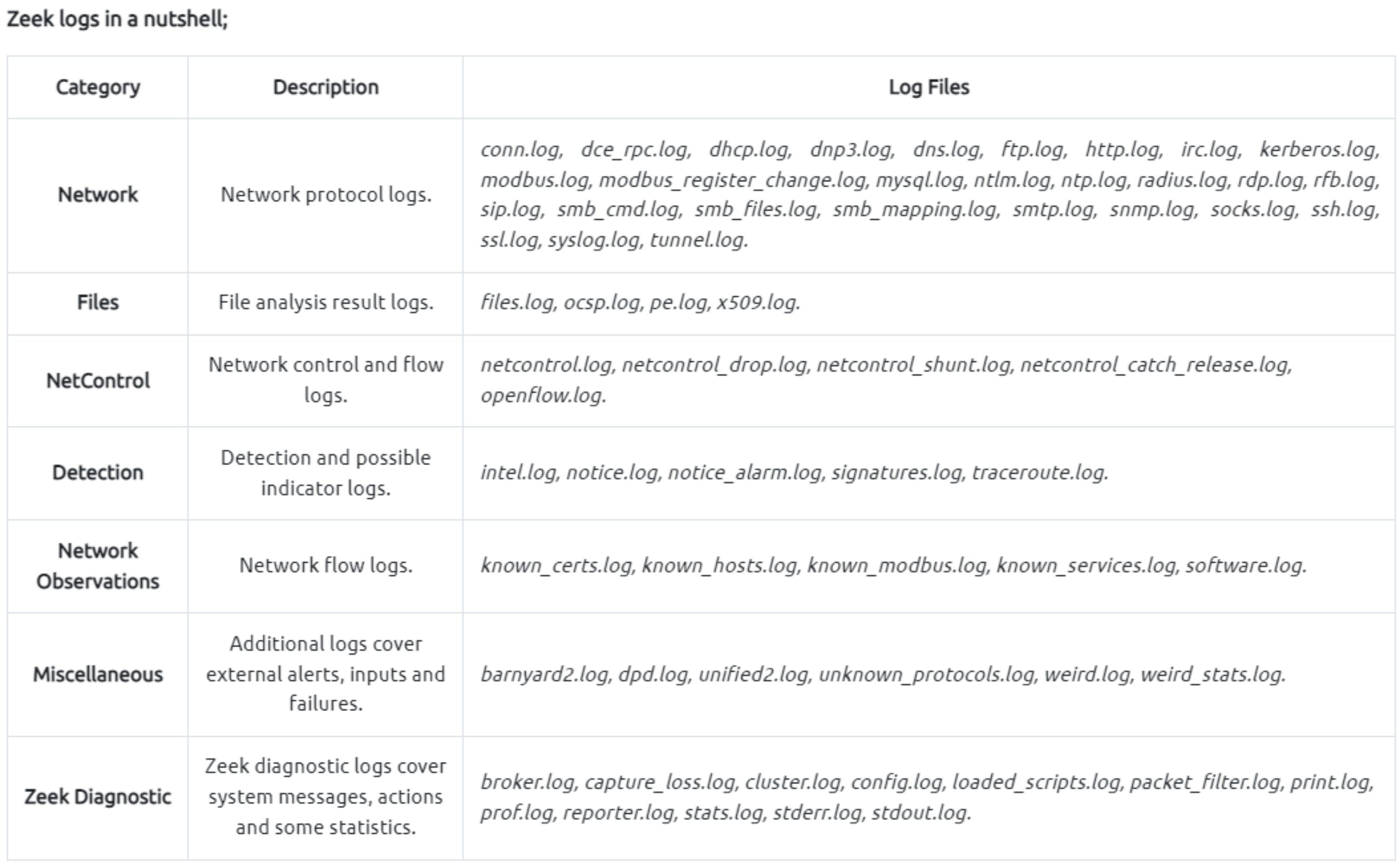
# *Zeek*

-Each log output consists of multiple fields, and each field holds a different part of the traffic data. Correlation is done through a unique value called "UID". The "UID" represents the unique identifier assigned to each session.



Zeek logs in a nutshell;

| Category | Description | Log Files |
|---|---|---|
| **Network** | Network protocol logs. | *conn.log, dce_rpc.log, dhcp.log, dnp3.log, dns.log, ftp.log, http.log, irc.log, kerberos.log, modbus.log, modbus_register_change.log, mysql.log, ntlm.log, ntp.log, radius.log, rdp.log, rfb.log, sip.log, smb_cmd.log, smb_files.log, smb_mapping.log, smtp.log, snmp.log, socks.log, ssh.log, ssl.log, syslog.log, tunnel.log.* |
| **Files** | File analysis result logs. | *files.log, ocsp.log, pe.log, x509.log.* |
| **NetControl** | Network control and flow logs. | *netcontrol.log, netcontrol_drop.log, netcontrol_shunt.log, netcontrol_catch_release.log, openflow.log.* |
| **Detection** | Detection and possible indicator logs. | *intel.log, notice.log, notice_alarm.log, signatures.log, traceroute.log.* |
| **Network Observations** | Network flow logs. | *known_certs.log, known_hosts.log, known_modbus.log, known_services.log, software.log.* |
| **Miscellaneous** | Additional logs cover external alerts, inputs and failures. | *barnyard2.log, dpd.log, unified2.log, unknown_protocols.log, weird.log, weird_stats.log.* |
| **Zeek Diagnostic** | Zeek diagnostic logs cover system messages, actions and some statistics. | *broker.log, capture_loss.log, cluster.log, config.log, loaded_scripts.log, packet_filter.log, print.log, prof.log, reporter.log, stats.log, stderr.log, stdout.log.* |

Please refer to Zeek's official documentation and Corelight log cheat sheet for more information. Although there are multiple log files, some log files are updated daily, and some are updated in each session. Some of the most commonly used logs are explained in the given table.

| Update Frequency | Log Name | Description |
|---|---|---|
| **Daily** | *known_hosts.log* | List of hosts that completed TCP handshakes. |
| **Daily** | *known_services.log* | List of services used by hosts. |
| **Daily** | *known_certs.log* | List of SSL certificates. |
| **Daily** | *software.log* | List of software used on the network. |
| **Per Session** | *notice.log* | Anomalies detected by Zeek. |
| **Per Session** | *intel.log* | Traffic contains malicious patterns/indicators. |
| **Per Session** | *signatures.log* | List of triggered signatures. |

| Tool/Auxilary Name | Purpose |
|---|---|
| **Zeek-cut** | Cut specific columns from zeek logs. |

Let's see the "zeek-cut" in action. Let's extract the uid, protocol, source and destination hosts, and source and destination ports from the conn.log. We will first read the logs with the `cat` command and then extract the event of interest fields with `zeek-cut` auxiliary to compare the difference.

zeek-cut usage example

```
root@ubuntu$ cat conn.log
...
#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]
1488571051.943250   CTMFXm1AcIsSnq2Ric  192.168.121.2   51153   192.168.120.22  53  udp dns 0.001263    36  106 SF  -
-0  Dd  1   64  1   134 -
1488571038.380901   CLsSsA3HLB2N6uJwW   192.168.121.10  50080   192.168.120.10  514 udp -   0.000505    234 0   S0  -
-0  D   2   290 0   0   -

root@ubuntu$ cat conn.log | zeek-cut uid proto id.orig_h id.orig_p id.resp_h id.resp_p
CTMFXm1AcIsSnq2Ric  udp 192.168.121.2   51153   192.168.120.22  53
CLsSsA3HLB2N6uJwW   udp 192.168.121.10  50080   192.168.120.10  514
```

As shown in the above output, the "zeek-cut" auxiliary provides massive help to extract specific fields with minimal effort. Now take time to read log formats, practice the log reading/extracting operations and answer the questions.

