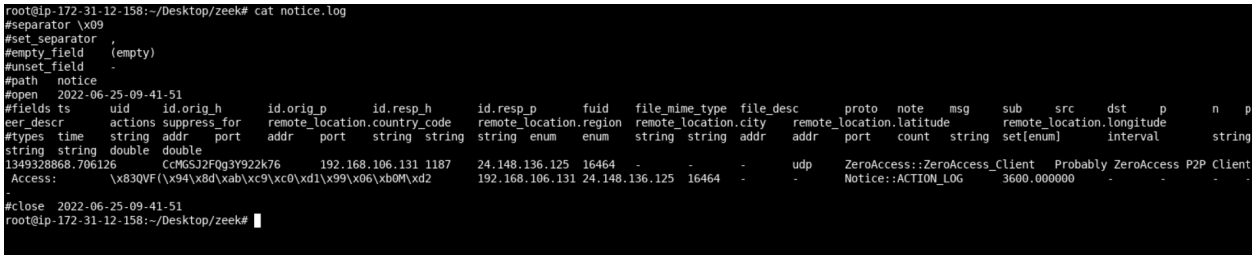
Let's now put the Zeek signature and the accompanying Zeek script to the test, as follows.

```
zeek -C -b -r ~/Desktop/PCAPs/ZeroAccess.pcap zeroaccess.zeek
```

You should now be able to find a **notice.log** file in the current directory, having the following content.

Run -

cat notice.log



Credits to Liam Randall (Critical Stack) for the Zeek script.

End of the Lab!

Effectively Using Snort

LAB 7

Scenario

The organization you work for is considering deploying Snort to enhance its traffic inspection capabilities. Your IT Security manager tasked you with thoroughly analyzing Snort's capabilities. He also provided you with PCAP files containing malicious traffic, so that you can experiment with writing Snort detection rules.

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.

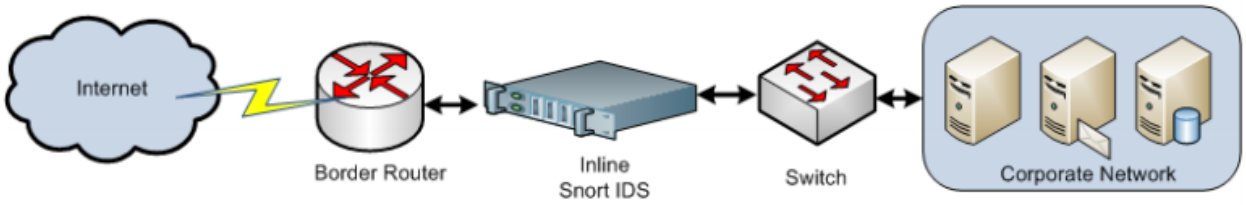
The most important Snort features and capabilities are:

- Deep packet inspection
- Packet capture logging
- Intrusion Detection

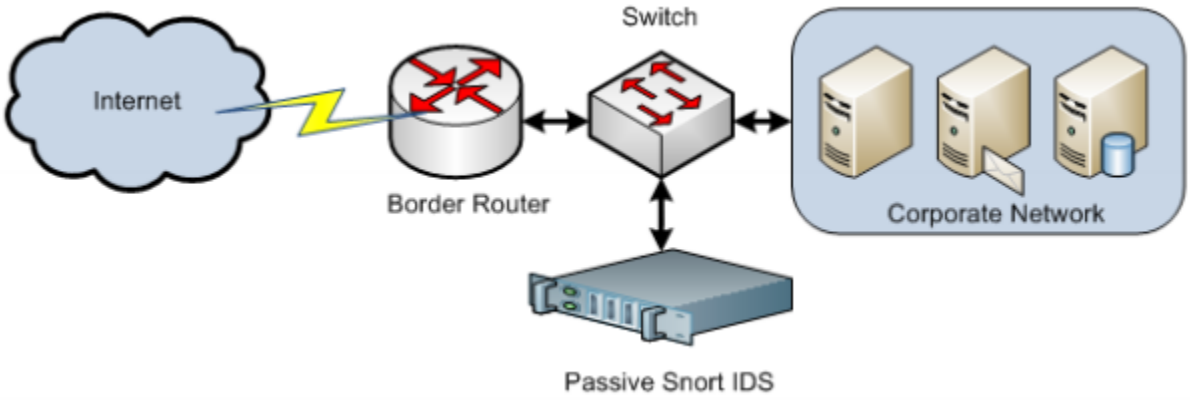
- Network Security Monitoring
- Anomaly Detection
- Multitenancy
- IPv6 and IPv4 support

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)



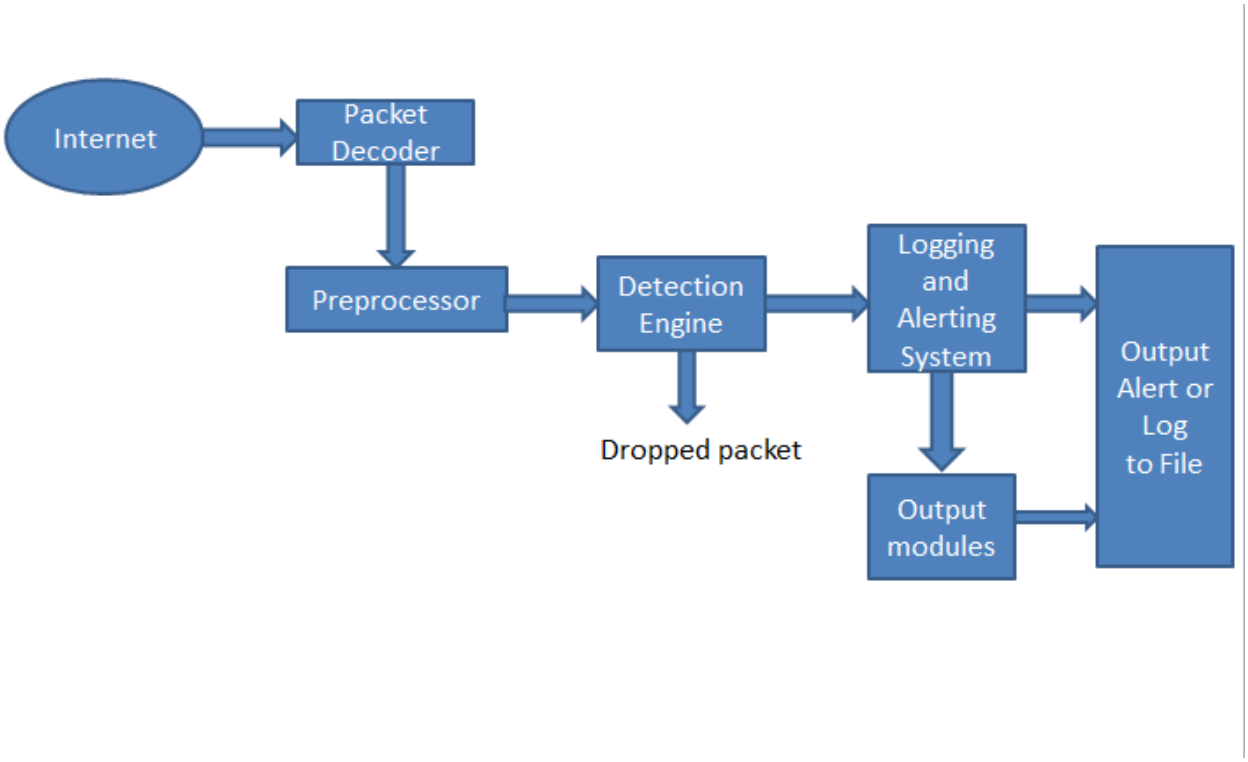
  - Passive IDS



  - 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/node27.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,
```

```
7777,7779,8000,8008,8014,8028,8080,8085,8088,8090,8118,8123,8
180,8181,8243,8280,8300,8800,8888,8899,9000,9060,9080,9090,90
91,9443,9999,11371,34443,34444,41080,50002,55555]
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 negation 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 theOpenSSL 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 -nnttttAr 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.

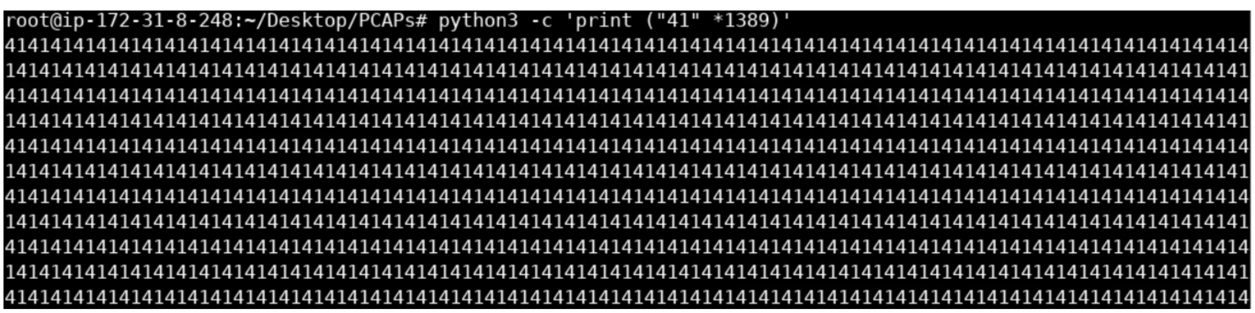
echo "copy-paste all the As here" | wc -m

You should see the below.

It is more efficient to specify the rule *content* in hex. Let's turn the above into hex as follows (A is 41 in hex).

python3 -c 'print ("41" * 1389)'

You should see the below.



Now, copy all the 41s.

Finally, create a Snort rule that will detect buffer overflow attempts (in this case over SMB), as follows.

vim /etc/snort/rules/local.rules

Inside vim enter the below.

alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (msg:"Buffer overflow activity over SMB"; content:"|paste all 41s here|"; sid:1000002; rev:1;)

You should see something similar inside the *local.rules* file.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (msg:"Buffer overflow activity over SMB";
content:"|-
4141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414
sid:1000002; 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

-k is used to disable Snort's entire checksum verification subsystem

You should see something similar to the below image.

```
02/22-22:13:34.487892  [**] [1:41978:5] OS-WINDOWS Microsoft Windows SMB remote code execution attempt [**] [
Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 192.168.1.6:55018 -> 192.168.1.4:
445
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
root@ip-172-31-8-248:~/Desktop/PCAPs#
```

So far, we wrote a rule that detects buffer overflow attempts on the wire (in this case over SMB).

Let's continue our analysis by executing tcpdump as follows, in order to search for a traffic portion that is related to the attacker obtaining Windows shell access.

cd PCAPs
tcpdump -nntttttAr eternalblue.pcap

Starting from the bottom up, you should see something similar to the below image.

```
2019-02-22 22:13:34.634511 IP 192.168.1.4.49165 > 192.168.1.6.4444: Flags [P.], seq 39:104, ack 1, win 256, l
ength 65
E..i..@...v&..........\..33..]2P....;..Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
```

The above is a unique enough traffic portion we can use to write a Snort rule that can detect Windows (7) shell access

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

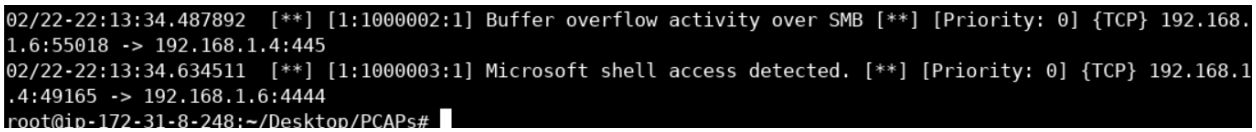
Inside vim enter the below.

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.

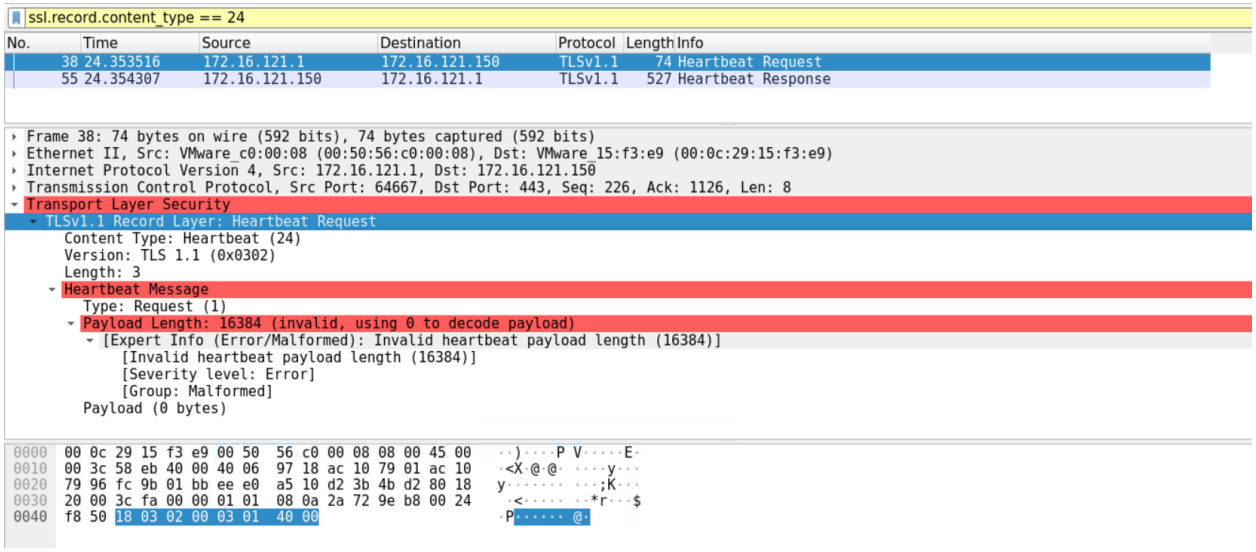


Task 3: Analyze the provided PCAP file and write a Snort rule to detect possible Heartbleed exploitation attempts

By reading the "*A technical view of theOpenSSL 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.

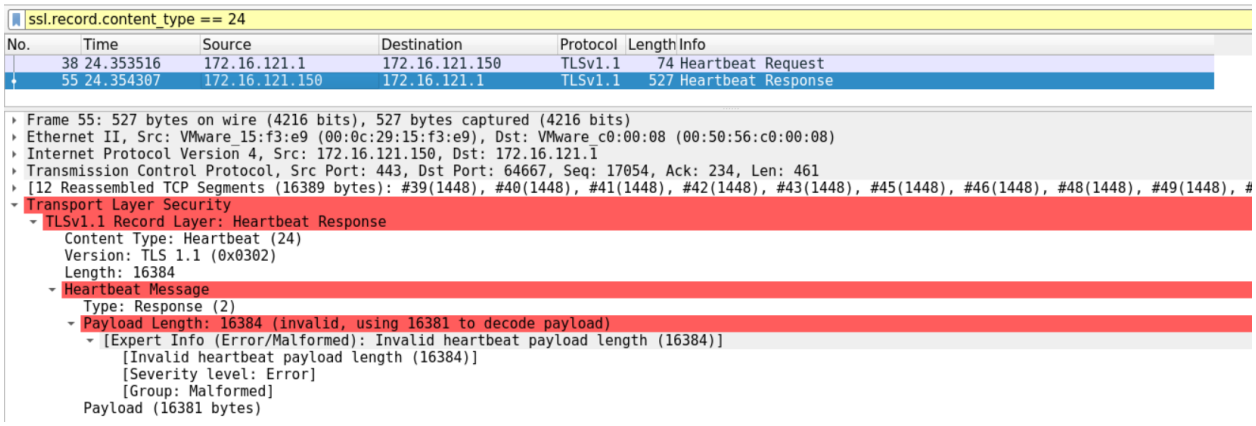
ssl.record.content_type == 24



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.



