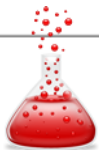




eLearnSecurity
Forging security professionals

SURICATA FUNDAMENTALS



LAB 4

LAB



1. 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!

2. 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

3. 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.

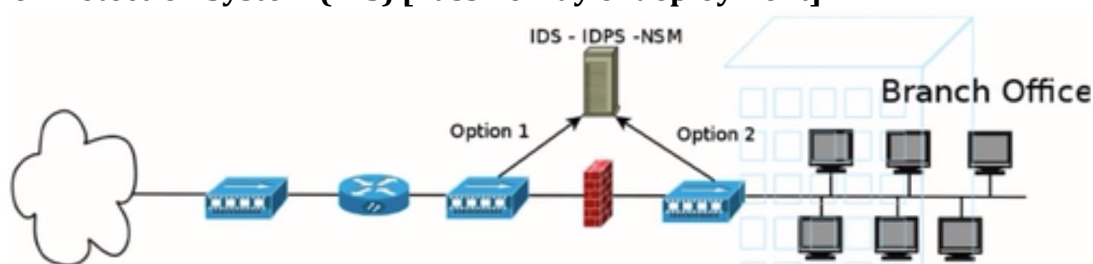


The most important Suricata features and capabilities are:

- 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:

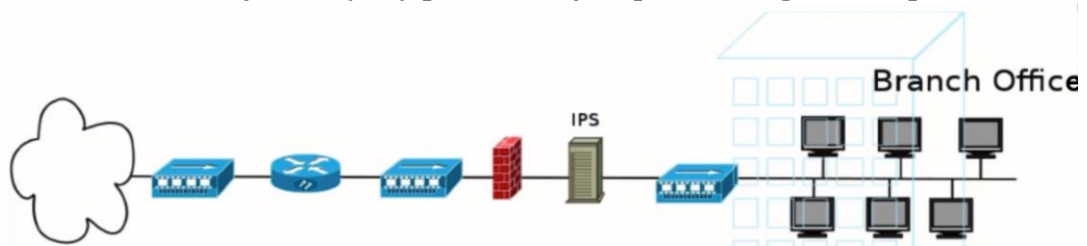
1. Intrusion Detection System (IDS) [Passive way of deployment]



- 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.



2. 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.

3. 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.

4. 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)

- 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.

2. Live Input

- **(Live) LibPCAP (Packets are read from network interfaces) [Passive IDS mode]**

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.

- **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.

4. RECOMMENDED TOOLS

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

5. NETWORK CONFIGURATION & CREDENTIALS

- Incident Responder's Subnet: **172.16.68.0/24**
- Suricata: **172.16.68.100**
- Connection Type: **SSH**
 - Use a Linux or Windows SSH client to connect to Suricata (172.16.68.100), as follows
 - Username: **elsanalyst**
 - Password: **@nalyst**

```
ssh elsanalyst@172.16.68.100 //For Linux-based machines
putty.exe, Session -> Host Name:172.16.68.100, Port: 22, Connection Type: SSH
// For Windows-based machines
```



6. 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 */home/elsanalyst* 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_archived.json*, *fast_archived.log* and *stats_archived.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 your command-line Fu...



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

Once you are connected to the deployed Suricata instance over SSH, you can list all Suricata rule files by executing the below.

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

You should see something similar to the following.

```
-rw-r--r-- 1 root root 3.2M Jan 23 15:15 emerging-trojan.rules
-rw-r--r-- 1 root root 139K Jan 23 15:15 emerging-user_agents.rules
-rw-r--r-- 1 root root 9.6K Jan 23 15:15 emerging-voip.rules
-rw-r--r-- 1 root root 214K Jan 23 15:15 emerging-web_client.rules
-rw-r--r-- 1 root root 263K Jan 23 15:15 emerging-web_server.rules
-rw-r--r-- 1 root root 3.7M Jan 23 15:15 emerging-web_specific_apps.rules
-rw-r--r-- 1 root root 10K Jan 23 15:15 emerging-worm.rules
-rw-r--r-- 1 root root 4.0K Dec 21 06:43 files.rules
-rw-r--r-- 1 root root 18K Jan 23 15:15 gpl-2.0.txt
-rw-r--r-- 1 root root 9.0K Dec 21 06:43 http-events.rules
-rw-r--r-- 1 root root 2.7K Dec 21 06:43 ipsec-events.rules
-rw-r--r-- 1 root root 585 Dec 21 06:43 kerberos-events.rules
-rw-r--r-- 1 root root 2.2K Jan 23 15:15 LICENSE
-rw-r--r-- 1 root root 2.1K Dec 21 06:43 modbus-events.rules
-rw-r--r-- 1 root root 558 Dec 21 06:43 nfs-events.rules
-rw-r--r-- 1 root root 558 Dec 21 06:43 ntp-events.rules
-rw-r--r-- 1 root root 3.6M Jan 23 15:15 sid-msg.map
-rw-r--r-- 1 root root 1.3K Dec 21 06:43 smb-events.rules
-rw-r--r-- 1 root root 4.9K Dec 21 06:43 smtp-events.rules
-rw-r--r-- 1 root root 13K Dec 21 06:43 stream-events.rules
-rw-r--r-- 1 root root 0 Jan 23 15:15 suricata-4.0-enhanced-open.txt
-rw-r--r-- 1 root root 5.1K Dec 21 06:43 tls-events.rules
-rw-r--r-- 1 root root 479K Jan 23 15:15 tor.rules
elsanalyst@training:~$
```