## *Zeek Scripts*

Zeek has its own event-driven scripting language, which is as powerful as high-level languages and allows us to investigate and correlate the detected events. Since it is as capable as high-level programming languages, you will need to spend time on

Zeek scripting language in order to become proficient. In this room, we will cover the basics of Zeek scripting to help you understand, modify and create basic scripts. Note that scripts can be used to apply a policy and in this case, they are called policy scripts.

| | |
|---|---|
| Zeek has base scripts installed by default, and these are not intended to be modified. | These scripts are located in **"/opt/zeek/share/zeek/base"**. |
| User-generated or modified scripts should be located in a specific path. | These scripts are located in **"/opt/zeek/share/zeek/site"**. |
| Policy scripts are located in a specific path. | These scripts are located in **"/opt/zeek/share/zeek/policy"**. |
| Like Snort, to automatically load/use a script in live sniffing mode, you must identify the script in the Zeek configuration file. You can also use a script for a single run, just like the signatures. | The configuration file is located in **"/opt/zeek/share/zeek/site/local.zeek"**. |

- Zeek scripts use the ".zeek" extension.
- Do not modify anything under the "zeek/base" directory. User-generated and modified scripts should be in the "zeek/site" directory.
- You can call scripts in live monitoring mode by loading them with the command `load @/script/path` or `load @script-name` in local.zeek file.
- Zeek is event-oriented, not packet-oriented! We need to use/write scripts to handle the event of interest.

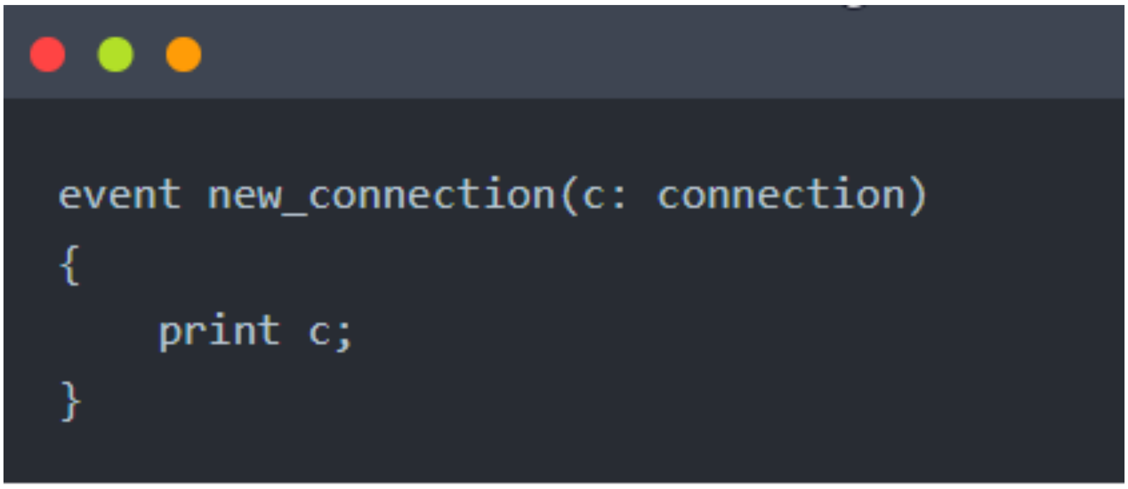
Scripts contain operators, types, attributes, declarations and statements, and directives. Let's look at a simple example event called "zeek_init" and "zeek_done". These events work once the Zeek process starts and stops. Note that these events don't have parameters, and some events will require parameters.

▼ View Script

Sample Script

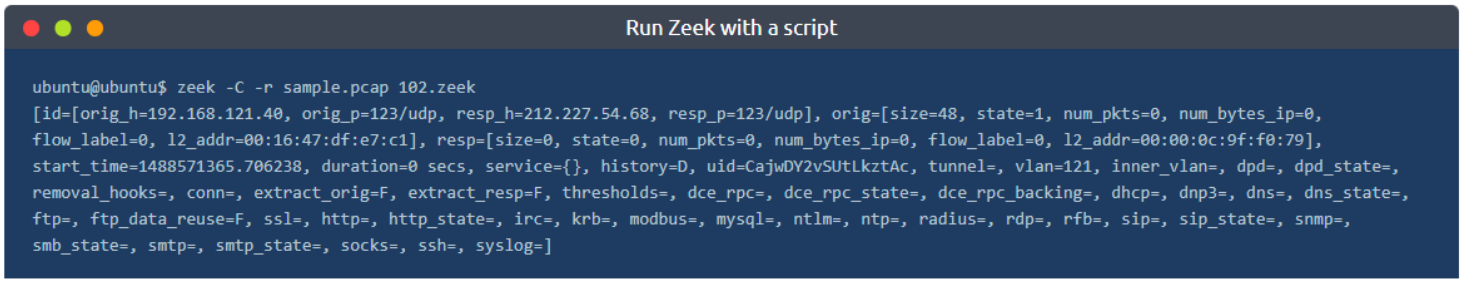
```
event zeek_init()
    {
     print ("Started Zeek!");
    }
event zeek_done()
    {
    print ("Stopped Zeek!");
    }

# zeek_init: Do actions once Zeek starts its process.
# zeek_done: Do activities once Zeek finishes its process.
# print: Prompt a message on the terminal.
```

Let's print the packet data to the terminal and see the raw data. In this script, we are requesting details of a connection and extracting them without any filtering or sorting of the data. To accomplish this, we are using the "new_connection" event. This event is automatically generated for each new connection. This script provides bulk information on the terminal. We need to get familiar with Zeek's data structure to reduce the amount of information and focus on the event of interest. To do so, we need to investigate the bulk data.



```
event new_connection(c: connection)
{
      print c;
}
```