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 hear
t.pcap
04/16-12:00:41.512593  [**] [1:1000004:1] Potential Heartbleed attack [**] [Priority: 0] {TCP} 172.16.121.1:6
4667 -> 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
```

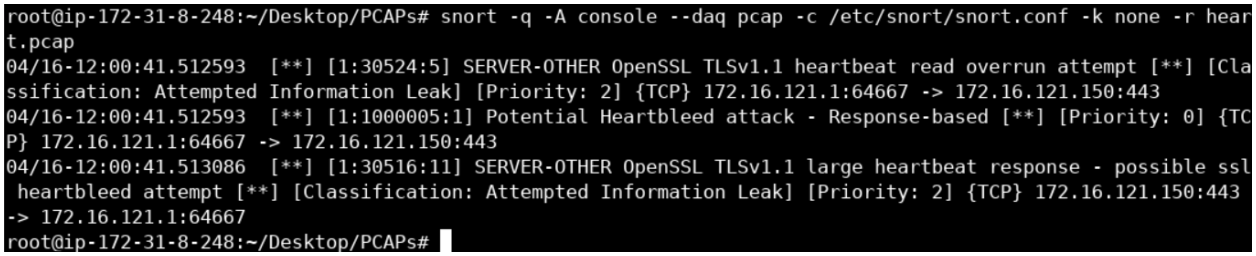
Also updating our rule with HOME_NET variable

```
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).



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**/**@nalyst**

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 many 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

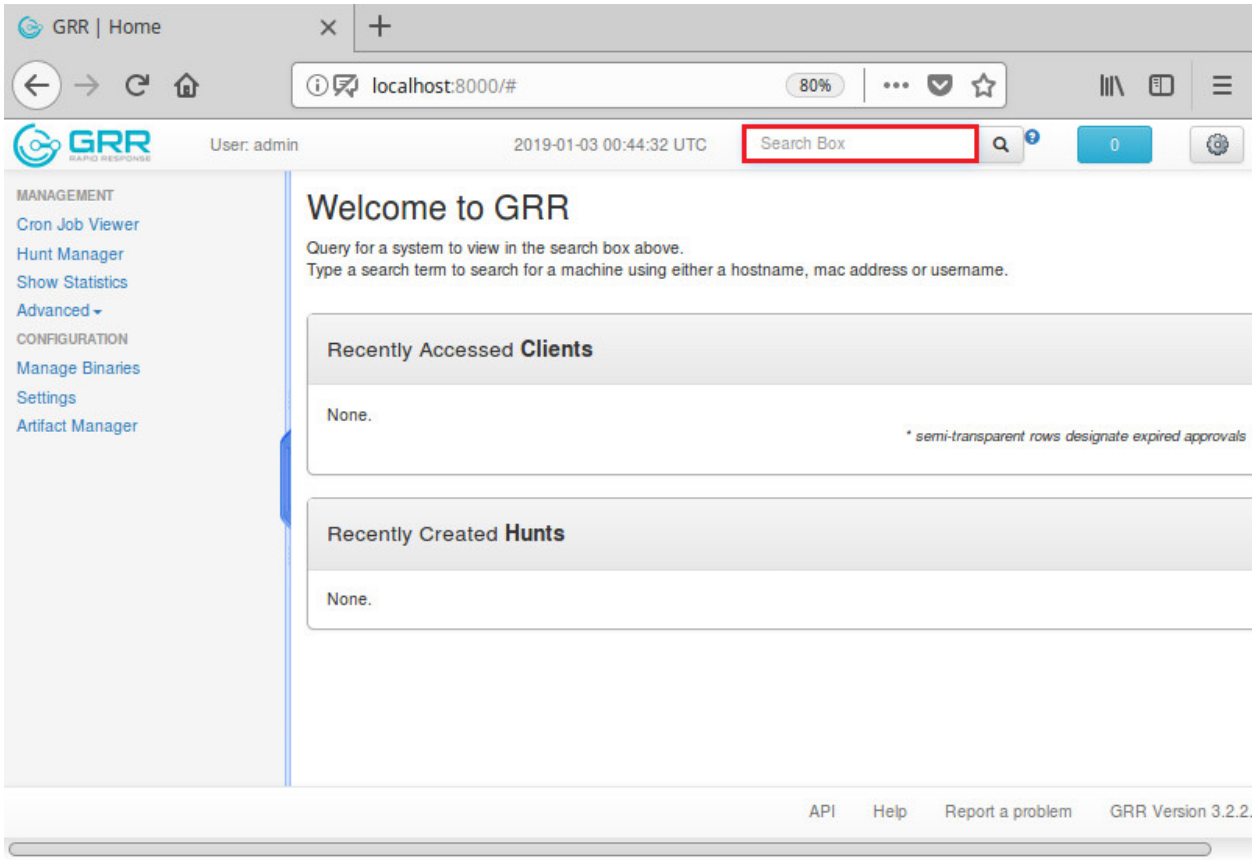
Below, you can find solutions for every task of this lab. Remember though, that you can follow your own strategy (which may be different from the one explained in the following lab).

Task 1: Identify any abnormalities on the win10-server.els-child.eLS.local endpoint, leveraging GRR

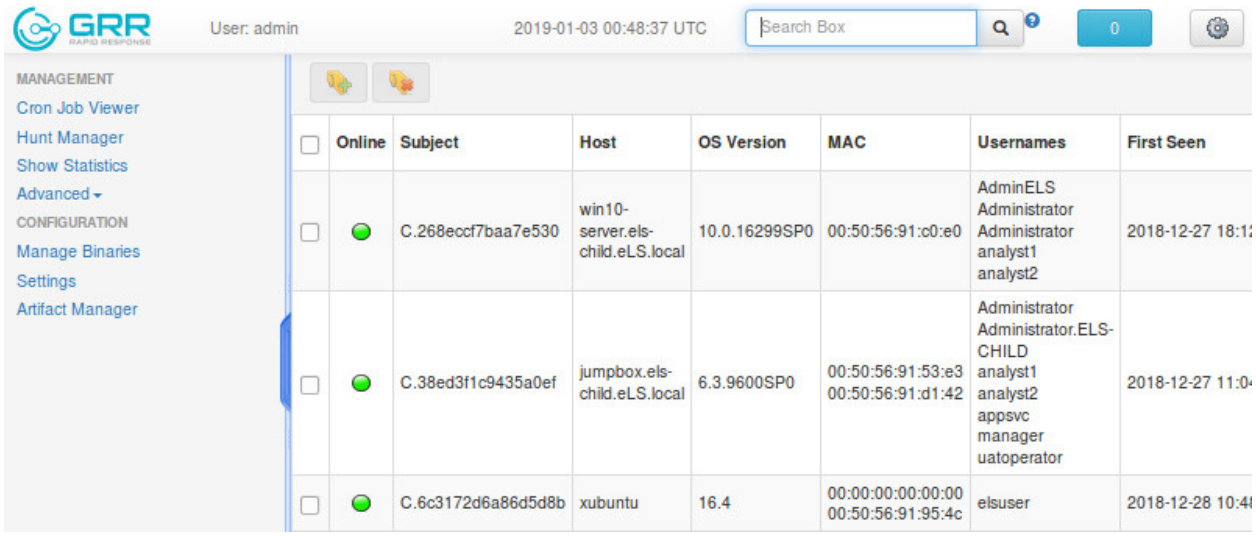
By the time a GRR client reports to a GRR server, the endpoint featuring the GRR client is being interrogated. "Interrogation" is a GRR process that effectively collects a treasure trove of endpoint information.

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.**

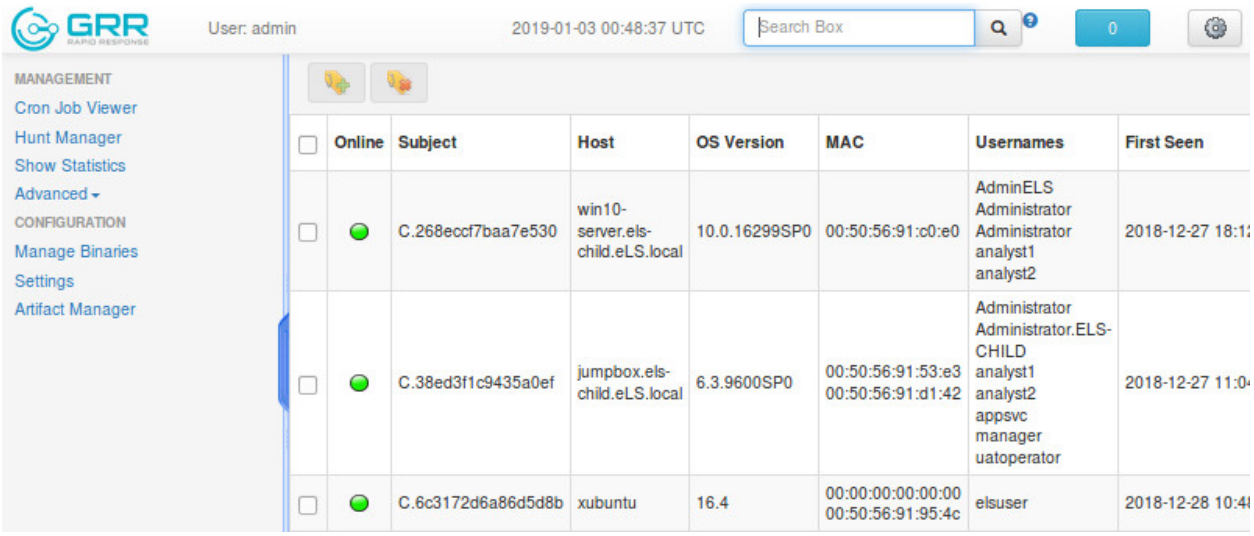


If you do so, you will be presented with the below:

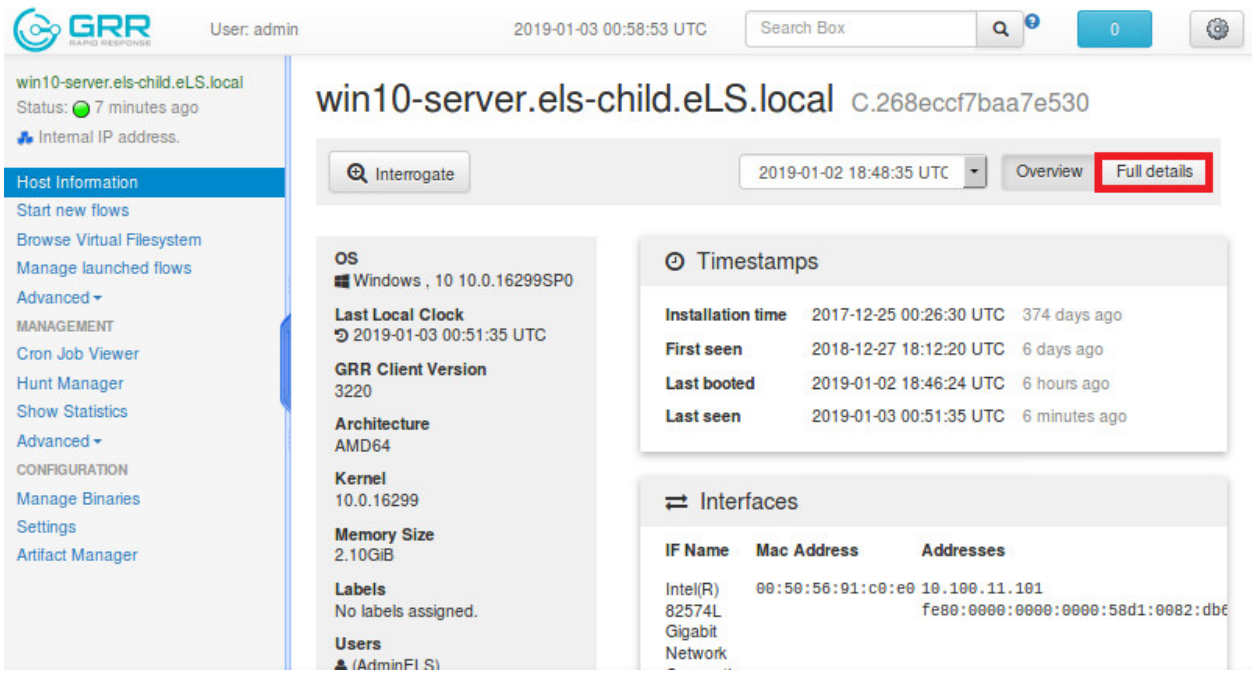


Important: Do not start any IR activities, until all bullets turn green! It may take some time until they do so... To refresh, click on the GRR logo and then, once again click on the **Search box** and press **Enter.**

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.

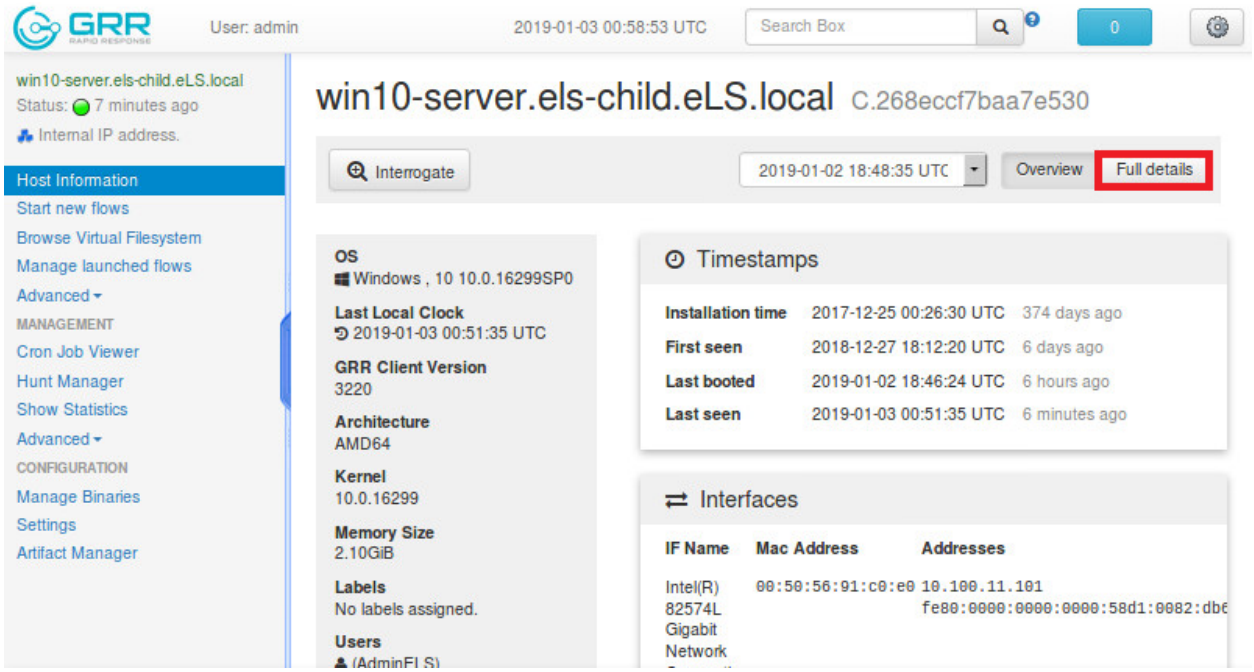


If you do so, you will be presented with the below.

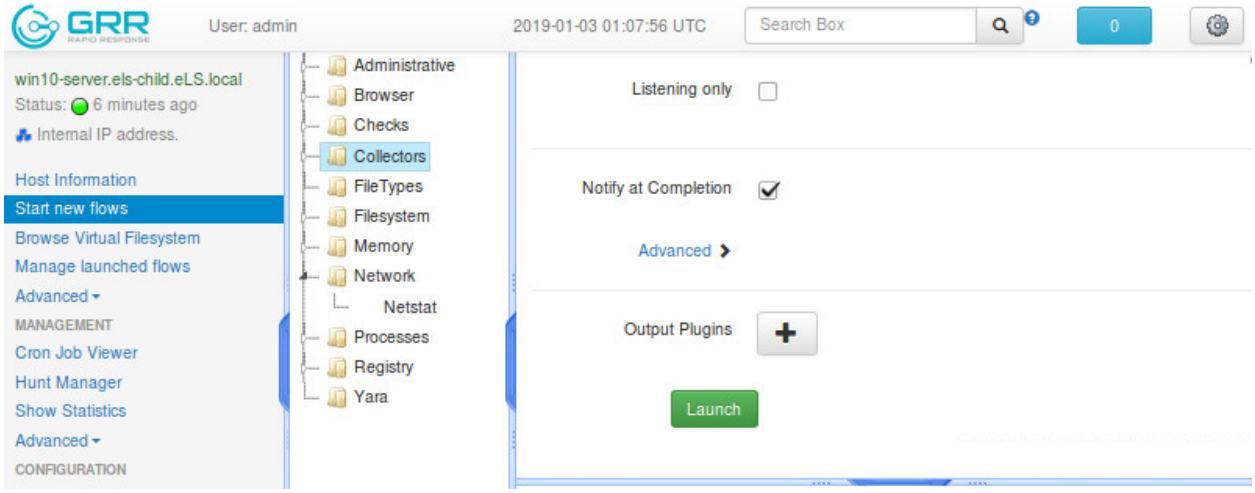


Click **Full Details**, for a more detailed representation of the acquired initial information. You can also browse through the same information collected on older dates and use this as a baseline.