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 1024:65535 -> $HOME_NET any (msg:"ET TROJAN Bandoock v1.
2 Client Ping Reply"; flowbits:isset,BE.Bandoock1.2; flow:established,from_server
; dsize:10; content:"&SEXREPLY&"; offset:0; reference:url,www.nuclearwintercrew.
com; reference:url,research.sunbelt-software.com/threatdisplay.aspx?name=Bandoock
&threatid=40408; reference:url,doc.emergingthreats.net/bin/view/Main/TrojanBando
ok; classtype:trojan-activity; sid:2003554; rev:5; metadata:created_at 2010_07_3
0, updated_at 2010_07_30;)
```

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 1024:65535 (msg:"ET TROJAN Bandoock v1.3
5 Initial Connection and Report"; flowbits:isnotset,BE.Bandoock1.35; flow:establi
shed,to_server; content:"|cf 8f|"; offset:0; depth:2; content:"|20 26 26 26|"; d
istance:50; flowbits:set,BE.Bandoock1.35; reference:url,www.nuclearwintercrew.com
; reference:url,research.sunbelt-software.com/threatdisplay.aspx?name=Bandoock&th
reatid=40408; reference:url,doc.emergingthreats.net/bin/view/Main/TrojanBandoock;
 classtype:trojan-activity; sid:2003555; rev:5; metadata:created_at 2010_07_30,
updated_at 2010_07_30;)
```



The rule inside the red rectangle 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](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.

```
////////////////////////////////////  
sudo su <- Submit the elsanalyst user's password you used to SSH into Suricata (@nalyst)  
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 /home/elsanalyst  
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.

```
////////////////////////////////////  
sudo suricata -r PCAPs/eicar-com.pcap  
////////////////////////////////////
```

You should see the following.



```

elsanalyst@training:~$ sudo suricata -r PCAPs/eicar-com.pcap
3/2/2019 -- 11:42:20 - <Notice> - This is Suricata version 4.1.2 RELEASE
3/2/2019 -- 11:42:20 - <Notice> - all 2 packet processing threads, 4 management
threads initialized, engine started.
3/2/2019 -- 11:42:20 - <Notice> - Signal Received. Stopping engine.
3/2/2019 -- 11:42:21 - <Notice> - Pcap-file module read 1 files, 10 packets, 106
8 bytes

```

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

```

elsanalyst@training:/var/log/suricata$ ls -lah
total 32K
drwxr-xr-x  5 root root  4.0K Feb  3 11:42 .
drwxrwxr-x 14 root syslog 4.0K Feb  3 07:35 ..
drwxr-xr-x  2 root root  4.0K Dec 21 07:11 certs
drwxr-xr-x  2 root root  4.0K Dec 21 07:11 core
-rw-r--r--  1 root root  4.0K Feb  3 11:42 eve.json
-rw-r--r--  1 root root    0 Feb  3 11:42 fast.log
drwxr-xr-x  2 root root  4.0K Dec 21 07:11 files
-rw-r--r--  1 root root  1.9K Feb  3 11:42 stats.log
-rw-r--r--  1 root root  347 Feb  3 11:42 suricata.log

```

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

```

sudo suricata -r PCAPs/eicar-com.pcap -k none -l .

```

You should see the below.

```

elsanalyst@training:~$ sudo suricata -r PCAPs/eicar-com.pcap -k none -l .
3/2/2019 -- 11:59:17 - <Notice> - This is Suricata version 4.1.2 RELEASE
3/2/2019 -- 11:59:17 - <Notice> - all 2 packet processing threads, 4 management
threads initialized, engine started.
3/2/2019 -- 11:59:17 - <Notice> - Signal Received. Stopping engine.
3/2/2019 -- 11:59:18 - <Notice> - Pcap-file module read 1 files, 10 packets, 106
8 bytes
elsanalyst@training:~$ ls
customsig.rules Desktop Downloads eve.json fast.log PCAPs stats.log

```



2. Live Input

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

```
ifconfig <- To identify the network interface Suricata will listen on
sudo suricata --pcap=ens33 -vv
```

You should see something similar to the below screenshot.