Run Zeek with the script

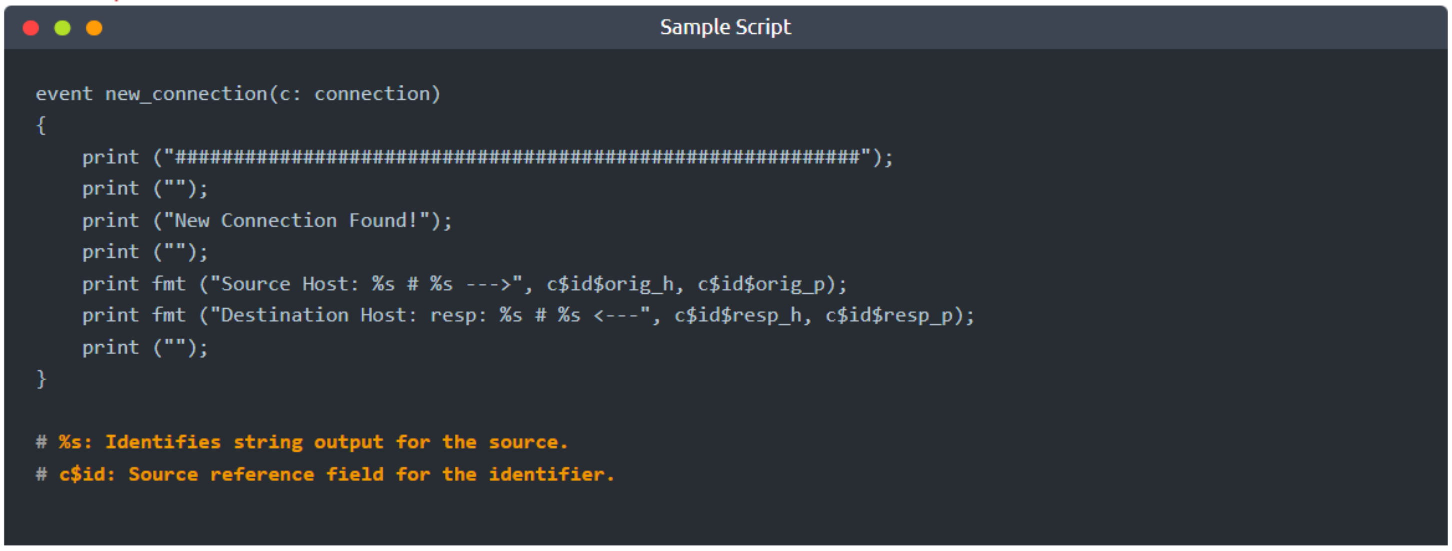


Run Zeek with a script

```
ubuntu@ubuntu$ zeek -C -r sample.pcap 102.zeek
[id=[orig_h=192.168.121.40, orig_p=123/udp, resp_h=212.227.54.68, resp_p=123/udp], orig=[size=48, state=1, num_pkts=0, num_bytes_ip=0,
flow_label=0, l2_addr=00:16:47:df:e7:c1], resp=[size=0, state=0, num_pkts=0, num_bytes_ip=0, flow_label=0, l2_addr=00:00:0c:9f:f0:79],
start_time=1488571365.706238, duration=0 secs, service={}, history=D, uid=CajwDY2vSUtLkztAc, tunnel=, vlan=121, inner_vlan=, dpd=, dpd_state=,
removal_hooks=, conn=, extract_orig=F, extract_resp=F, thresholds=, dce_rpc=, dce_rpc_state=, dce_rpc_backing=, dhcp=, dnp3=, dns=, dns_state=,
ftp=, ftp_data_reuse=F, ssl=, http=, http_state=, irc=, krb=, modbus=, mysql=, ntlm=, ntp=, radius=, rdp=, rfb=, sip=, sip_state=, snmp=,
smb_state=, smtp=, smtp_state=, socks=, ssh=, syslog=]
```

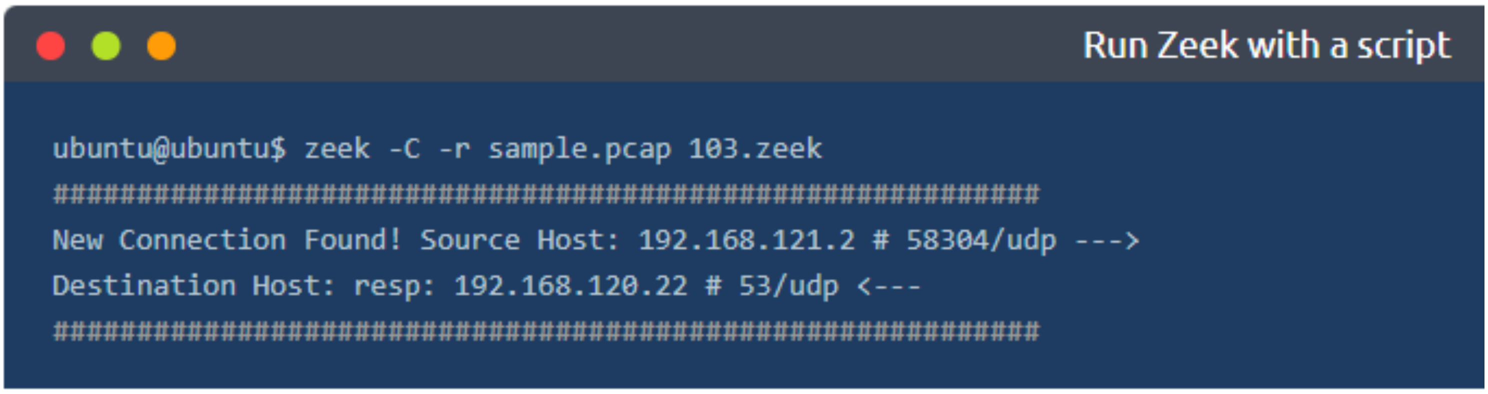
The above terminal provides bulk data for each connection. This style is not the best usage, and in real life, we will need to filter the information for specific purposes. If you look closely at the output, you can see an ID and field value for each part. To filter the event of interest, we will use the primary tag (in this case, it is c --comes from "c: connection"--), id value (id=), and field name. You should notice that the fields are the same as the fields in the log files.

▼ View Script



Sample Script

```
event new_connection(c: connection)
{
    print ("###########################################################");
    print ("");
    print ("New Connection Found!");
    print ("");
    print fmt ("Source Host: %s # %s --->", c$id$orig_h, c$id$orig_p);
    print fmt ("Destination Host: resp: %s # %s <---", c$id$resp_h, c$id$resp_p);
    print ("");
}

# %s: Identifies string output for the source.
# c$id: Source reference field for the identifier.
```



```
ubuntu@ubuntu$ zeek -C -r sample.pcap 103.zeek
###########################################################
New Connection Found! Source Host: 192.168.121.2 # 58304/udp --->
Destination Host: resp: 192.168.120.22 # 53/udp <---
###########################################################
```

The above output shows that we successfully extract specific information from the events. Remember that this script extracts the event of interest (in this example, a new connection), and we still have logs in the working directory. We can always modify and optimise the scripts at any time.

## Scripts 201 | Use Scripts and Signatures Together

Up to here, we covered the basics of Zeek scripts. Now it is time to use scripts collaboratively with other scripts and signatures to get one step closer to event correlation. Zeek scripts can refer to signatures and other Zeek scripts as well. This flexibility provides a massive advantage in event correlation.

Let's demonstrate this concept with an example. We will create a script that detects if our previously created "**ftp-admin**" rule has a hit.