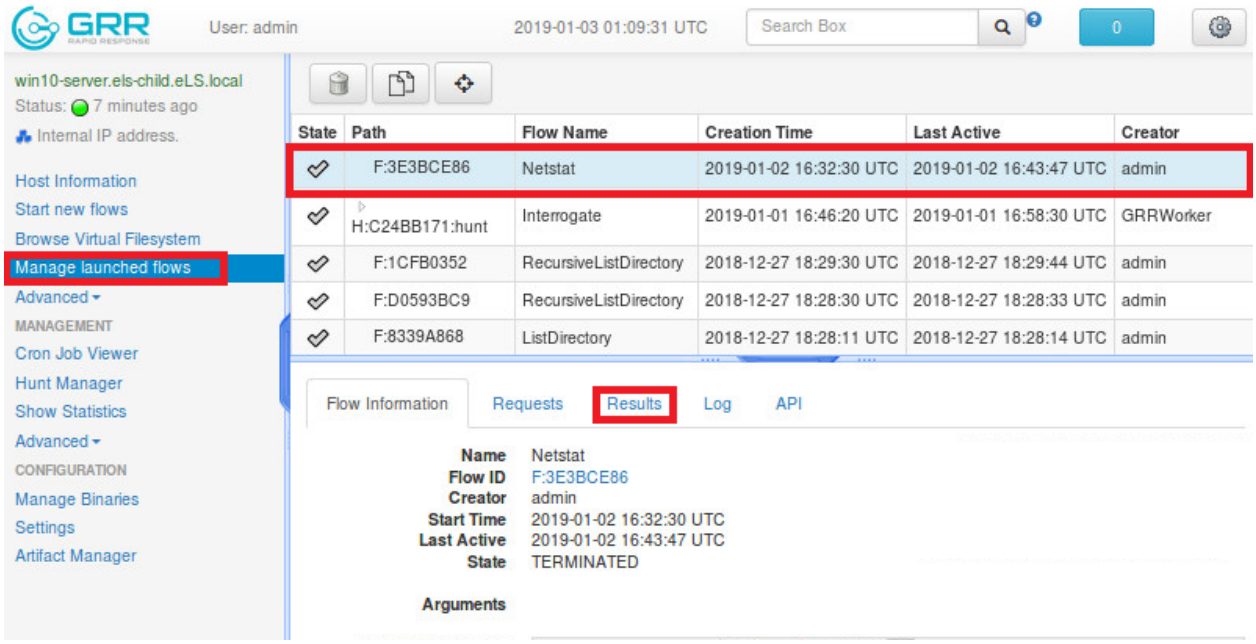
To start gathering important information about this endpoint, such as communications with other endpoints, listening ports etc., you can utilize GRR **flows**. To do so, click on **Start new flows**.



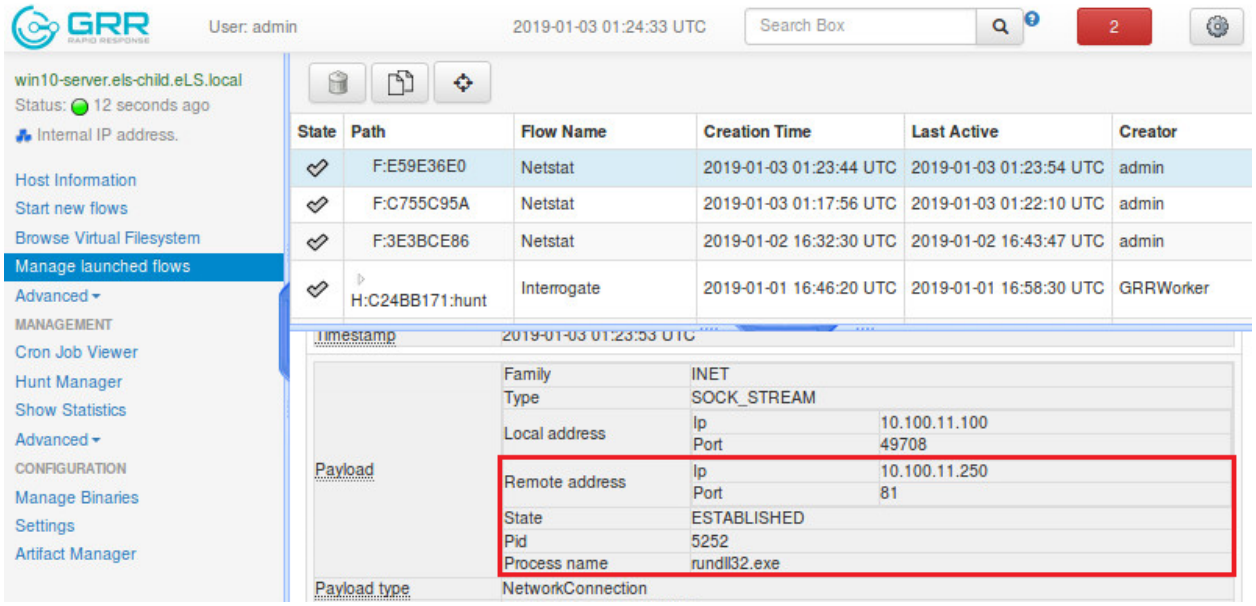
For example, to list all active network connections on this endpoint, you should go to **Network** -> **Netstat** and press **Launch**



To collect the results, you should click on **Manage launched flows** and then, click on the launched flow.



You will most probably be presented with two (2) pages of results, but the most curious looking result is the below.



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...

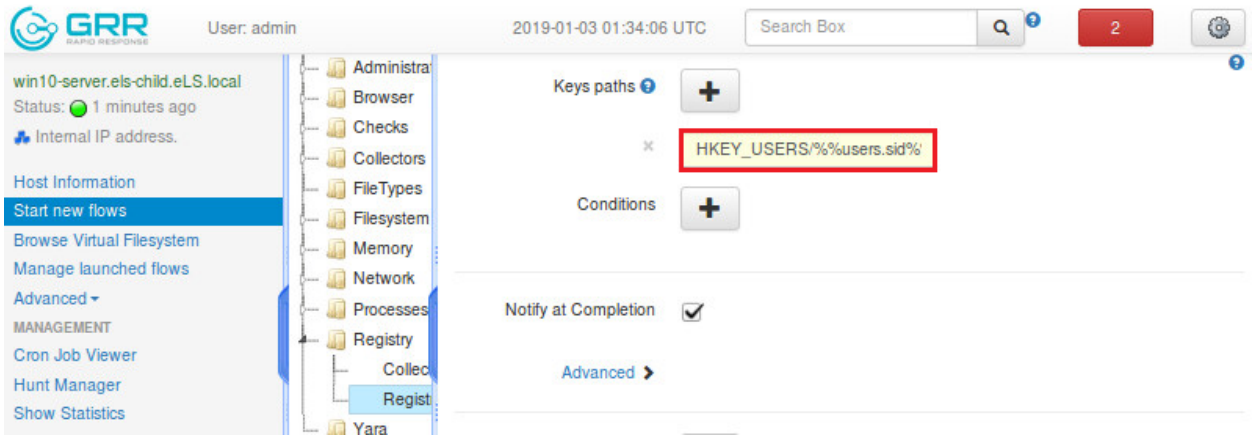
If you cannot find the result above, then, timing was bad (the connection went inactive). Don't worry, you can still identify that there is something wrong with rundll32, by clicking on **Start new flows**, navigating to **Processes** -> **ListProcesses** and finally clicking **Launch**.

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).

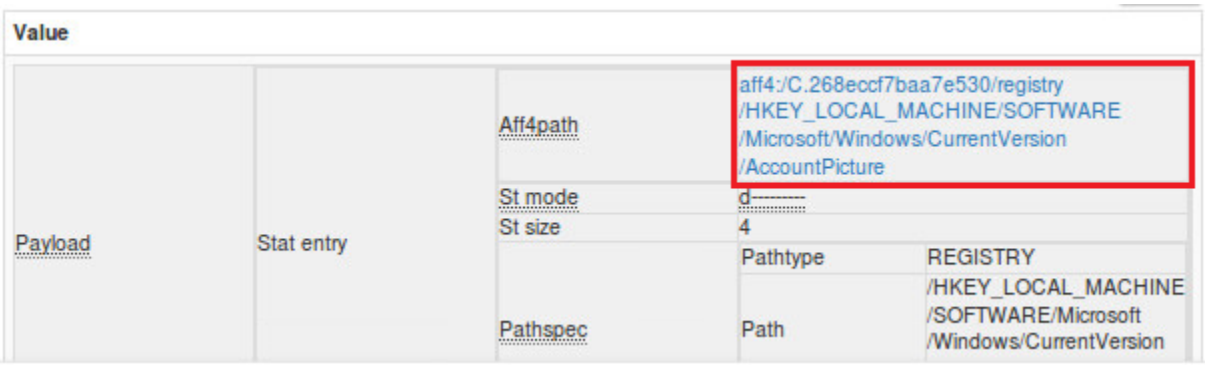
To do so, click on **Start new flows** and then navigate to **Registry** -> **RegistryFinder**.



Now, replace what is included in the red rectangle above with HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/ , *scroll down and press* *Launch**.

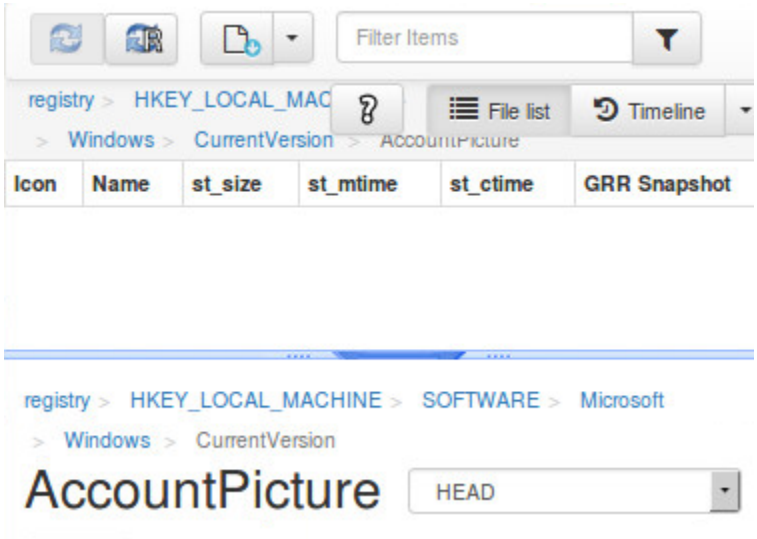
The results will appear in the **Manage launched flows** area, as we showed you previously. It will take a while until the results reach the GRR server...

You can inspect the results by clicking on them.

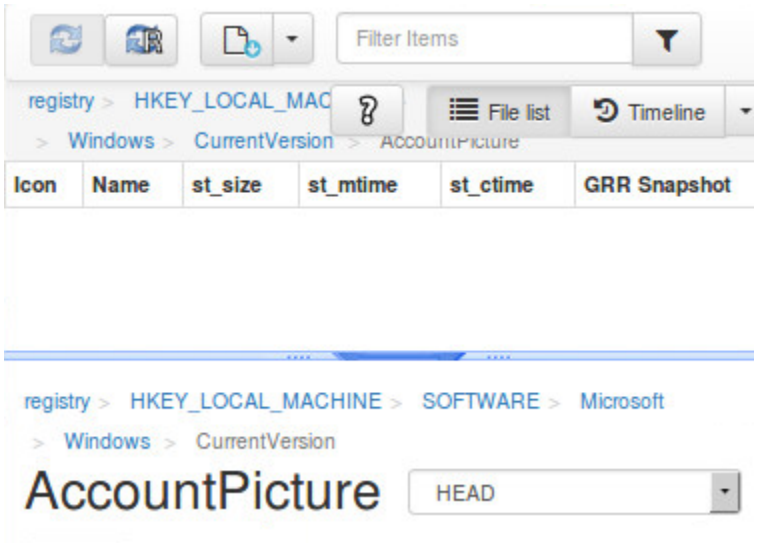


You will be taken to the **Browse Virtual Filesystem**, where you can investigate further.

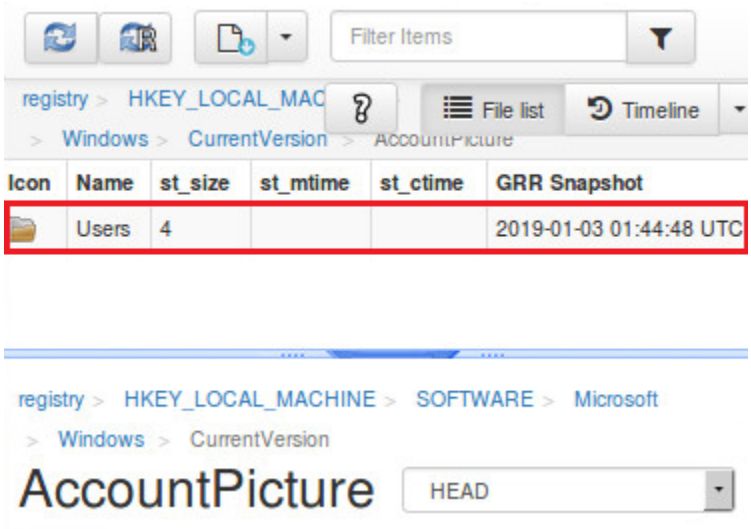
For example, if you wanted to investigate AccountPicture further, the first thing you would see, is that it appears to be empty.



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).



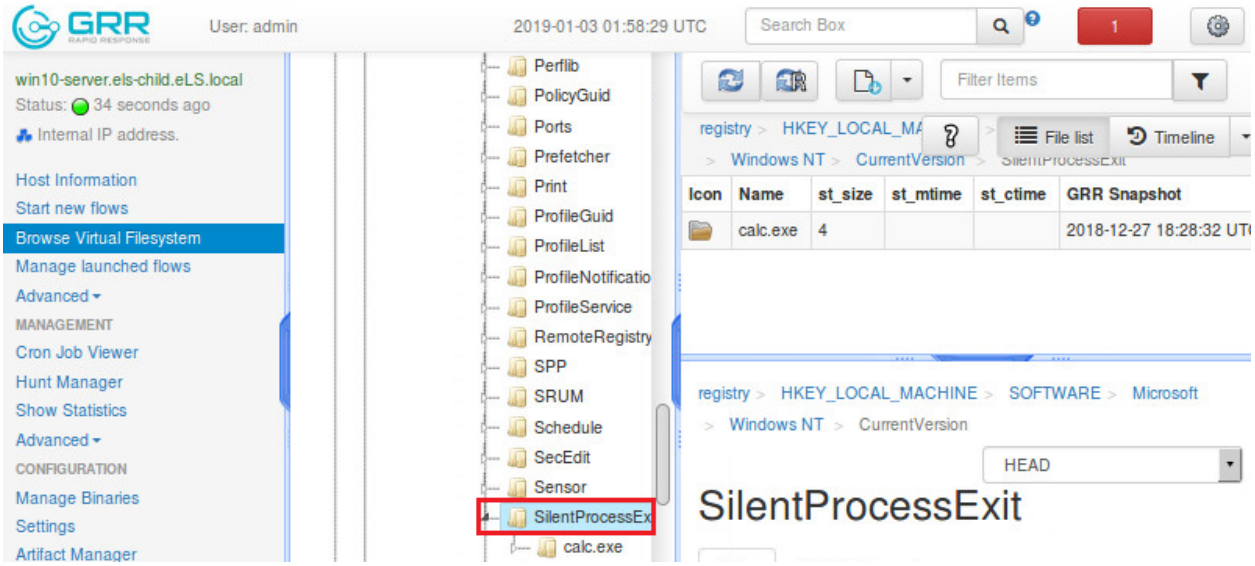
While investigating all results, you will not find anything suspicious.