```
elsanalyst@training:~$ ifconfig
ens33    Link encap:Ethernet  HWaddr 00:50:56:91:28:08
          inet addr:172.16.68.100  Bcast:172.16.68.255  Mask:255.255.255.0
          inet6 addr: fe80::51b9:1dda:814d:7aaa/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11599 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9434 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:989593 (989.5 KB)  TX bytes:4827919 (4.8 MB)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:2673722 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2673722 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:197876426 (197.8 MB)  TX bytes:197876426 (197.8 MB)

elsanalyst@training:~$ sudo suricata --pcap=ens33 -vv
3/2/2019 -- 12:23:52 - <Notice> - This is Suricata version 4.1.2 RELEASE
3/2/2019 -- 12:23:52 - <Info> - CPUs/cores online: 1
3/2/2019 -- 12:23:52 - <Info> - fast output device (regular) initialized: fast.l
og
3/2/2019 -- 12:23:52 - <Info> - eve-log output device (regular) initialized: eve
.json
3/2/2019 -- 12:23:52 - <Info> - stats output device (regular) initialized: stats
.log
3/2/2019 -- 12:23:52 - <Info> - Running in live mode, activating unix socket
3/2/2019 -- 12:23:52 - <Info> - 1 rule files processed. 1 rules successfully loa
ded, 0 rules failed
```

Alternatively, you can execute the following.

```
ifconfig <- To identify the network interface Suricata will listen on
sudo suricata -i ens33 -vv
```



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

```
3/2/2019 -- 12:27:15 - <Perf> - 1 cores, so using 1 threads
3/2/2019 -- 12:27:15 - <Perf> - Using 1 AF PACKET threads for interface ens33
```

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).

```
////////////////////////////////////
sudo suricata -q 0
////////////////////////////////////
```

You should see something similar to the below.

```
elsanalyst@training:~$ sudo suricata -q 0
[sudo] password for elsanalyst:
3/2/2019 -- 13:17:09 - <Notice> - This is Suricata version 4.1.2 RELEASE
3/2/2019 -- 13:17:09 - <Notice> - all 3 packet processing threads, 4 management
threads initialized, engine started.
```

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

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

```
////////////////////////////////////
sudo suricata -i ens33
sudo suricata -af-packet=ens33
////////////////////////////////////
```



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

1. eve.json

Inside the `/var/log/suricata` directory, you will find a file named `eve_archived.json`. It is an older `eve.json` file that was archived so that you can experiment on it.

You can start inspecting it, by executing the following.

```
less eve_archived.json
```

You should see something similar to the below.

```
{ "timestamp": "2017-04-20T14:15:58.732253-0700", "flow_id": 1023783386098781, "pcap_cnt": 1, "event_type": "dns", "src_ip": "10.16.1.11", "src_port": 41805, "dest_ip": "10.16.1.1", "dest_port": 53, "proto": "UDP", "dns": { "type": "query", "id": 36146, "rrname": "www.suricata-ids.org", "rrtype": "A", "tx_id": 0 } }
{ "timestamp": "2017-04-20T14:15:58.732859-0700", "flow_id": 1023783386098781, "pcap_cnt": 2, "event_type": "dns", "src_ip": "10.16.1.1", "src_port": 53, "dest_ip": "10.16.1.11", "dest_port": 41805, "proto": "UDP", "dns": { "version": 2, "type": "answer", "id": 36146, "flags": "81a0", "qr": true, "rd": true, "ra": true, "rrname": "www.suricata-ids.org", "rrtype": "A", "rcode": "NOERROR", "answers": [ { "rrname": "www.suricata-ids.org", "rrtype": "CNAME", "ttl": 3544, "rdata": "suricata-ids.org" }, { "rrname": "suricata-ids.org", "rrtype": "CNAME", "ttl": 3544, "rdata": "suricata-ids.org" } ] } }
```

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

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

```
{
  "timestamp": "2016-07-21T19:17:57.575378-0700",
  "flow_id": 9657273141138,
  "pcap_cnt": 519,
  "event_type": "alert",
  "src_ip": "10.0.2.15",
  "src_port": 1046,
  "dest_ip": "31.184.235.253",
  "dest_port": 6892,
  "proto": "UDP",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 102,
    "rev": 1,
    "signature": "ET TROJAN Cerber UDP 'hi' Checkin",
    "category": "A Network Trojan was detected",
    "severity": 1
  },
  "app_proto": "failed",
  "flow": {
    "pkts_toserver": 1,
    "pkts_toclient": 0,
    "bytes_toserver": 60,
    "bytes_toclient": 0,
    "start": "2016-07-21T19:17:57.575378-0700"
  }
}
```

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

If you do so, you will see the below.