▼ View Script

Sample Script

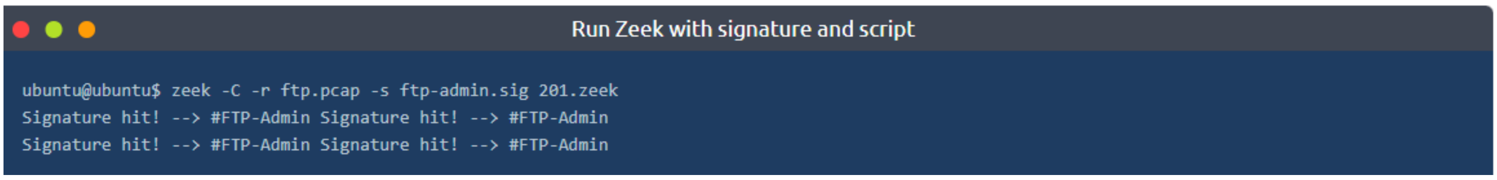
```
event signature_match (state: signature_state, msg: string, data: string)
{
if (state$sig_id == "ftp-admin")
    {
    print ("Signature hit! --> #FTP-Admin ");
    }
}
```

This basic script quickly checks if there is a signature hit and provides terminal output to notify us. We are using the "signature_match" event to accomplish this. You can read more about events here. Note that we are looking only for "ftp-admin" signature hits. The signature is shown below.

▼ View Script

Sample Script

```
signature ftp-admin {
    ip-proto == tcp
    ftp /.*USER.*admin.*/
    event "FTP Username Input Found!"
}
```


Run Zeek with signature and script

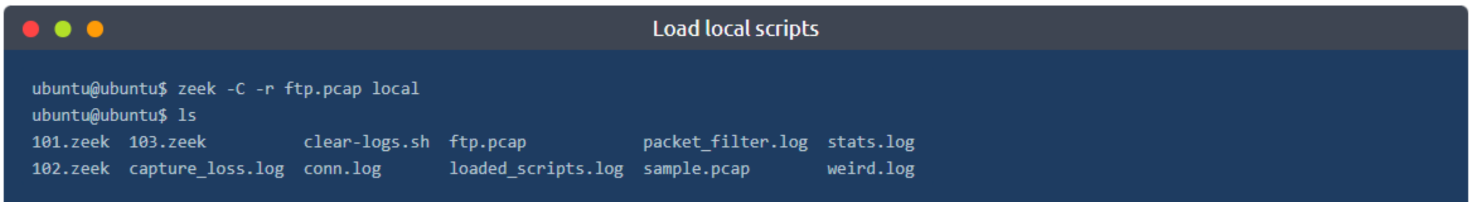
```
ubuntu@ubuntu$ zeek -C -r ftp.pcap -s ftp-admin.sig 201.zeek
Signature hit! --> #FTP-Admin Signature hit! --> #FTP-Admin
Signature hit! --> #FTP-Admin Signature hit! --> #FTP-Admin
```

The above output shows that we successfully combined the signature and script. Zeek processed the signature and logs then the script controlled the outputs and provided a terminal output for each rule hit.

## Scripts 202 | Load Local Scripts

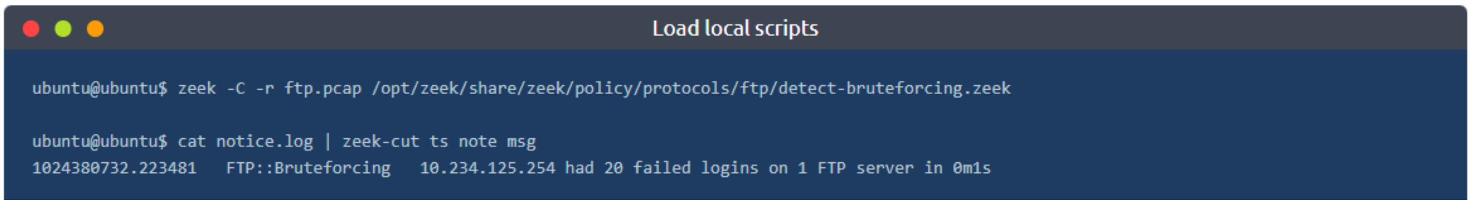
### Load all local scripts

We mentioned that Zeek has base scripts located in "/opt/zeek/share/zeek/base". You can load all local scripts identified in your "local.zeek" file. Note that base scripts cover multiple framework functionalities. You can load all base scripts by easily running the `local` command.


Load local scripts

```
ubuntu@ubuntu$ zeek -C -r ftp.pcap local
ubuntu@ubuntu$ ls
101.zeek  103.zeek          clear-logs.sh  ftp.pcap           packet_filter.log  stats.log
102.zeek  capture_loss.log  conn.log       loaded_scripts.log  sample.pcap        weird.log
```

The above output demonstrates how to run all base scripts using the "local" command. Look at the above terminal output; Zeek provided additional log files this time. Loaded scripts generated loaded_scripts.log, capture_loss.log, notice.log, stats.log files. Note that, in our instance, 465 scripts loaded and used by using the "local" command. However, Zeek doesn't provide log files for the scripts doesn't have hits or results.


Load local scripts

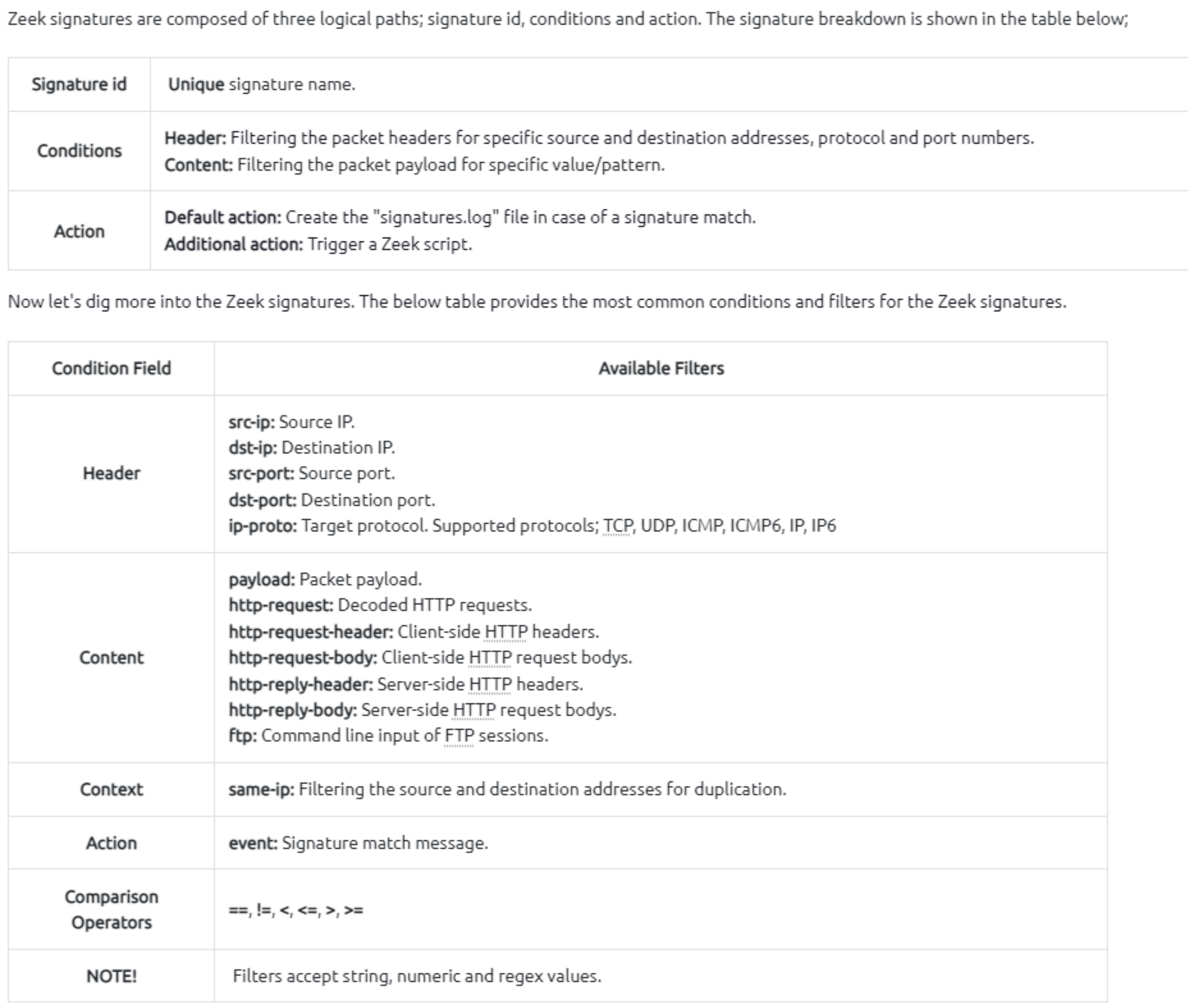
```
ubuntu@ubuntu$ zeek -C -r ftp.pcap /opt/zeek/share/zeek/policy/protocols/ftp/detect-bruteforcing.zeek

ubuntu@ubuntu$ cat notice.log | zeek-cut ts note msg
1024380732.223481    FTP::Bruteforcing    10.234.125.254 had 20 failed logins on 1 FTP server in 0m1s
```