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.

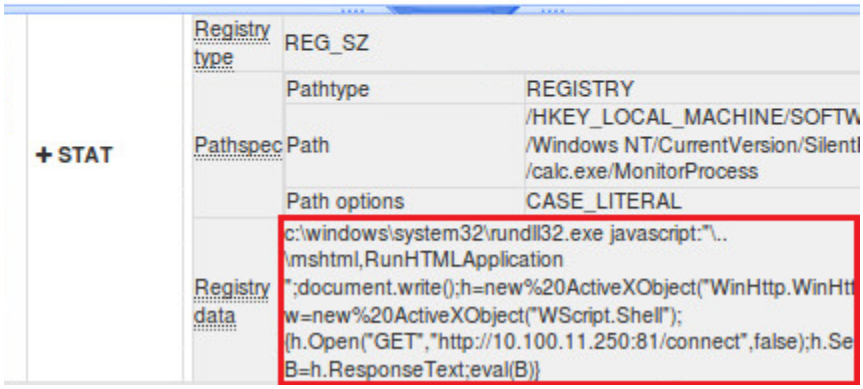


That *calc.exe* entry is certainly suspicious. Investigate it further, by first double-clicking on it and then, clicking the refresh button as you did previously. Once the latest results arrive, you will see the below.

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



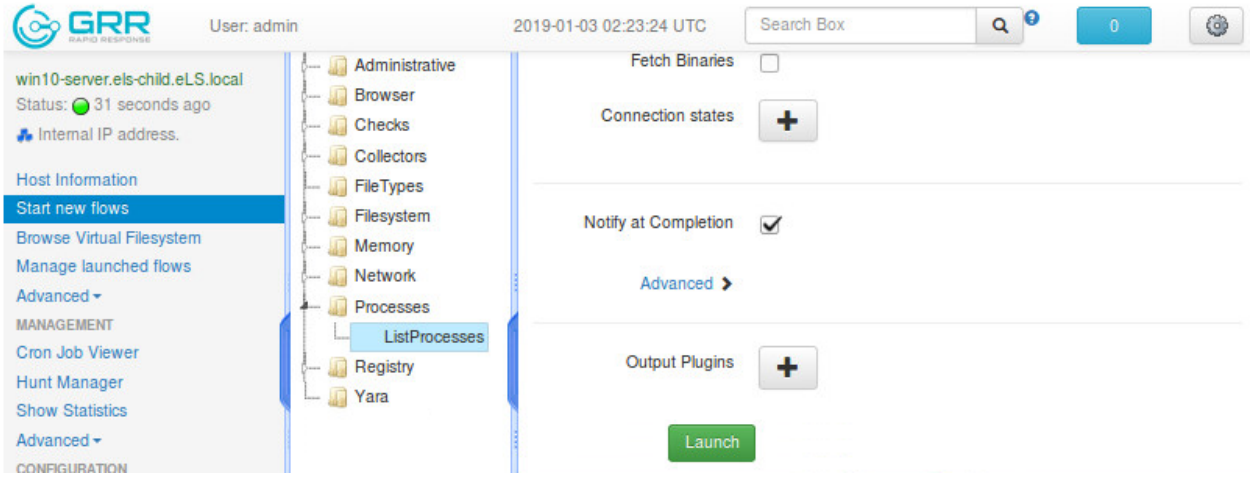
rundll32 executing JavaScript code is something 99% ill-intended (and this is why the endpoint was communicating with the 10.100.11.250 intranet machine on port 81). In addition, it looks like this is a persistence mechanism, taking into consideration that the Windows registry has been the go-to place 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**.



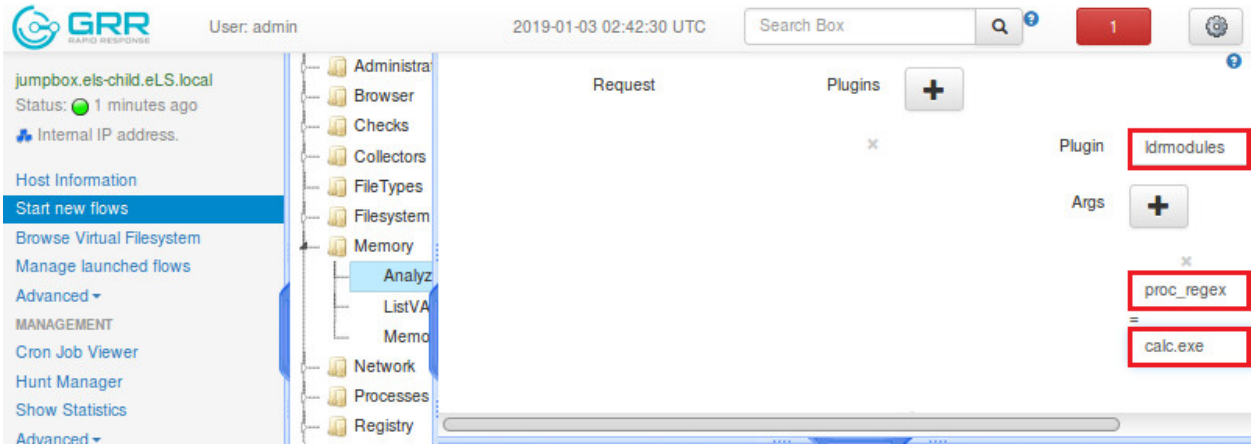
You will see results similar to the below in the **Manage launched flows** area (after ~10 minutes).

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

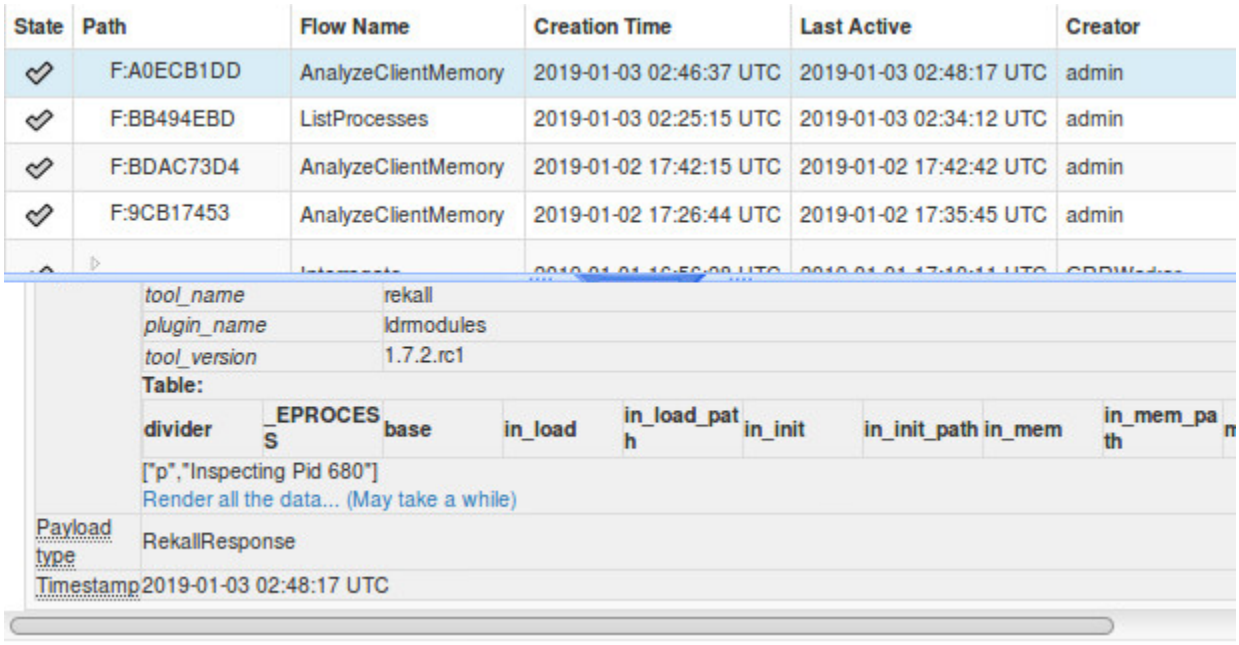
What looks interesting is that *calc.exe* is running. Of course, it could be something benign, but attackers commonly abuse *calc.exe* and *notepad.exe* processes to inject malicious code in them (and make it look like a legitimate Windows process). To inspect what has been loaded into/by *calc.exe* 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 *ldrmodules* 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

tool_name	rekall
plugin_name	ldrmodules
tool_version	1.7.2.rc1

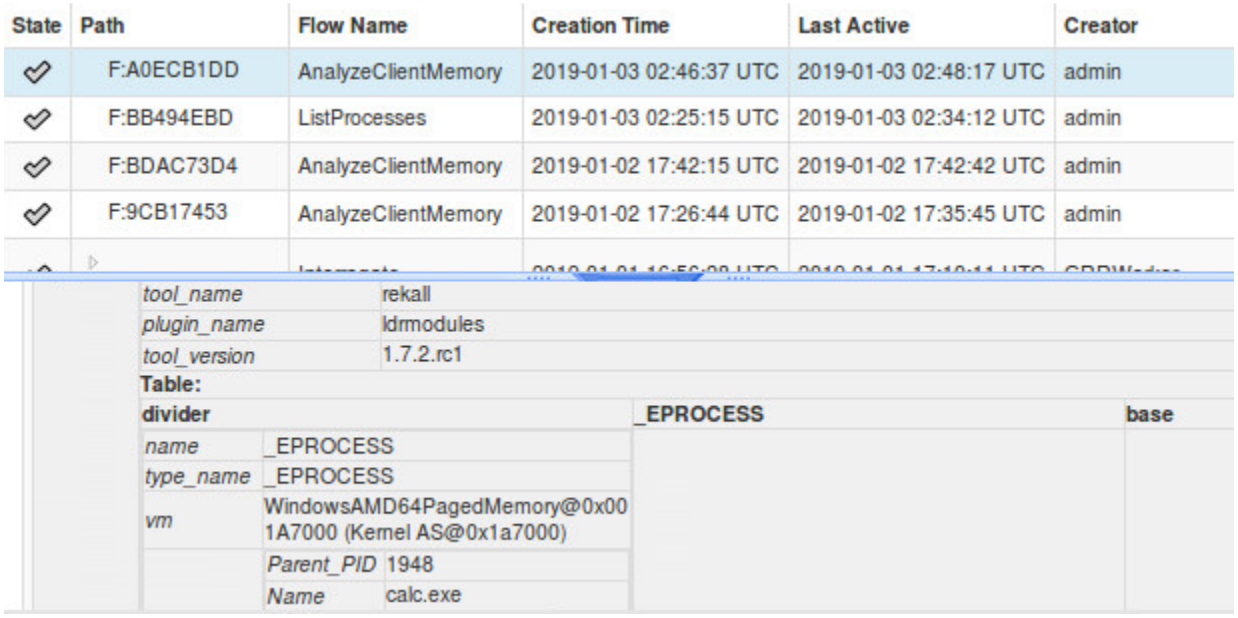
Table:

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)

Payload type	RekallResponse
Timestamp	2019-01-03 02:48:17 UTC

Now, click on **Render all the data... (May take a while)**. You will be presented with something similar to the below.



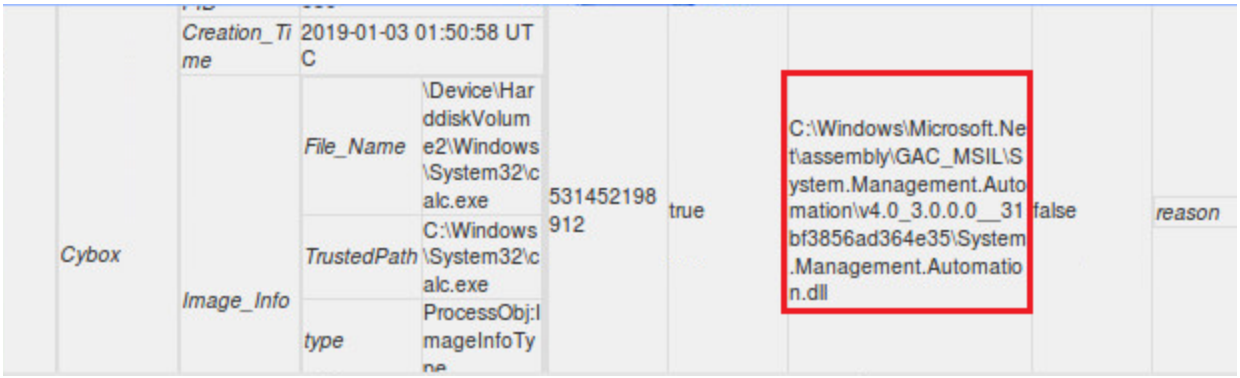
State	Path	Flow Name	Creation Time	Last Active	Creator
✓	F:A0ECB1DD	AnalyzeClientMemory	2019-01-03 02:46:37 UTC	2019-01-03 02:48:17 UTC	admin
✓	F:BB494EBD	ListProcesses	2019-01-03 02:25:15 UTC	2019-01-03 02:34:12 UTC	admin
✓	F:BDAC73D4	AnalyzeClientMemory	2019-01-02 17:42:15 UTC	2019-01-02 17:42:42 UTC	admin
✓	F:9CB17453	AnalyzeClientMemory	2019-01-02 17:26:44 UTC	2019-01-02 17:35:45 UTC	admin

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.



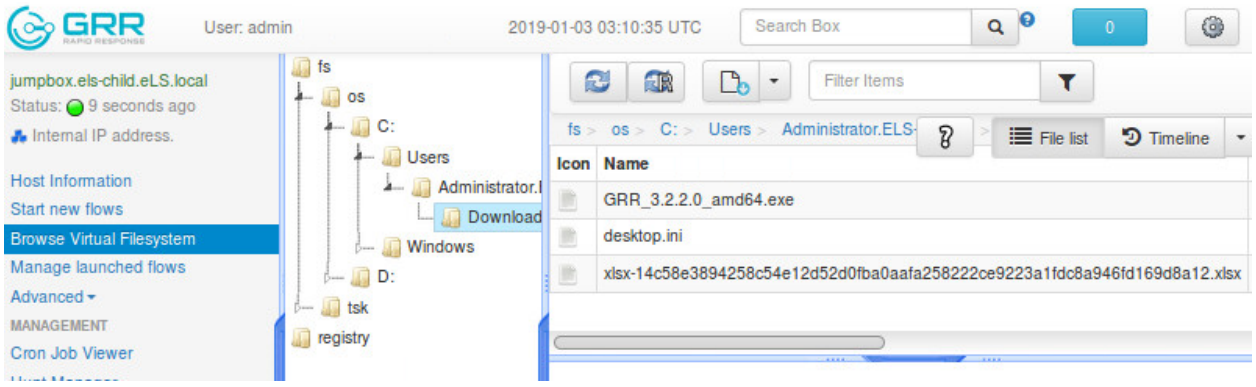
		Creation_Time	2019-01-03 01:50:58 UTC					
	Cybox	Image_Info	File_Name	\Device\HarddiskVolume2\Windows\System32\calc.exe	531452198912	true	C:\Windows\Microsoft.Net\assembly\GAC_MSIL\System.Management.Automation\v4.0_3.0.0.0__31bf3856ad364e35\System.Management.Automation.dll	false	reason
			TrustedPath	C:\Windows\System32\calc.exe					
			type	ProcessObj:ImageInfoType					

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.