```
{
  "src_ip": "10.0.2.15",
  "src_port": 1046,
  "dest_ip": "31.184.235.253",
  "dest_port": 6892,
  "proto": "UDP",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 102,
    "rev": 1,
    "signature": "ET TROJAN Cerber UDP 'hi' Checkin",
    "category": "A Network Trojan was detected",
    "severity": 1
  },
  "app_proto": "failed",
  "flow": {
    "pkts_toserver": 1,
    "pkts_toclient": 0,
    "bytes_toserver": 60,
    "bytes_toclient": 0,
    "start": "2016-07-21T19:17:58.574376-0700"
  }
}
```



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

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

You should see the below.

```
root@training:/var/log/suricata# cat eve_archived.json | jq -c 'select(.event_type == "tls")'
{"timestamp": "2017-06-26T14:38:20.190199-0700", "flow_id": 1446465508221150, "pcap_cnt": 11, "event_type": "tls", "src_ip": "10.16.1.11", "src_port": 52614, "dest_ip": "192.0.78.24", "dest_port": 443, "proto": "TCP", "tls": {"subject": "CN=tls.automattic.com", "issuerdn": "C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3", "serial": "03:43:E7:2B:25:CA:70:36:F1:B9:65:3A:9D:1D:E9:56:47:BF", "fingerprint": "7b:fd:8d:cf:69:c0:89:d6:98:9c:cd:b3:b7:04:dd:a8:91:2c:ae:16", "sni": "suricata-ids.org", "version": "TLS 1.2", "notbefore": "2017-05-07T20:18:00", "notafter": "2017-08-05T20:18:00", "ja3": {}}}
```

Try doing the same for SSH and DNS events...

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.

```
sudo su <- Submit the elsanalyst user's password you used to SSH into Suricata (@analyst)
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*.

```
////////////////////////////////////  
sudo suricata -r PCAPs/tls-suricata-ids-org.pcap  
sudo su  
cd /var/log/suricata  
////////////////////////////////////
```

You will see something similar to the below.

```
elsanalyst@training:~$ sudo suricata -r PCAPs/tls-suricata-ids-org.pcap  
3/2/2019 -- 15:53:58 - <Notice> - This is Suricata version 4.1.2 RELEASE  
3/2/2019 -- 15:53:58 - <Notice> - all 2 packet processing threads, 4 management  
threads initialized, engine started.  
3/2/2019 -- 15:53:58 - <Notice> - Signal Received. Stopping engine.  
3/2/2019 -- 15:53:59 - <Notice> - Pcap-file module read 1 files, 40 packets, 166  
76 bytes  
elsanalyst@training:~$ sudo su  
root@training:/home/elsanalyst# ls /var/log/suricata/  
certs eve_archived.json fast_archived.log files stats.log  
core eve.json fast.log http.log suricata.log
```

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.

2. 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_archived.log*. It is an older *fast.log* file that was archived so that you can experiment on it.



You can start inspecting it by executing the following.

```
cat fast_archived.log
```

You should see the below.

```
] [Classification: A Network Trojan was detected] [Priority: 1] {UDP} 10.0.2.15:
1046 -> 31.184.235.248:6892
07/21/2016-19:17:57.574593  [**] [1:102:1] ET TROJAN Cerber UDP 'hi' Checkin [**
] [Classification: A Network Trojan was detected] [Priority: 1] {UDP} 10.0.2.15:
1046 -> 31.184.235.249:6892
07/21/2016-19:17:57.574785  [**] [1:102:1] ET TROJAN Cerber UDP 'hi' Checkin [**
] [Classification: A Network Trojan was detected] [Priority: 1] {UDP} 10.0.2.15:
1046 -> 31.184.235.250:6892
```

3. 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_archived.log*. It is an older *stats.log* file that was archived so that you can experiment on it.

You can start inspecting it, by executing the following.

```
less stats_archived.log
```

You should see the below.



```
-----
Date: 2/3/2019 -- 15:28:06 (uptime: 0d, 00h 00m 01s)
-----
```

Counter	TM Name	Value
decoder.pkts	Total	2
decoder.bytes	Total	233
decoder.ipv4	Total	2
decoder.ethernet	Total	2
decoder.udp	Total	2
decoder.avg_pkt_size	Total	116
decoder.max_pkt_size	Total	142
flow.udp	Total	1

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_archived.json*

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

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

```
elsanalyst@training:~$ sudo cat /var/log/suricata/eve_archived.json | jq -c '.event_type'|sort|uniq -c|sort -nr
[sudo] password for elsanalyst:
1061 "flow"
 512 "alert"
  28 "dns"
  19 "http"
  16 "fileinfo"
```