The above output shows how to load a specific script. This script provides much more information than the one we created. It provides one single line output and a connection summary for the suspicious incident. You can find and read more on the prebuilt scripts and frameworks by visiting Zeek's online book here.

## ***Zeek Signatures***

Signatures:

Signatures need to run when zeek starts in order for zeek to look for a signature.



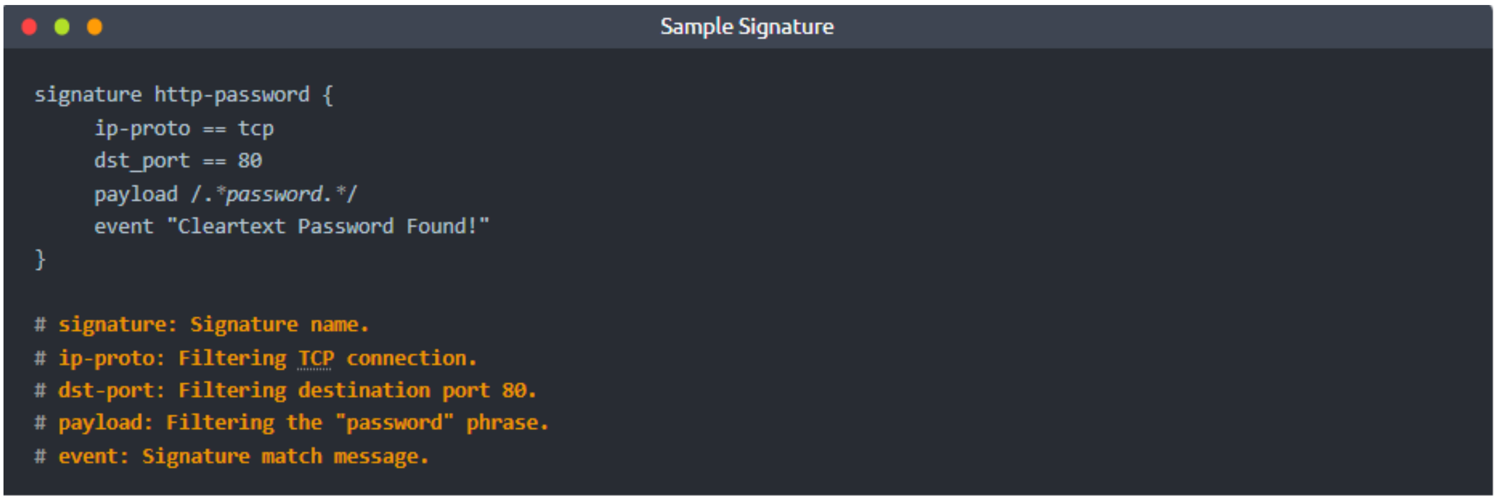
Zeek signatures are composed of three logical paths; signature id, conditions and action. The signature breakdown is shown in the table below;

| | |
|---|---|
| **Signature id** | **Unique** signature name. |
| **Conditions** | **Header:** Filtering the packet headers for specific source and destination addresses, protocol and port numbers.<br>**Content:** Filtering the packet payload for specific value/pattern. |
| **Action** | **Default action:** Create the "signatures.log" file in case of a signature match.<br>**Additional action:** Trigger a Zeek script. |

Now let's dig more into the Zeek signatures. The below table provides the most common conditions and filters for the Zeek signatures.

| Condition Field | Available Filters |
|---|---|
| **Header** | **src-ip:** Source IP.<br>**dst-ip:** Destination IP.<br>**src-port:** Source port.<br>**dst-port:** Destination port.<br>**ip-proto:** Target protocol. Supported protocols; TCP, UDP, ICMP, ICMP6, IP, IP6 |
| **Content** | **payload:** Packet payload.<br>**http-request:** Decoded HTTP requests.<br>**http-request-header:** Client-side HTTP headers.<br>**http-request-body:** Client-side HTTP request bodys.<br>**http-reply-header:** Server-side HTTP headers.<br>**http-reply-body:** Server-side HTTP request bodys.<br>**ftp:** Command line input of FTP sessions. |
| **Context** | **same-ip:** Filtering the source and destination addresses for duplication. |
| **Action** | **event:** Signature match message. |
| **Comparison Operators** | **==, !=, <, <=, >, >=** |
| **NOTE!** | Filters accept string, numeric and regex values. |

Sample Singature:



```
signature http-password {
     ip-proto == tcp
     dst_port == 80
     payload /.*password.*/
     event "Cleartext Password Found!"
}

# signature: Signature name.
# ip-proto: Filtering TCP connection.
# dst-port: Filtering destination port 80.
# payload: Filtering the "password" phrase.
# event: Signature match message.
```

## *Zeek FrameWorks*

we mentioned that Zeek highly relies on scripts, and the frameworks depend on scripts. Let's have a closer look at the file hash framework and see the script behind it.