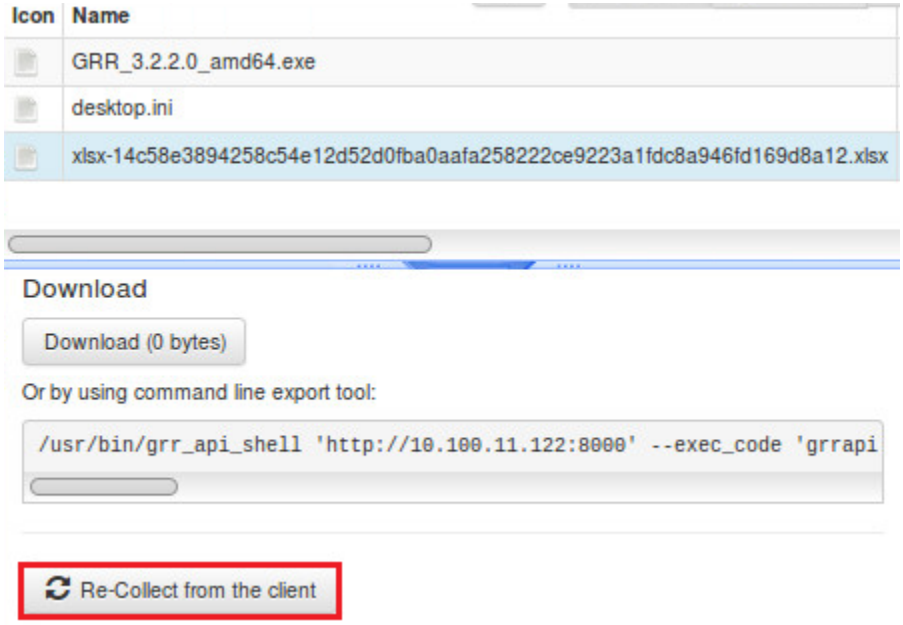
To do so, click on **Browse Virtual Filesystem**, then double-click on *fs -> os -> C: -> Users -> Administrator.ELS-CHILD -> Downloads* and finally click on the refresh button, as you did previously.

Once the latest directory contents are returned (after some minutes), you should see the below.

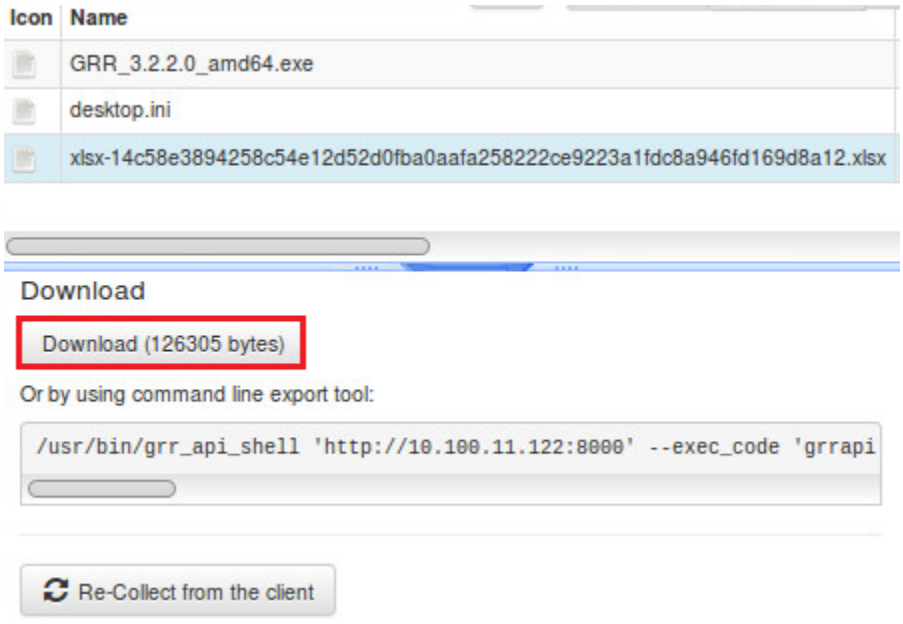


The GRR client installer and a .xlsx file are included. Let's give this .xlsx file a closer look.

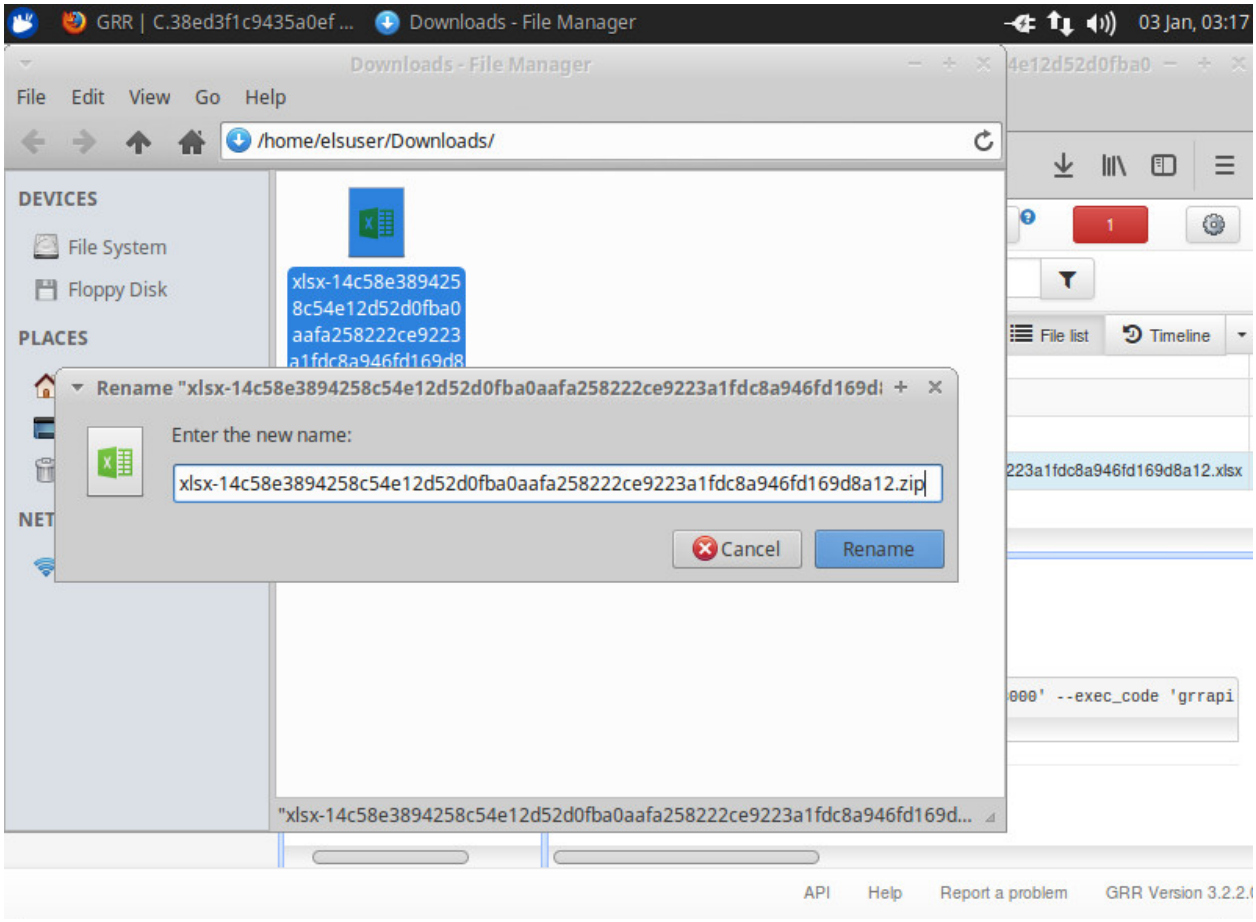
To do so, click on the .xslx file, click **Download**, scroll all the way down and finally click **Re-Collect from the client**.



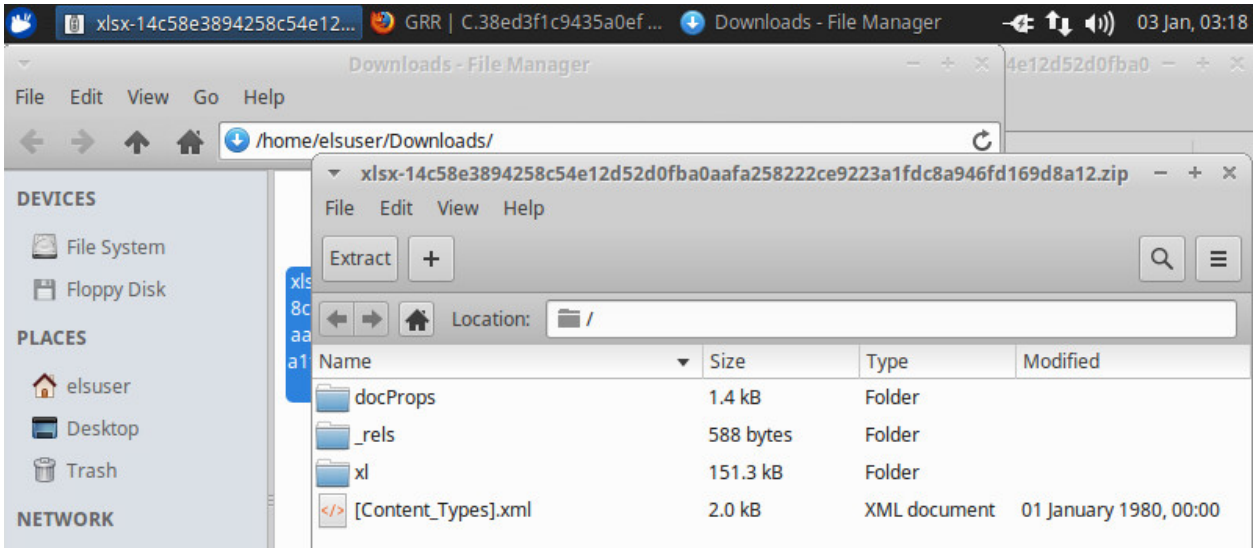
Once re-collecting is done, browse to the directory again and simply click on **Download (126305 bytes)**, as follows. It will save it into the GRR server's Download directory.



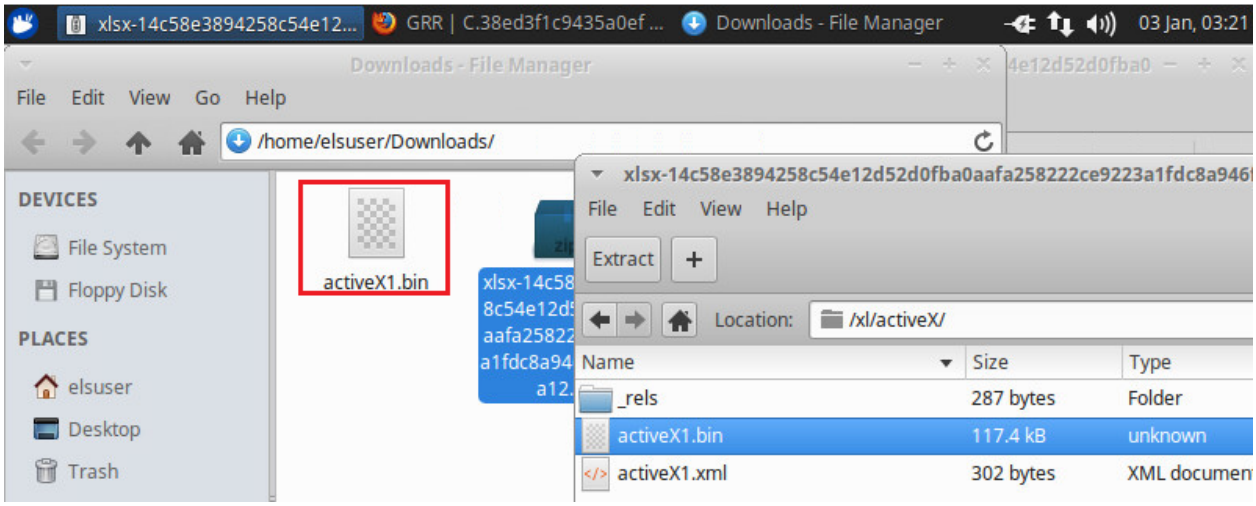
What you need to now is that .xlsx files are effectively .zip files. So, let's rename this file to .zip



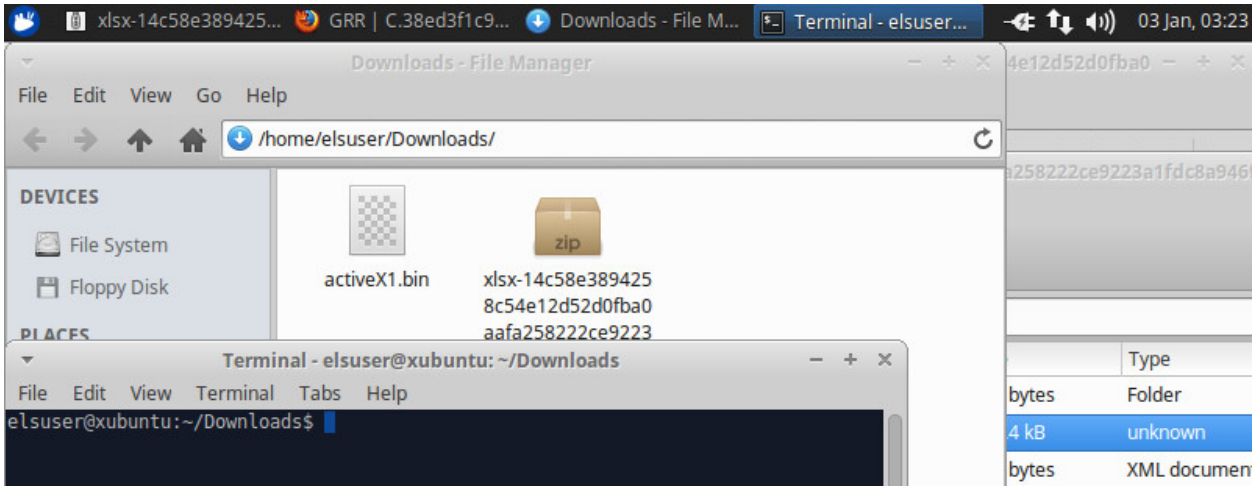
Once you rename it, double-click on it. You will be presented with the below.



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.**

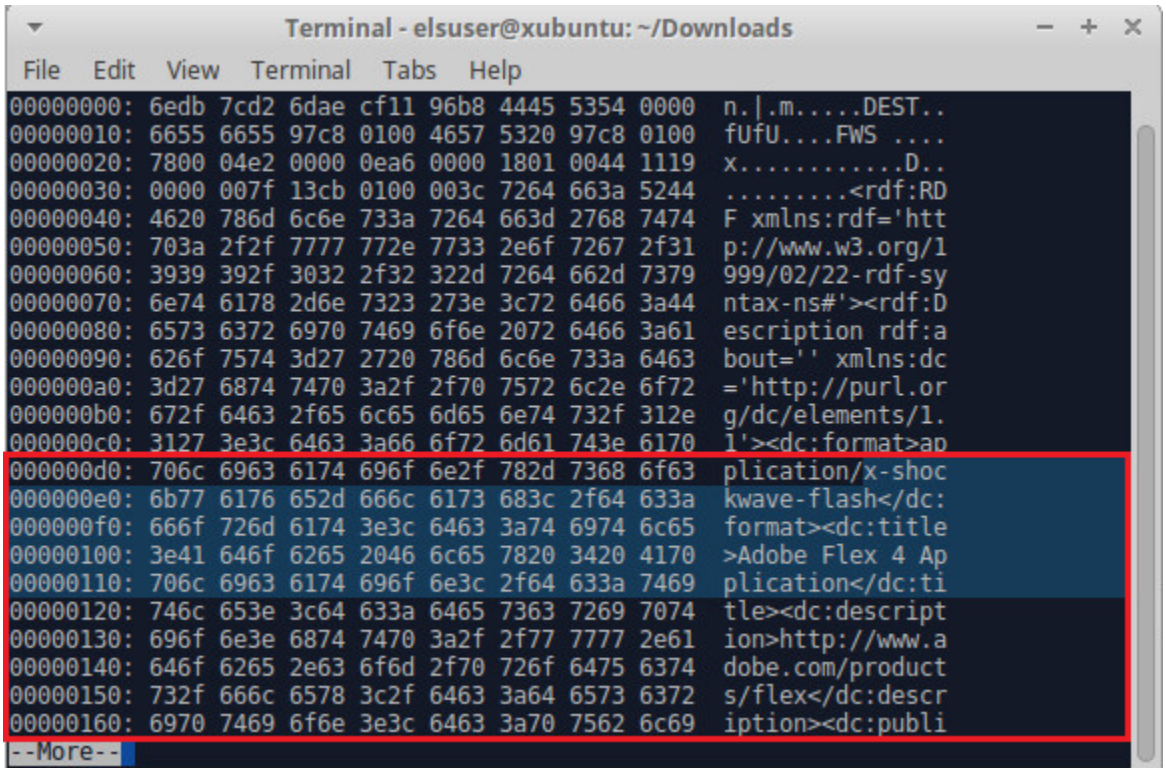


You did that, so that you can inspect the activeX1.bin file with the xxd tool.