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

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

```
elsanalyst@training:~$ sudo cat /var/log/suricata/eve_archived.json | jq -c 'select(.alert)' | tail -1 | jq .
{
  "timestamp": "2016-07-21T19:17:58.574376-0700",
  "flow_id": 839578098779048,
  "pcap_cnt": 521,
  "event_type": "alert",
  "src_ip": "10.0.2.15",
  "src_port": 1046,
  "dest_ip": "31.184.235.255",
  "dest_port": 6892,
  "proto": "UDP",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 102,
    "rev": 1,
    "signature": "ET TROJAN Cerber UDP 'hi' Checkin",

```

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

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

```
elsanalyst@training:~$ sudo cat /var/log/suricata/eve_archived.json | jq -c 'select(.http)' | tail -1 | jq .
{
  "timestamp": "2016-07-14T12:55:57.822470-0700",
  "flow_id": 1323489580292764,
  "pcap_cnt": 290,
  "event_type": "fileinfo",
  "src_ip": "173.237.190.2",
  "src_port": 80,
  "dest_ip": "172.16.57.213",
  "dest_port": 2263,
  "proto": "TCP",
  "http": {
    "hostname": "www.montearts.com",
    "url": "/wp-includes/adobecloud/ao.php",
    "http_user_agent": "Mozilla/5.0 (Windows NT 5.1; rv:43.0) Gecko/20100101 Firefox/43.0",

```

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_archived.json* through EveBox, execute the below.

```
sudo evebox oneshot /var/log/suricata/eve_archived.json
```

You will come across the following.

```
2019-02-04 04:21:10 (oneshot.go:241) <Info> -- /var/log/suricata/eve_archived.js
on: 1602 events (96%)
2019-02-04 04:21:10 (oneshot.go:241) <Info> -- /var/log/suricata/eve_archived.js
on: 1615 events (97%)
2019-02-04 04:21:10 (oneshot.go:241) <Info> -- /var/log/suricata/eve_archived.js
on: 1627 events (98%)
2019-02-04 04:21:10 (oneshot.go:241) <Info> -- /var/log/suricata/eve_archived.js
on: 1642 events (99%)
2019-02-04 04:21:11 (oneshot.go:241) <Info> -- /var/log/suricata/eve_archived.js
on: 1654 events (100%)
2019-02-04 04:21:11 (oneshot.go:285) <Info> -- Starting server.
2019-02-04 04:21:11 (server.go:131) <Info> -- Session reaper started
2019-02-04 04:21:11 (server.go:165) <Info> -- Authentication disabled.
2019-02-04 04:21:11 (server.go:276) <Info> -- Listening on 127.0.0.1:5636
2019-02-04 04:21:11 (oneshot.go:333) <Info> -- Bound to port 5636
2019-02-04 04:21:11 (oneshot.go:337) <Info> -- Attempting to start browser.

If your browser didn't open, go to http://localhost:5636

** Press CTRL-C to exit and cleanup.. **
```

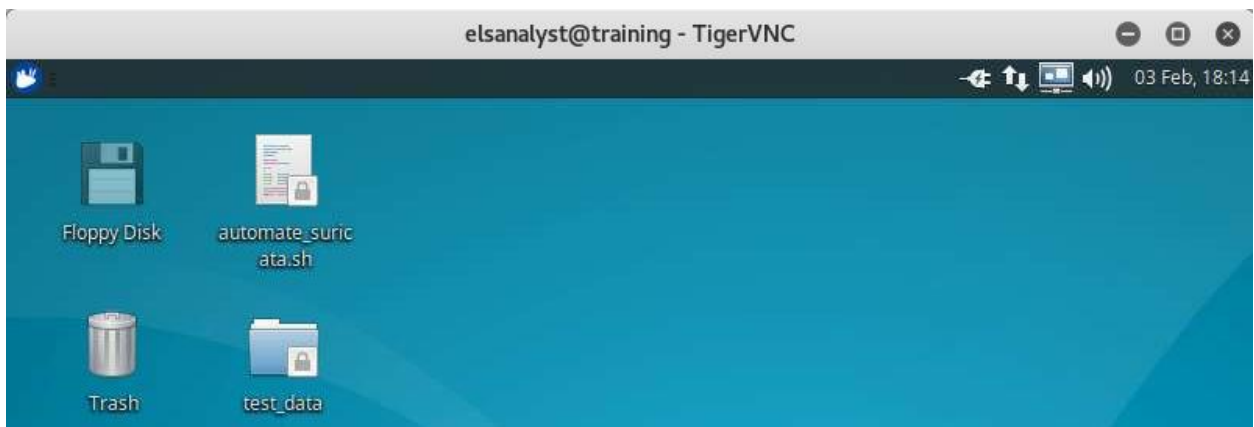
Then, on your host machine (not inside the SSH session), execute this:

```
vncviewer 172.16.68.100 ← For Linux-based machines
```