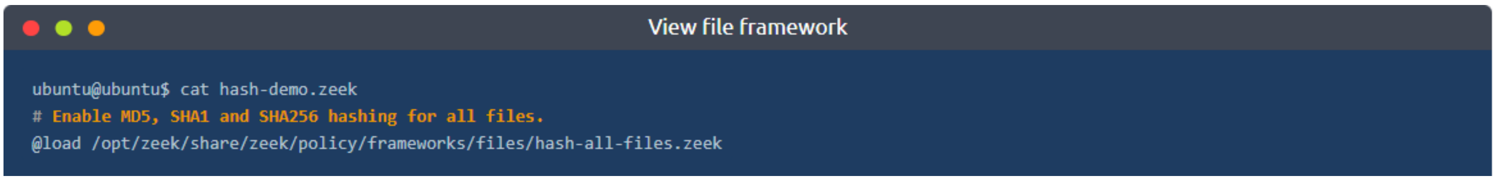
You can read more on the intelligence framework here and here.

### Scripts 203 | Load Frameworks

Zeek has 15+ frameworks that help analysts to discover the different events of interest. In this task, we will cover the common frameworks and functions. You can find and read more on the prebuilt scripts and frameworks by visiting Zeek's online book here.

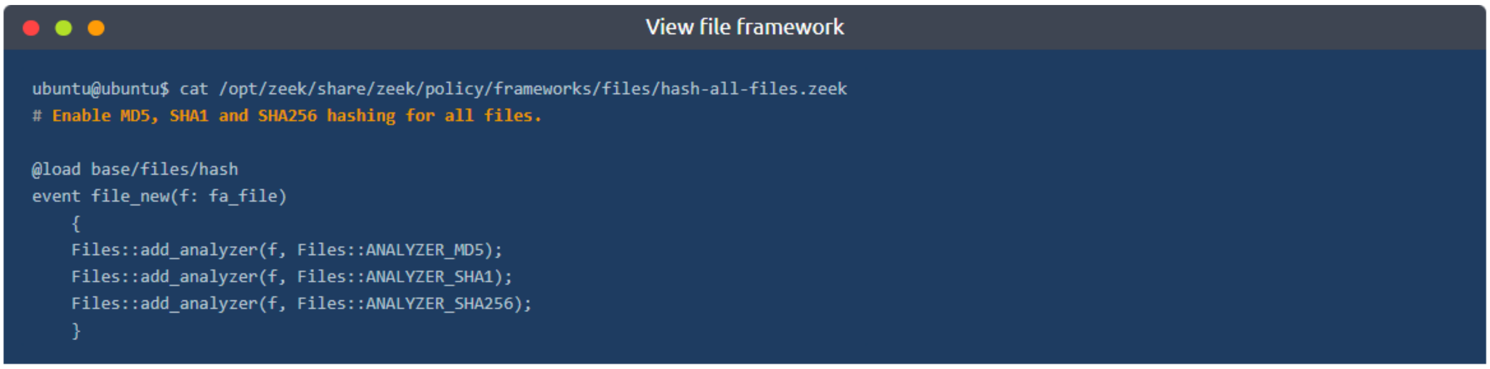
**File Framework | Hashes**

Not all framework functionalities are intended to be used in CLI mode. The majority of them are used in scripting. You can easily see the usage of frameworks in scripts by calling a specific framework as `load @ $PATH/base/frameworks/framework-name`. Now, let's use a prebuilt function of the file framework and have MD5, SHA1 and SHA256 hashes of the detected files. We will call the "File Analysis" framework's "hash-all-files" script to accomplish this. Before loading the scripts, let's look at how it works.



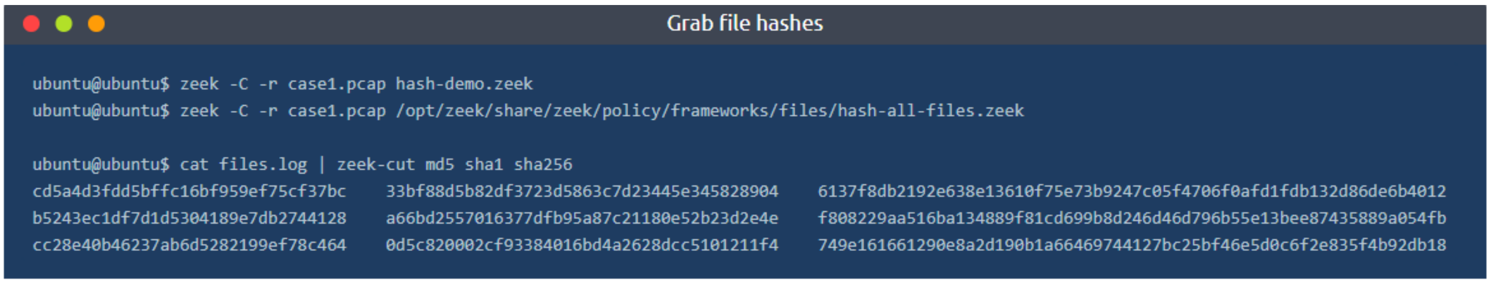
```
ubuntu@ubuntu$ cat hash-demo.zeek
# Enable MD5, SHA1 and SHA256 hashing for all files.
@load /opt/zeek/share/zeek/policy/frameworks/files/hash-all-files.zeek
```

The above output shows how frameworks are loaded. In earlier tasks, we mentioned that Zeek highly relies on scripts, and the frameworks depend on scripts. Let's have a closer look at the file hash framework and see the script behind it.



```
ubuntu@ubuntu$ cat /opt/zeek/share/zeek/policy/frameworks/files/hash-all-files.zeek
# Enable MD5, SHA1 and SHA256 hashing for all files.

@load base/files/hash
event file_new(f: fa_file)
    {
    Files::add_analyzer(f, Files::ANALYZER_MD5);
    Files::add_analyzer(f, Files::ANALYZER_SHA1);
    Files::add_analyzer(f, Files::ANALYZER_SHA256);
    }
```

Now let's execute the script and investigate the log file.

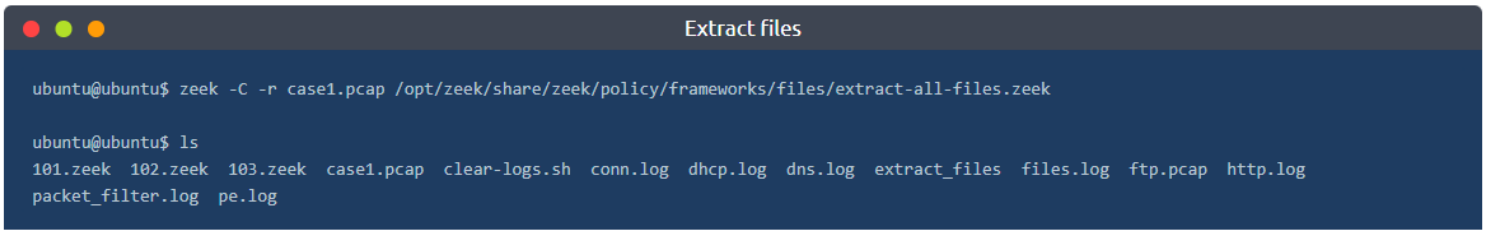


```
ubuntu@ubuntu$ zeek -C -r case1.pcap hash-demo.zeek
ubuntu@ubuntu$ zeek -C -r case1.pcap /opt/zeek/share/zeek/policy/frameworks/files/hash-all-files.zeek

ubuntu@ubuntu$ cat files.log | zeek-cut md5 sha1 sha256
cd5a4d3fdd5bffc16bf959ef75cf37bc    33bf88d5b82df3723d5863c7d23445e345828904    6137f8db2192e638e13610f75e73b9247c05f4706f0afd1fdb132d86de6b4012
b5243ec1df7d1d5304189e7db2744128    a66bd2557016377dfb95a87c21180e52b23d2e4e    f808229aa516ba134889f81cd699b8d246d46d796b55e13bee87435889a054fb
cc28e40b46237ab6d5282199ef78c464    0d5c820002cf93384016bd4a2628dcc5101211f4    749e161661290e8a2d190b1a66469744127bc25bf46e5d0c6f2e835f4b92db18
```

Look at the above terminal outputs. Both of the scripts provided the same result. Here the preference is up to the user. Both of the usage formats are true. Prebuilt frameworks are commonly used in scriptings with the "@load" method. Specific scripts are used as practical scripts for particular use cases.