Inside the terminal, execute:

```
xxd activeX1.bin | more
```

You will be presented with the below.



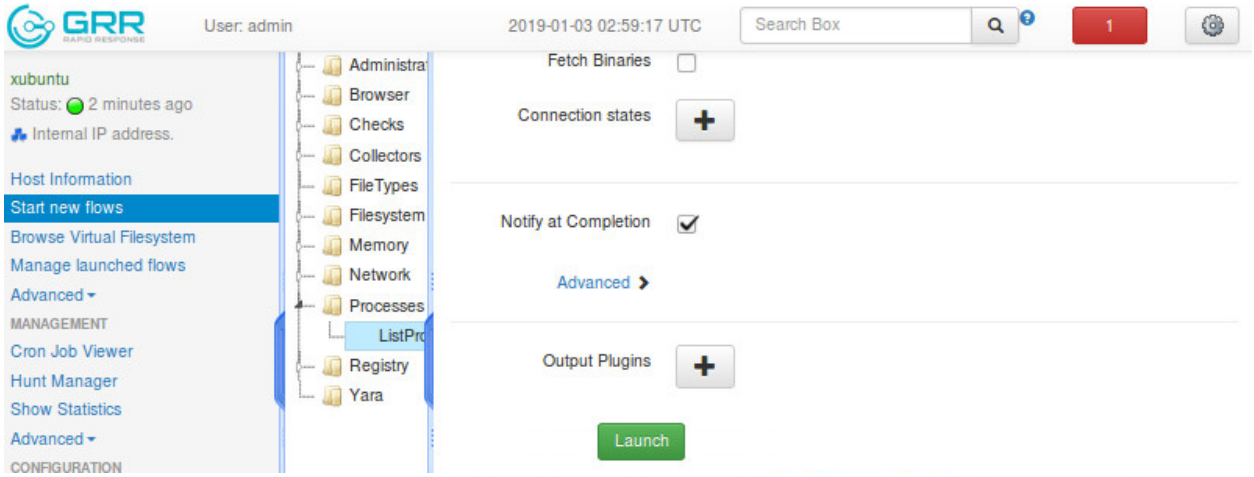
The above means, that the .xlsx file contained a Flash Application, something common only on malicious Office documents that leverage Flash vulnerabilities for malicious code execution purposes.

You most probably identified how this endpoint got infected in the first place!

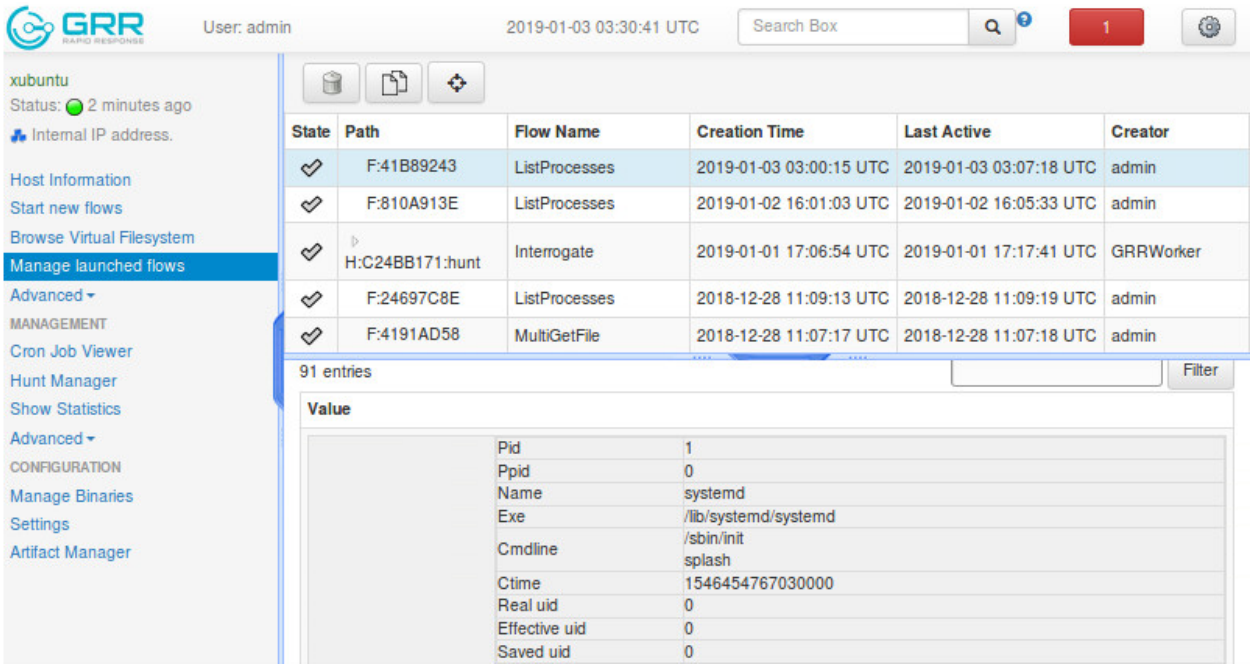
Task 3: Identify any abnormalities on the Xubuntu endpoint, leveraging GRR

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.

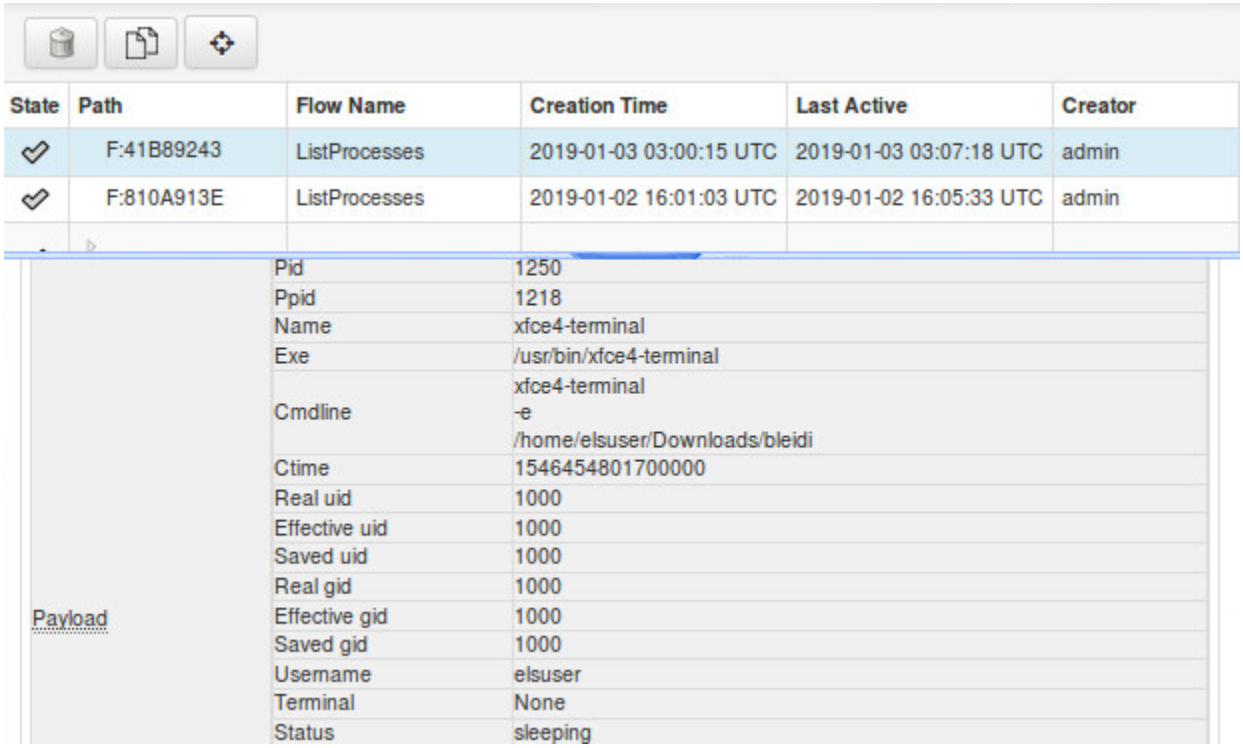
To list this endpoint's running processes, click on **Start new flows**, then navigate to **Processes** -> **ListProcesses** and finally click **Launch**.

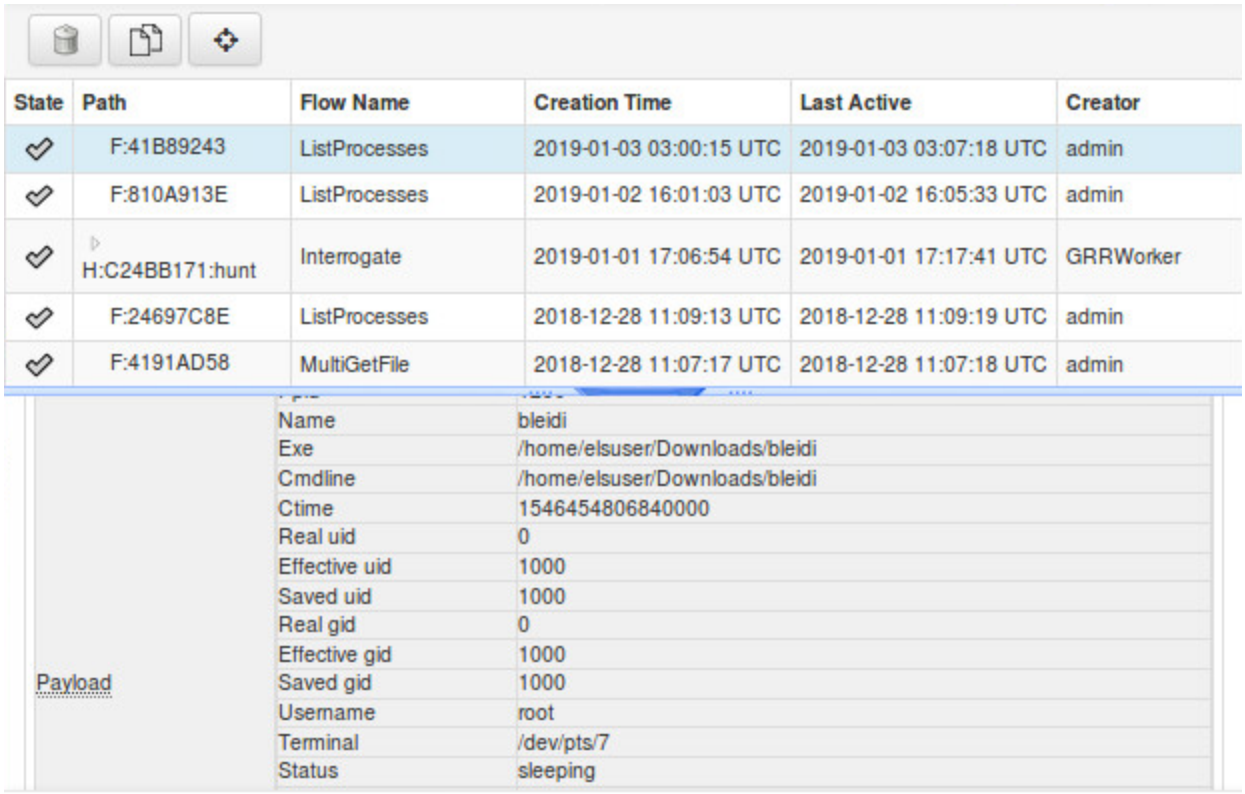


You will see results similar to the below in the **Manage launched flows** area (after ~10 minutes).



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).





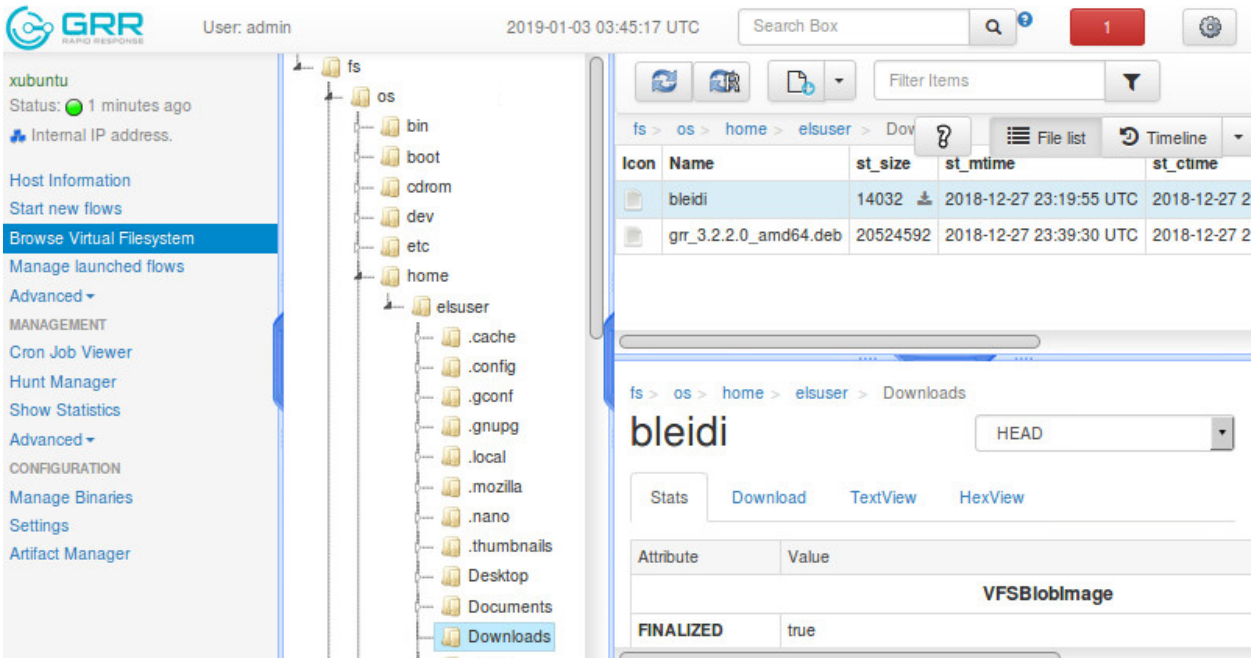
State	Path	Flow Name	Creation Time	Last Active	Creator
✓	F:41B89243	ListProcesses	2019-01-03 03:00:15 UTC	2019-01-03 03:07:18 UTC	admin
✓	F:810A913E	ListProcesses	2019-01-02 16:01:03 UTC	2019-01-02 16:05:33 UTC	admin
✓	H:C24BB171:hunt	Interrogate	2019-01-01 17:06:54 UTC	2019-01-01 17:17:41 UTC	GRRWorker
✓	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

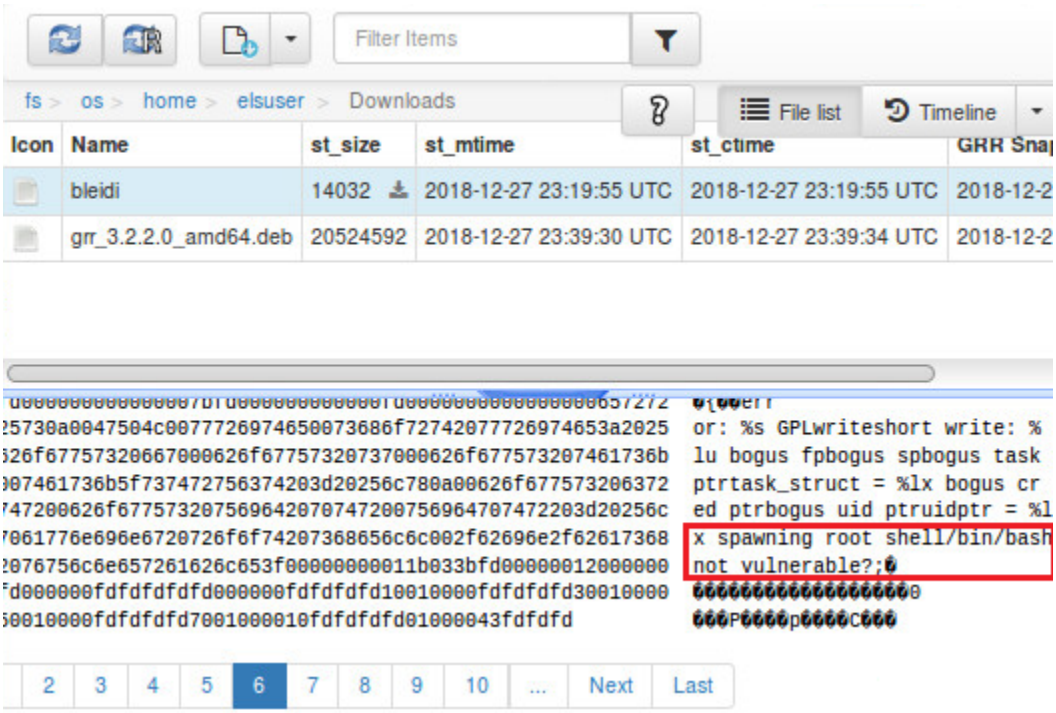
Name	bleidi
Exe	/home/elsuser/Downloads/bleidi
Cmdline	/home/elsuser/Downloads/bleidi
Ctime	1546454806840000
Real uid	0
Effective uid	1000
Saved uid	1000
Real gid	0
Effective gid	1000
Saved gid	1000
Username	root
Terminal	/dev/pts/7
Status	sleeping