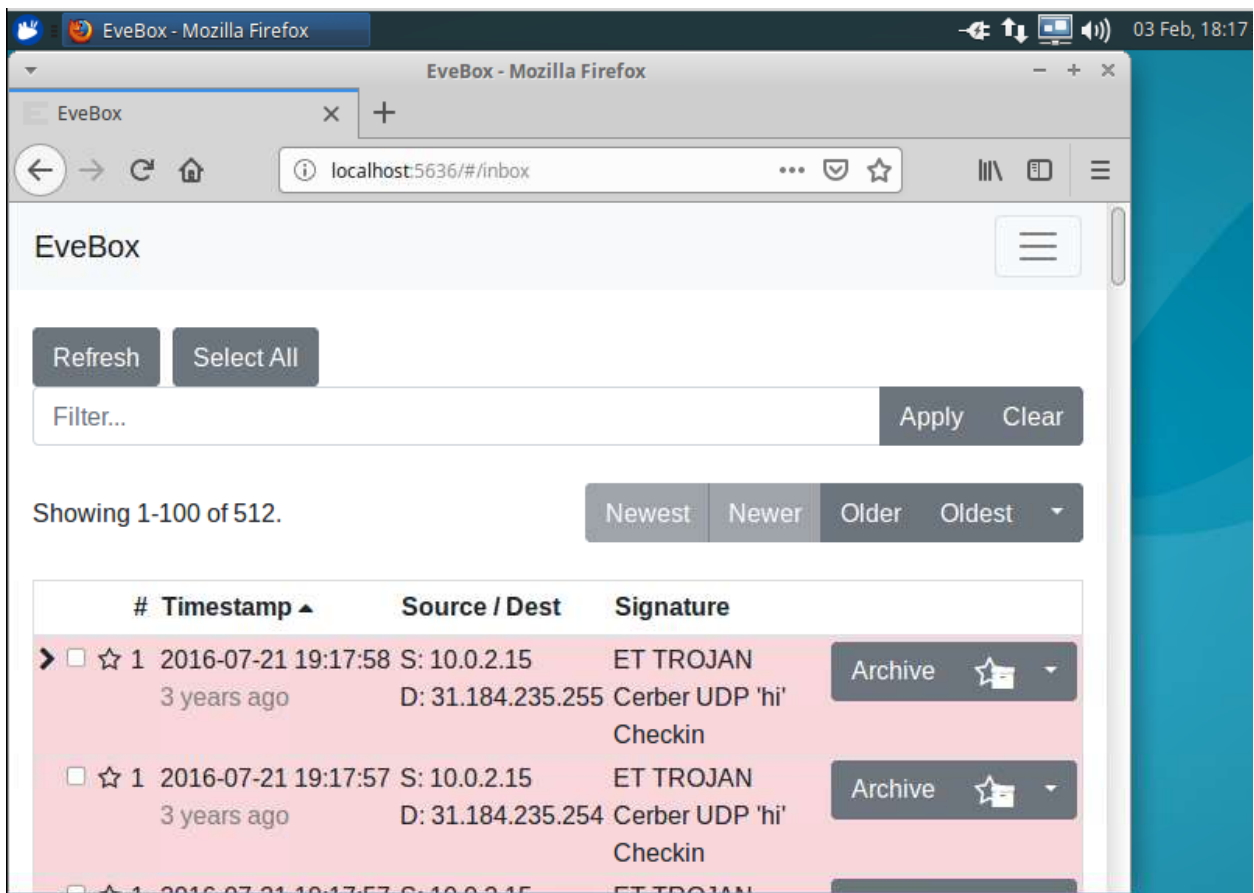
```
tvnviewer.exe, Remote Host: 172.16.68.100 ← For Windows-based machines
```

You will be presented with the below.





Now, launch a web browser and navigate to <http://localhost:5636>, as follows.