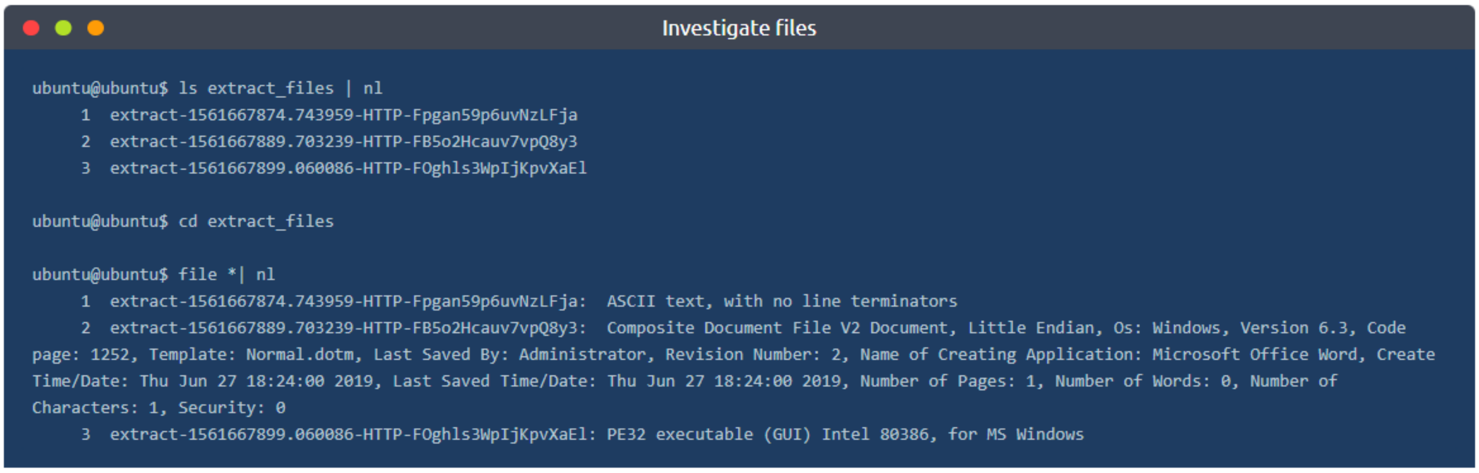
## File Framework | Extract Files

The file framework can extract the files transferred. Let's see this feature in action!



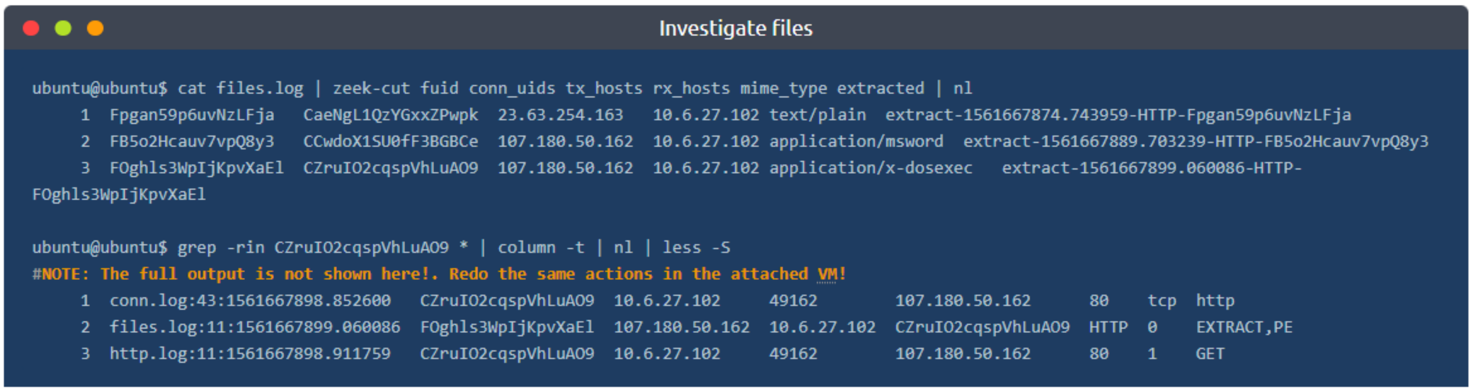
```
ubuntu@ubuntu$ zeek -C -r case1.pcap /opt/zeek/share/zeek/policy/frameworks/files/extract-all-files.zeek

ubuntu@ubuntu$ ls
101.zeek  102.zeek  103.zeek  case1.pcap  clear-logs.sh  conn.log  dhcp.log  dns.log  extract_files  files.log  ftp.pcap  http.log
packet_filter.log  pe.log
```

We successfully extracted files from the pcap. A new folder called "extract_files" is automatically created, and all detected files are located in it. First, we will list the contents of the folder, and then we will use the `file` command to determine the file type of the extracted files.


Investigate files

```
ubuntu@ubuntu$ ls extract_files | nl
     1  extract-1561667874.743959-HTTP-Fpgan59p6uvNzLFja
     2  extract-1561667889.703239-HTTP-FB5o2Hcauv7vpQ8y3
     3  extract-1561667899.060086-HTTP-FOghls3WpIjKpvXaEl

ubuntu@ubuntu$ cd extract_files

ubuntu@ubuntu$ file *| nl
     1  extract-1561667874.743959-HTTP-Fpgan59p6uvNzLFja:  ASCII text, with no line terminators
     2  extract-1561667889.703239-HTTP-FB5o2Hcauv7vpQ8y3:  Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.3, Code
page: 1252, Template: Normal.dotm, Last Saved By: Administrator, Revision Number: 2, Name of Creating Application: Microsoft Office Word, Create
Time/Date: Thu Jun 27 18:24:00 2019, Last Saved Time/Date: Thu Jun 27 18:24:00 2019, Number of Pages: 1, Number of Words: 0, Number of
Characters: 1, Security: 0
     3  extract-1561667899.060086-HTTP-FOghls3WpIjKpvXaEl: PE32 executable (GUI) Intel 80386, for MS Windows
```

Zeek extracted three files. The "file" command shows us one .txt file, one .doc/.docx file and one .exe file. Zeek renames extracted files. The name format consists of four values that come from conn.log and files.log files; default "extract" keyword, timestamp value (ts), protocol (source), and connection id (conn_uids). Let's look at the files.log to understand possible anomalies better and verify the findings. Look at the below output; files.log provides the same results with additional details. Let's focus on the .exe and correlate this finding by searching its connection id (conn_uids). The given terminal output shows us that there are three files extracted from the traffic capture. Let's look at the file.log and correlate the findings with the rest of the log files.