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=ren6OSvwFvg
4. https://chip-dfir.techanarchy.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

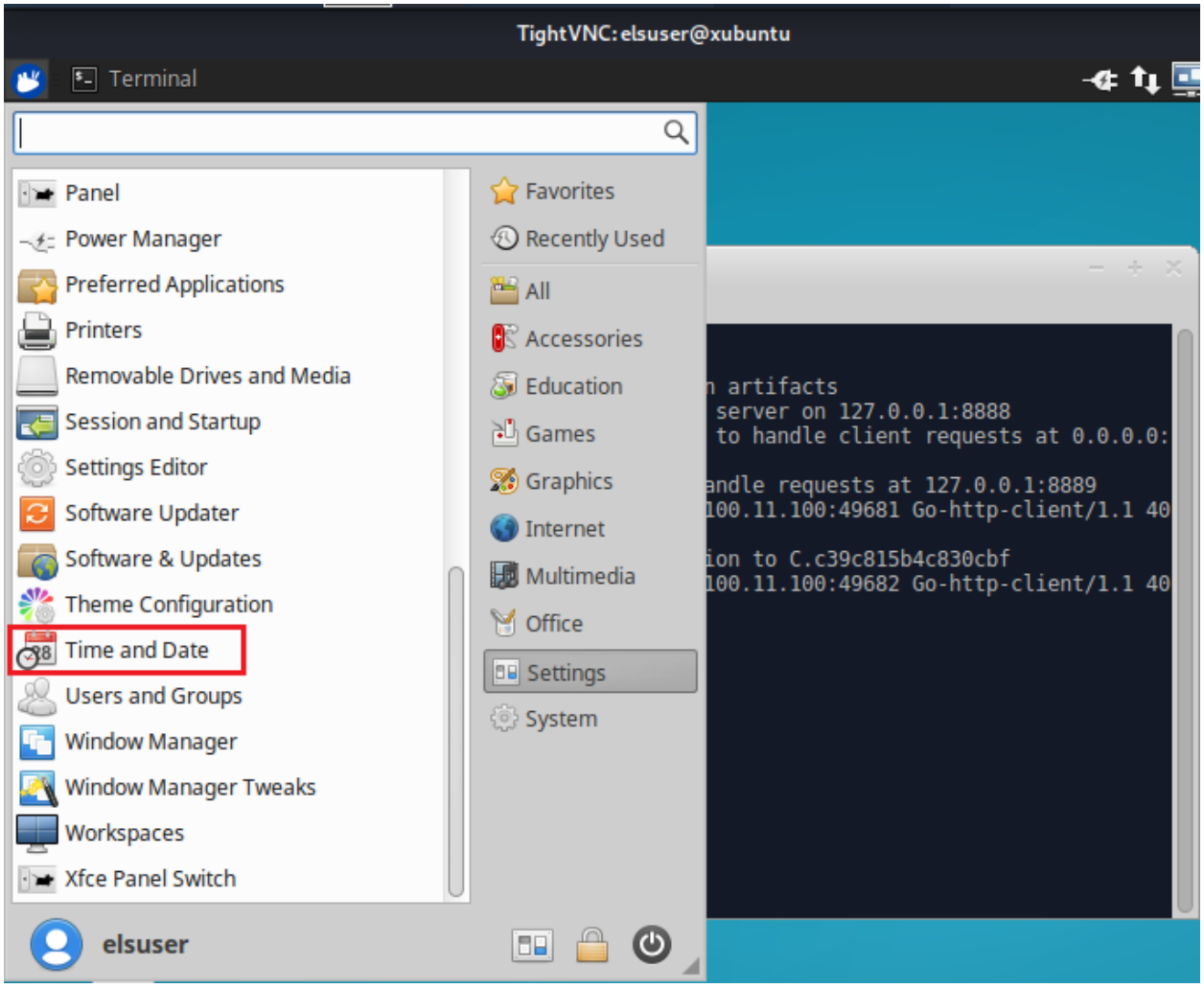
- Incident Responder's Subnet: **172.16.67.0/24**
- Under-investigation endpoints' subnet: **10.100.11.0/24**
- Velociraptor server
    - **IP:** 10.100.11.121
- Connection Type: **VNC**
    - Use a Linux or Windows VNC client (https://tightvnc.com/download.html) to connect to Velociraptor server (10.100.11.121)

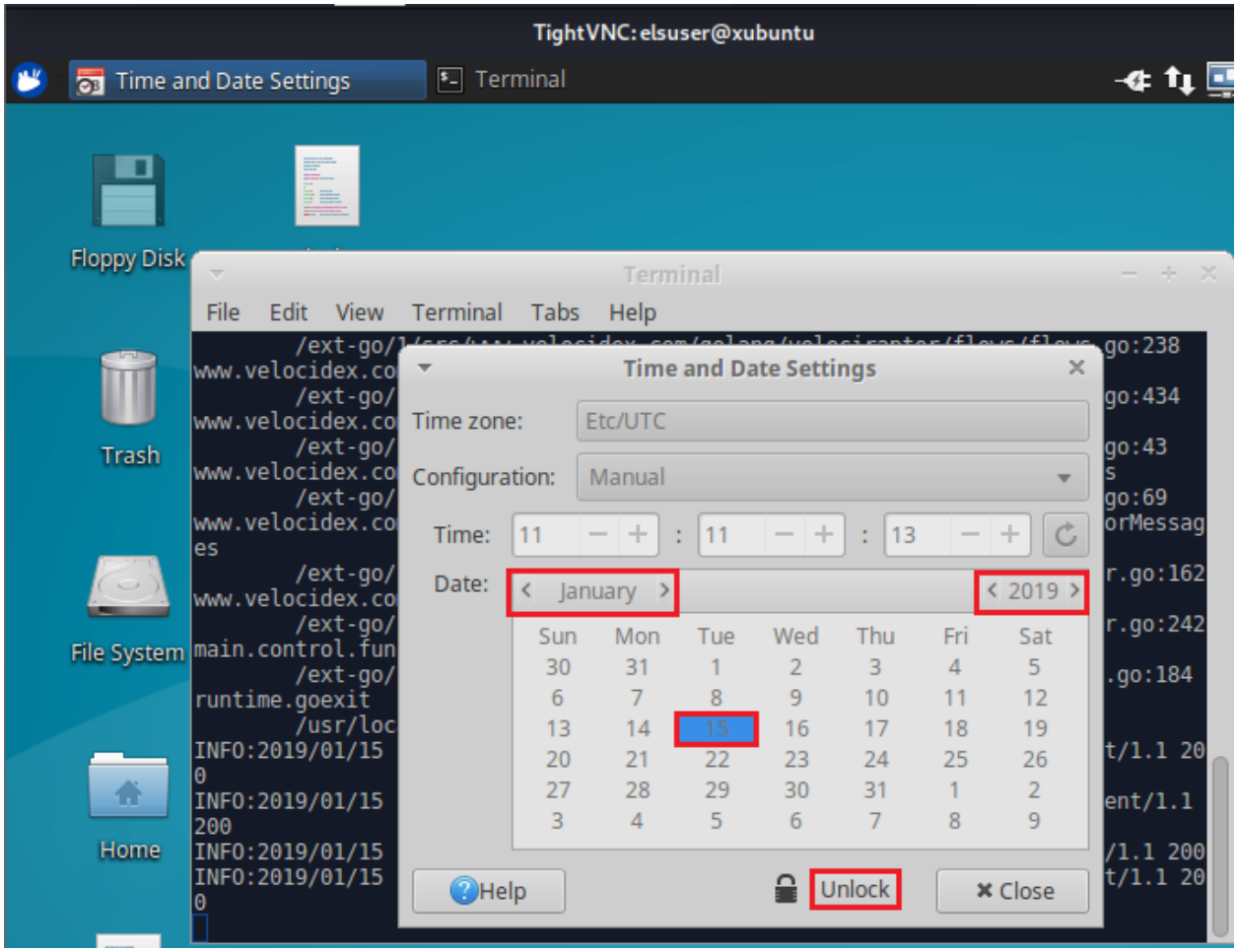
vncviewer 10.100.11.121 <- For Linux-based machines
tvnviewer.exe, Remote Host:10.100.11.121 <- For Windows-based machines

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 **7hwAJc3l** as the password to unlock and then lock the calendar that will appear)


TightVNC: elsuser@xubuntu
Terminal
Panel
Power Manager
Preferred Applications
Printers
Removable Drives and Media
Session and Startup
Settings Editor
Software Updater
Software & Updates
Theme Configuration
Time and Date
Users and Groups
Window Manager
Window Manager Tweaks
Workspaces
Xfce Panel Switch
Favorites
Recently Used
All
Accessories
Education
Games
Graphics
Internet
Multimedia
Office
Settings
System
elsuser
n artifacts
server on 127.0.0.1:8888
to handle client requests at 0.0.0.0:
andle requests at 127.0.0.1:8889
100.11.100:49681 Go-http-client/1.1 40
ion to C.c39c815b4c830cbf
100.11.100:49682 Go-http-client/1.1 40



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

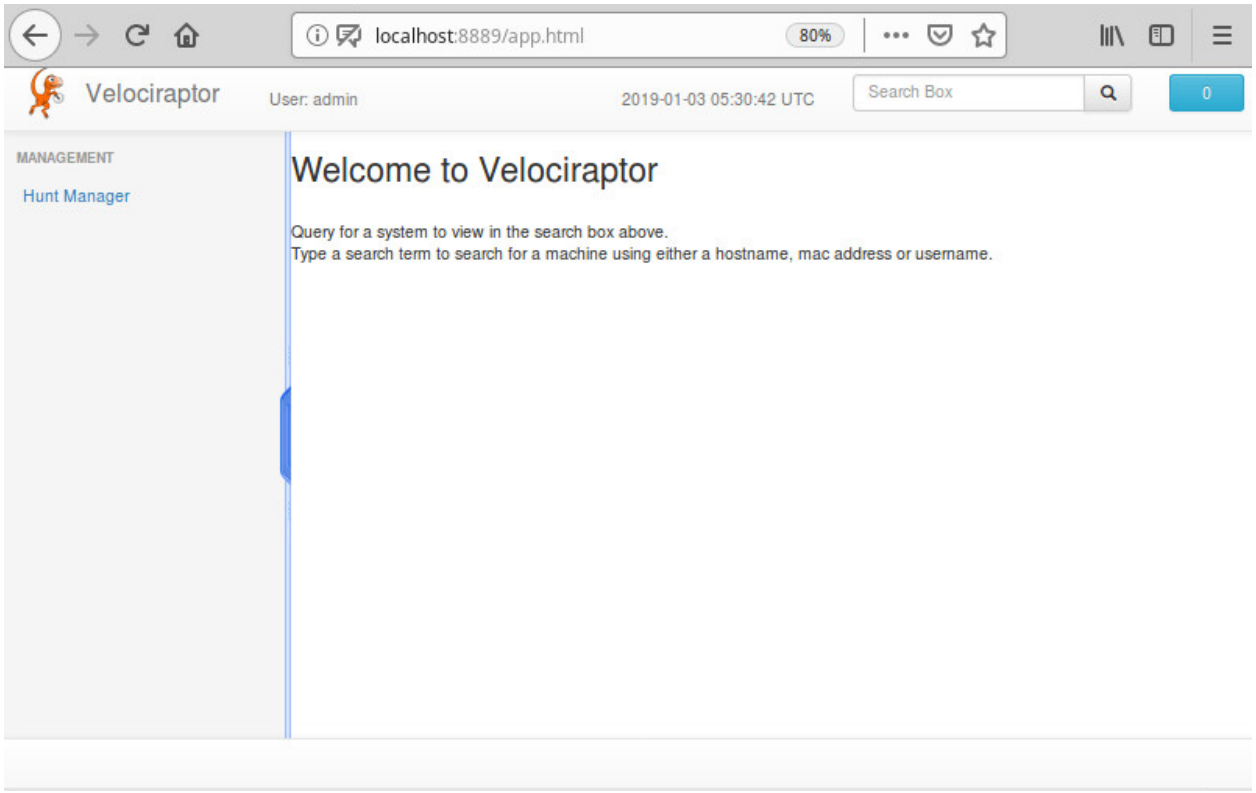
SOLUTIONS

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).

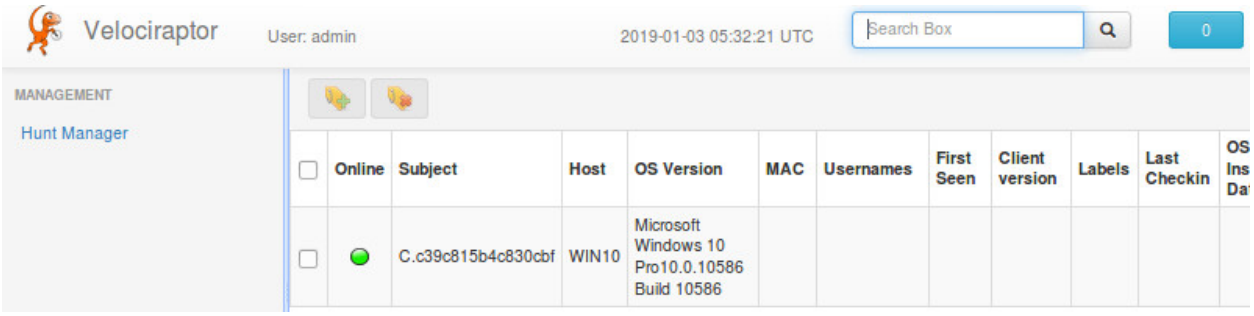
Task 1: Identify any abnormalities on the win10.els-child.eLS.local endpoint, leveraging Velociraptor

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**.