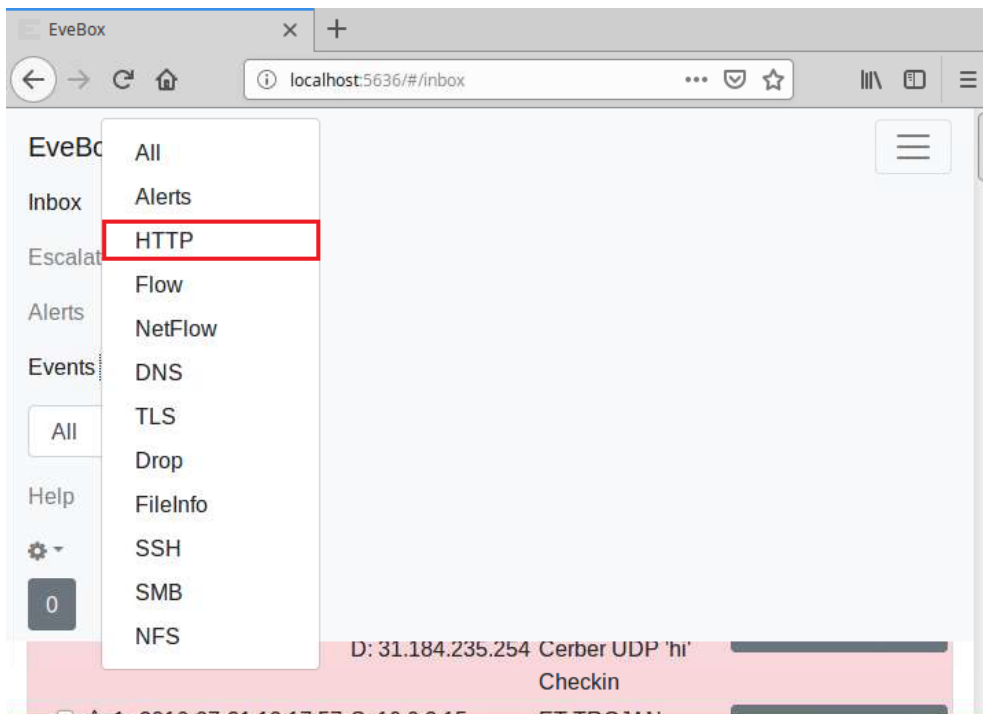


You will notice that EveBox parsed the whole *eve_archived.json* and presented it to you in a much more organized way. Feel free to look around and get familiar with EveBox's layout and capabilities.

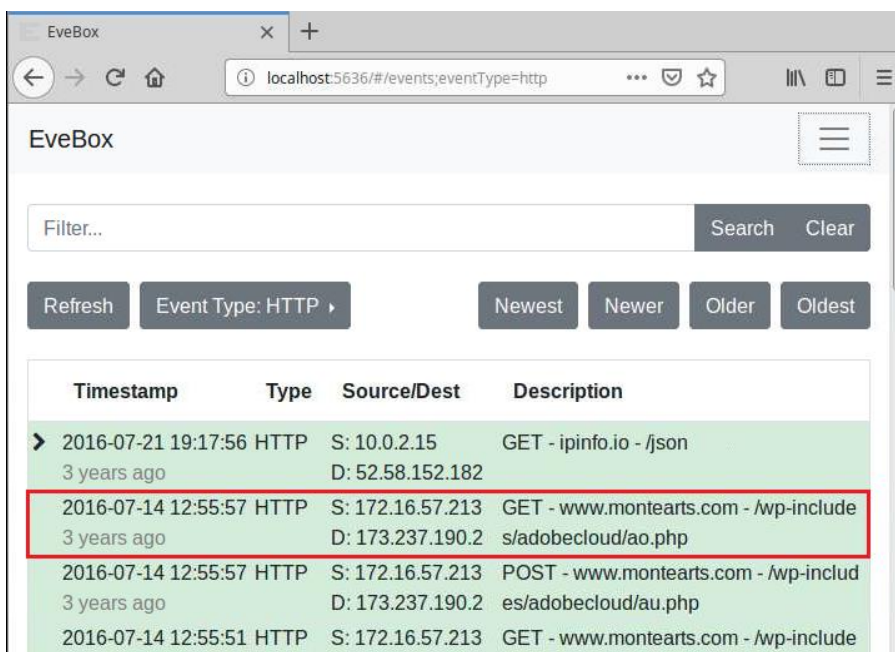


Let's see an example of EveBox's analysis capabilities.