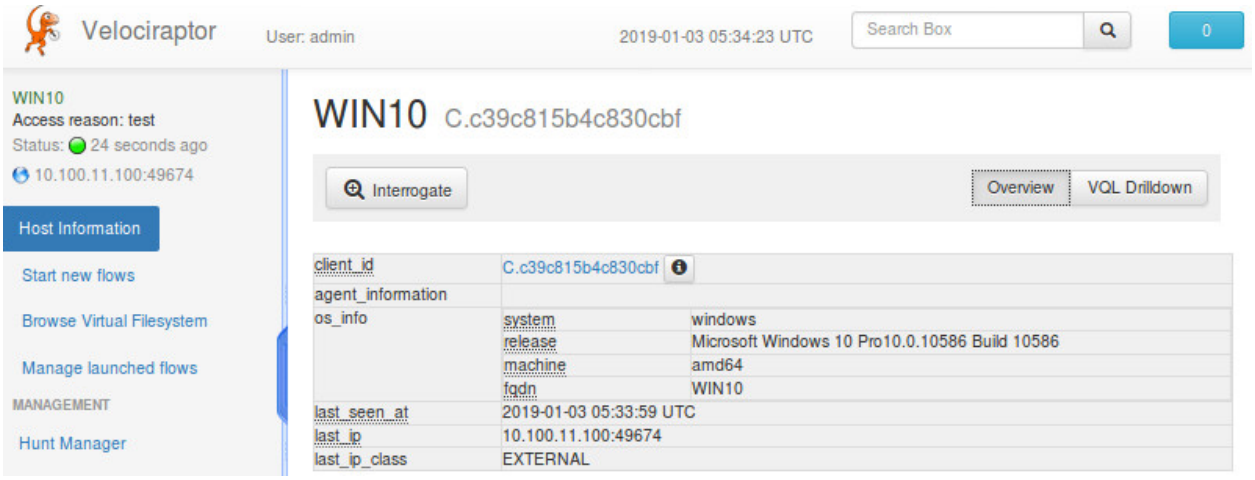
If you do so, you will be presented with the below:



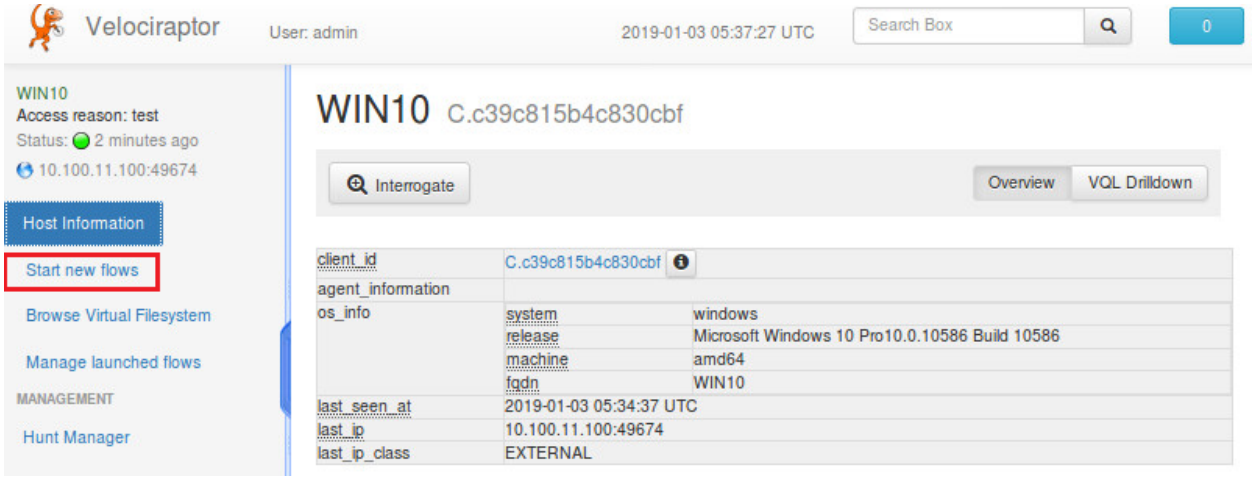
Important: Do not start any IR activities, until the bullet turns green! It may take some time until it does so... To refresh, click on the Velociraptor logo and then, once again click on the **Search box** and press **Enter**.

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).

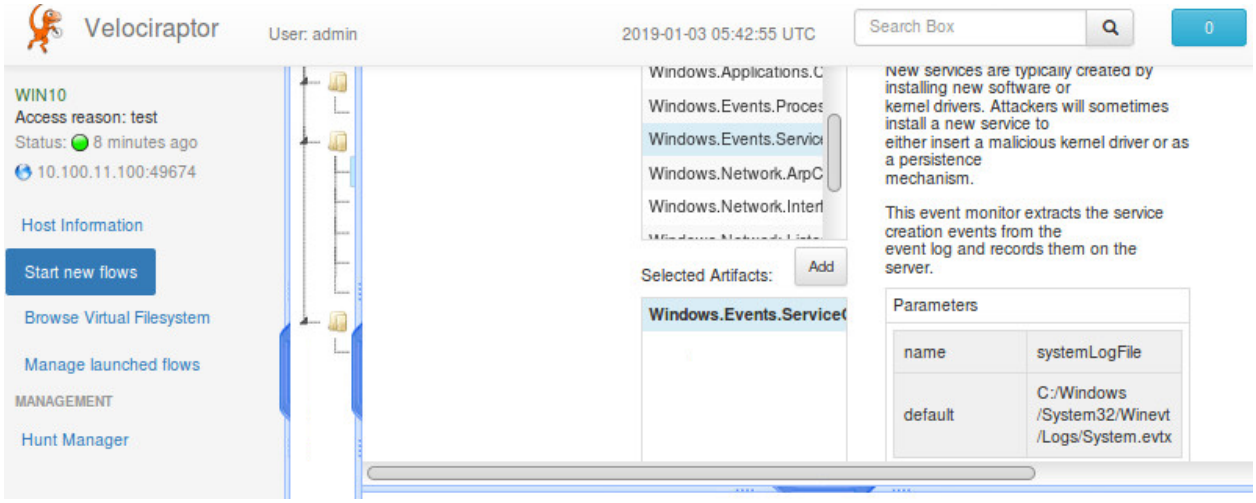
If you do so, you will be presented with the below.



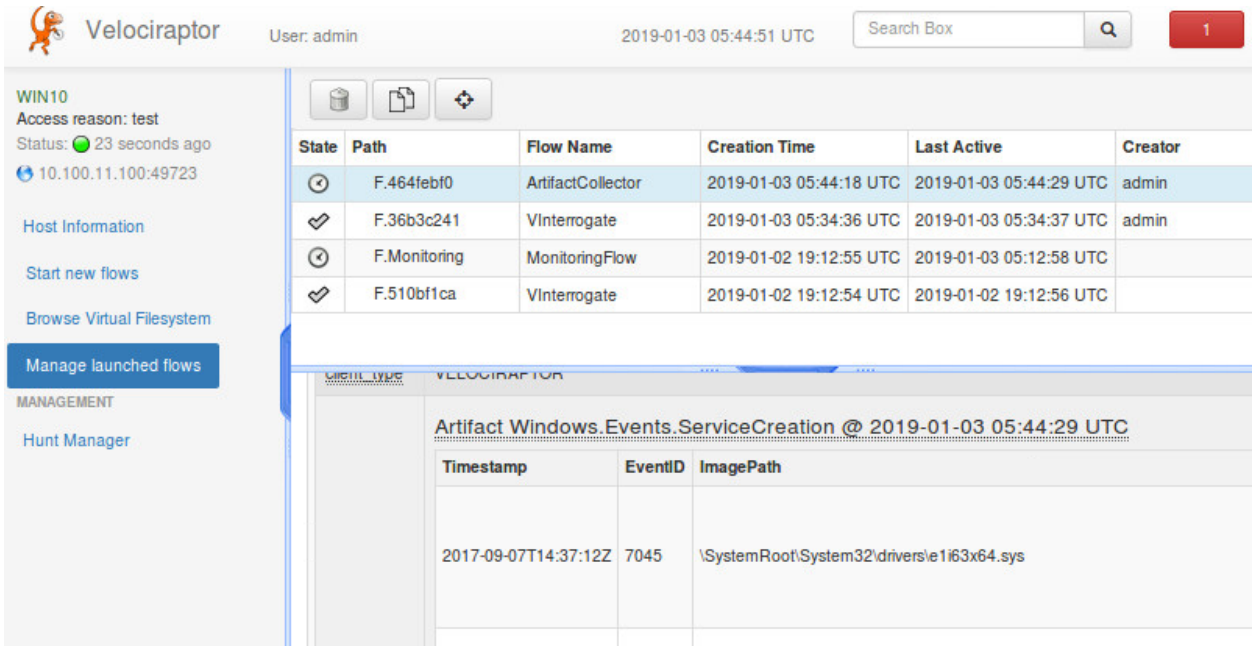
To start gathering important information about this endpoint, such as communications with other endpoints, newly created services etc., you can utilize Velociraptor **flows**. To do so, click on **Start new flows**.



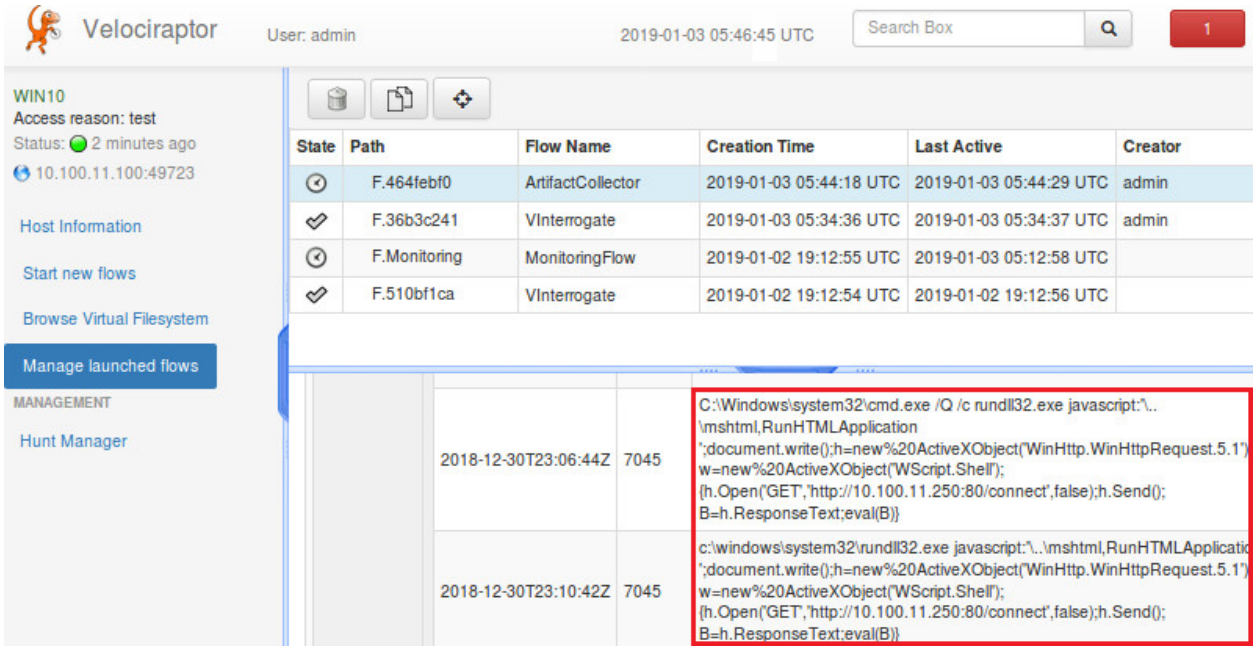
For example, to list all newly-created services on this endpoint, you should go to **Collectors -> Artifact Collector**, [add] **Windows.Events.ServiceCreation** and finally, scroll all the way down and press **Launch.**



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).



Now, if you carefully look at the results, you will notice some curious looking ones.

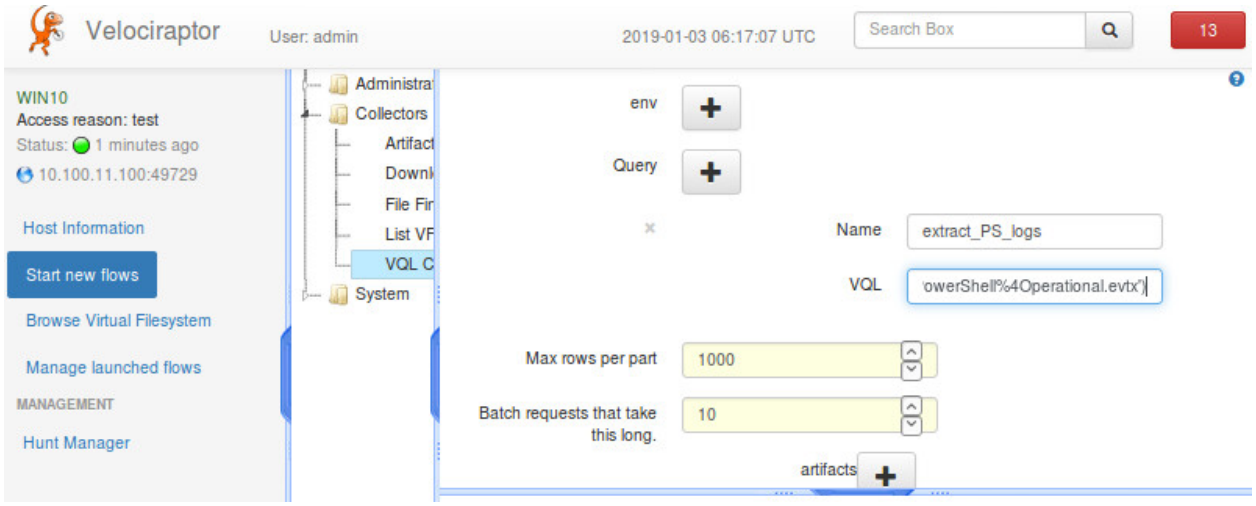


Specifically, once again you come across *rundll32* executing malicious JavaScript code. This time, persistence is achieved through the creation of a malicious service. If you scroll to the right, you will also see that the malicious service's name is *Log_Aggregator*.

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.

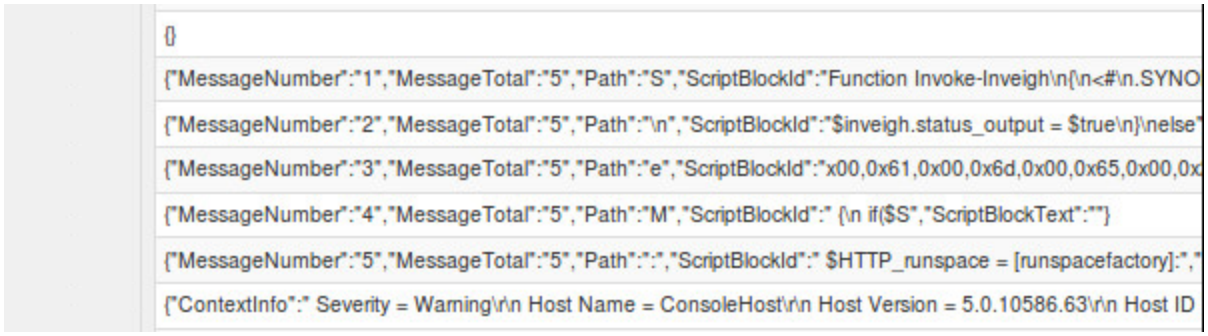
You can do so, by clicking on **Start new flows**, navigating to **Collectors** -> **VQL Collector** and specifying the query below (refer to https://docs.velociraptor.velocidex.com/blog/html/2018/11/09/event_queries_and_endpoint_monitoring.html for more information).

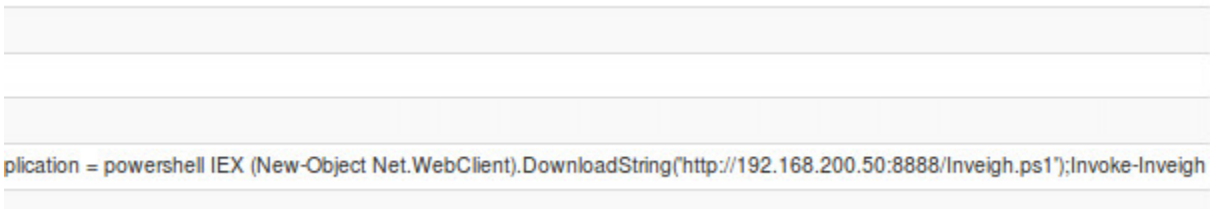
```
SELECT EventData, System.TimeCreated.SystemTime from
parse_evtx(filename=
'c:/windows/system32/winevt/logs/Microsoft-Windows-PowerShell
%4Operational.evtx')
```

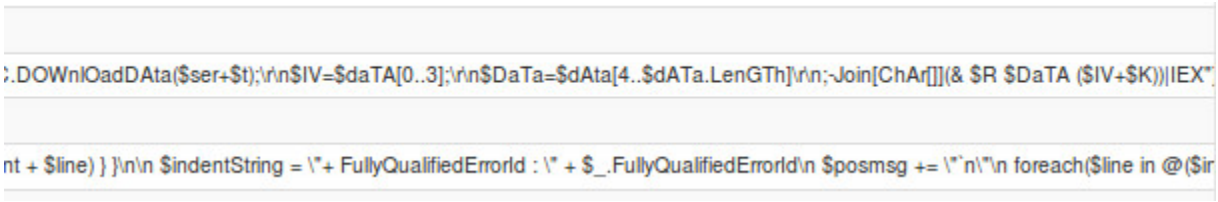


The results will appear in the **Manage launched flows** area, as we showed you previously.

The results are certainly interesting! Specifically, the ones below (output excerpt).







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