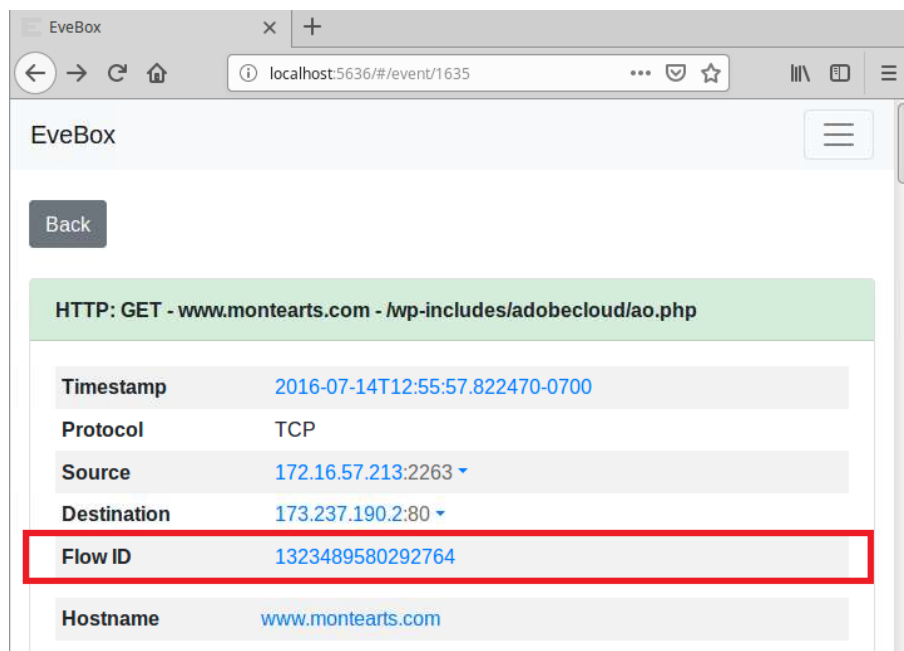
Try the following... First, list all HTTP events.



Then, choose the third entry.



Finally, click on the Flow ID.



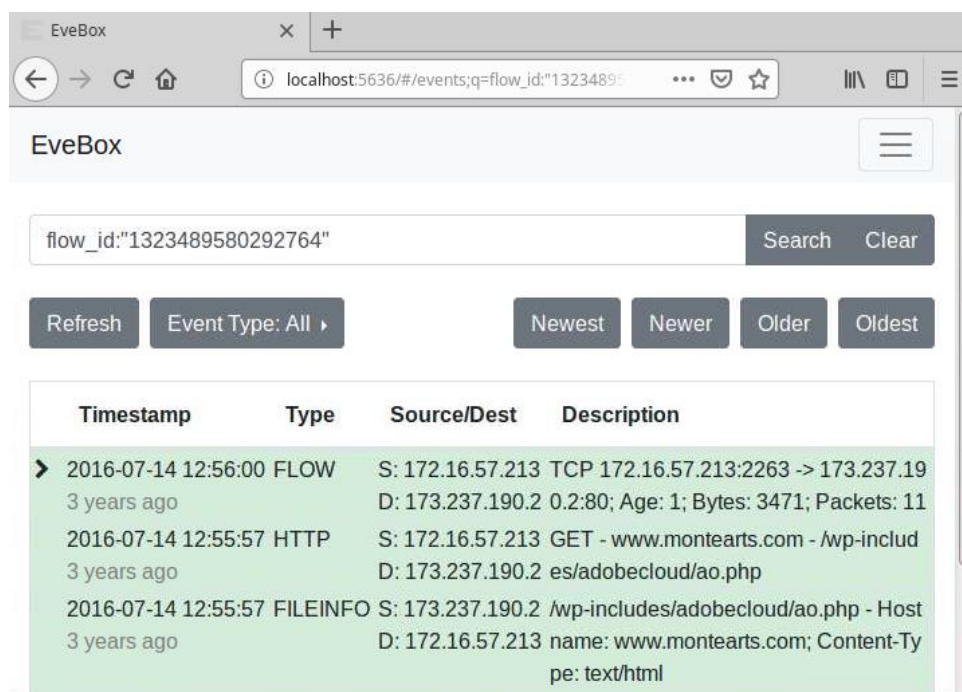
EveBox

Back

HTTP: GET - www.monteararts.com - /wp-includes/adobecloud/ao.php

Timestamp	2016-07-14T12:55:57.822470-0700
Protocol	TCP
Source	172.16.57.213:2263
Destination	173.237.190.2:80
Flow ID	1323489580292764
Hostname	www.monteararts.com

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.



EveBox

flow_id:"1323489580292764" Search Clear

Refresh Event Type: All Newest Newer Older Oldest

Timestamp	Type	Source/Dest	Description
2016-07-14 12:56:00 3 years ago	FLOW	S: 172.16.57.213 D: 173.237.190.2	TCP 172.16.57.213:2263 -> 173.237.190.2:80; Age: 1; Bytes: 3471; Packets: 11
2016-07-14 12:55:57 3 years ago	HTTP	S: 172.16.57.213 D: 173.237.190.2	GET - www.monteararts.com - /wp-includes/adobecloud/ao.php
2016-07-14 12:55:57 3 years ago	FILEINFO	S: 173.237.190.2 D: 172.16.57.213	/wp-includes/adobecloud/ao.php - Host name: www.monteararts.com; Content-Type: text/html



Additional Resources:

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