Investigate files

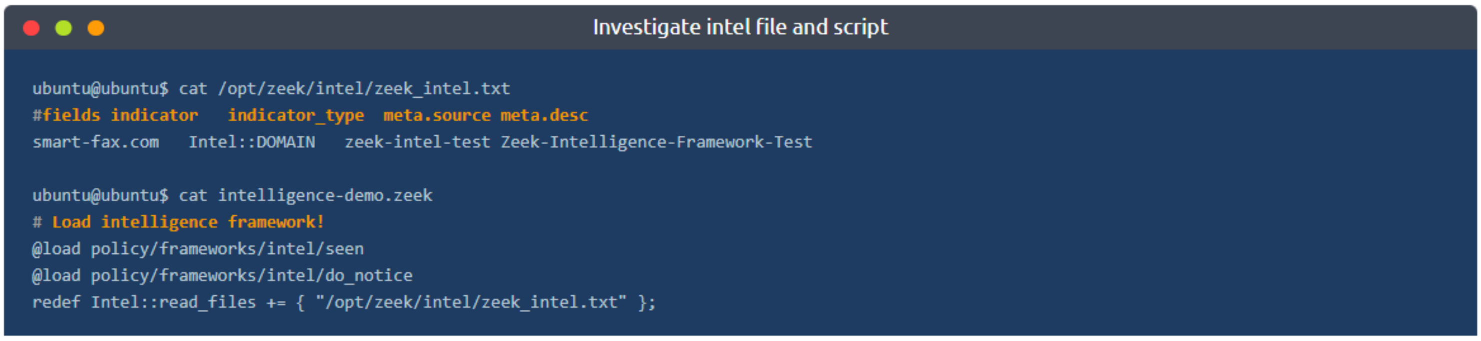
```
ubuntu@ubuntu$ cat files.log | zeek-cut fuid conn_uids tx_hosts rx_hosts mime_type extracted | nl
     1  Fpgan59p6uvNzLFja   CaeNgL1QzYGxxZPwpk  23.63.254.163   10.6.27.102 text/plain   extract-1561667874.743959-HTTP-Fpgan59p6uvNzLFja
     2  FB5o2Hcauv7vpQ8y3   CCwdoX1SU0fF3BGBCe  107.180.50.162  10.6.27.102 application/msword  extract-1561667889.703239-HTTP-FB5o2Hcauv7vpQ8y3
     3  FOghls3WpIjKpvXaEl  CZruIO2cqspVhLuAO9  107.180.50.162  10.6.27.102 application/x-dosexec    extract-1561667899.060086-HTTP-
FOghls3WpIjKpvXaEl

ubuntu@ubuntu$ grep -rin CZruIO2cqspVhLuAO9 * | column -t | nl | less -S
#NOTE: The full output is not shown here!. Redo the same actions in the attached VM!
     1  conn.log:43:1561667898.852600   CZruIO2cqspVhLuAO9  10.6.27.102     49162           107.180.50.162      80    tcp   http
     2  files.log:11:1561667899.060086  FOghls3WpIjKpvXaEl  107.180.50.162  10.6.27.102     CZruIO2cqspVhLuAO9  HTTP  0     EXTRACT,PE
     3  http.log:11:1561667898.911759   CZruIO2cqspVhLuAO9  10.6.27.102     49162           107.180.50.162      80    1     GET
```

The "grep" tool helps us investigate the particular value across all available logs. The above terminal output shows us that the connection id linked with .exe appears in conn.log, files.log, and http.log files. Given example demonstrates how to filter some fields and correlate the findings with the rest of the logs. We've listed the source and destination addresses, file and connection id numbers, MIME types, and file names. Up to now, provided outputs and findings show us that record number three is a .exe file, and other log files provide additional information.

## Notice Framework | Intelligence

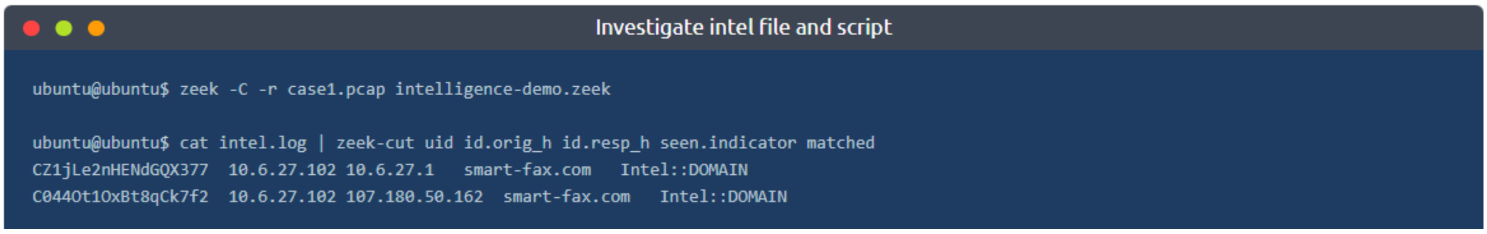
The intelligence framework can work with data feeds to process and correlate events and identify anomalies. The intelligence framework requires a feed to match and

create alerts from the network traffic. Let's demonstrate a single user-generated threat intel file and let Zeek use it as the primary intelligence source.
Intelligence source location: `/opt/zeek/intel/zeek_intel.txt`
There are two critical points you should never forget. First, the source file has to be tab-delimited. Second, you can manually update the source and adding extra lines doesn't require any re-deployment. However, if you delete a line from the file, you will need to re-deploy the Zeek instance.
Let's add the suspicious URL gathered from the case1.pcap file as a source intel and see this feature in action! Before executing the script, let's look at the intelligence file and the script contents.


Investigate intel file and script

```
ubuntu@ubuntu$ cat /opt/zeek/intel/zeek_intel.txt
#fields indicator   indicator_type  meta.source meta.desc
smart-fax.com   Intel::DOMAIN   zeek-intel-test Zeek-Intelligence-Framework-Test

ubuntu@ubuntu$ cat intelligence-demo.zeek
# Load intelligence framework!
@load policy/frameworks/intel/seen
@load policy/frameworks/intel/do_notice
redef Intel::read_files += { "/opt/zeek/intel/zeek_intel.txt" };
```

The above output shows the contents of the intel file and script contents. There is one intelligence input, and it is focused on a domain name, so when this domain name appears in the network traffic, Zeek will create the "intel.log" file and provide the available details.


Investigate intel file and script

```
ubuntu@ubuntu$ zeek -C -r case1.pcap intelligence-demo.zeek

ubuntu@ubuntu$ cat intel.log | zeek-cut uid id.orig_h id.resp_h seen.indicator matched
CZ1jLe2nHENdGQX377  10.6.27.102 10.6.27.1   smart-fax.com   Intel::DOMAIN
C044Ot1OxBt8qCk7f2  10.6.27.102 107.180.50.162  smart-fax.com   Intel::DOMAIN
```

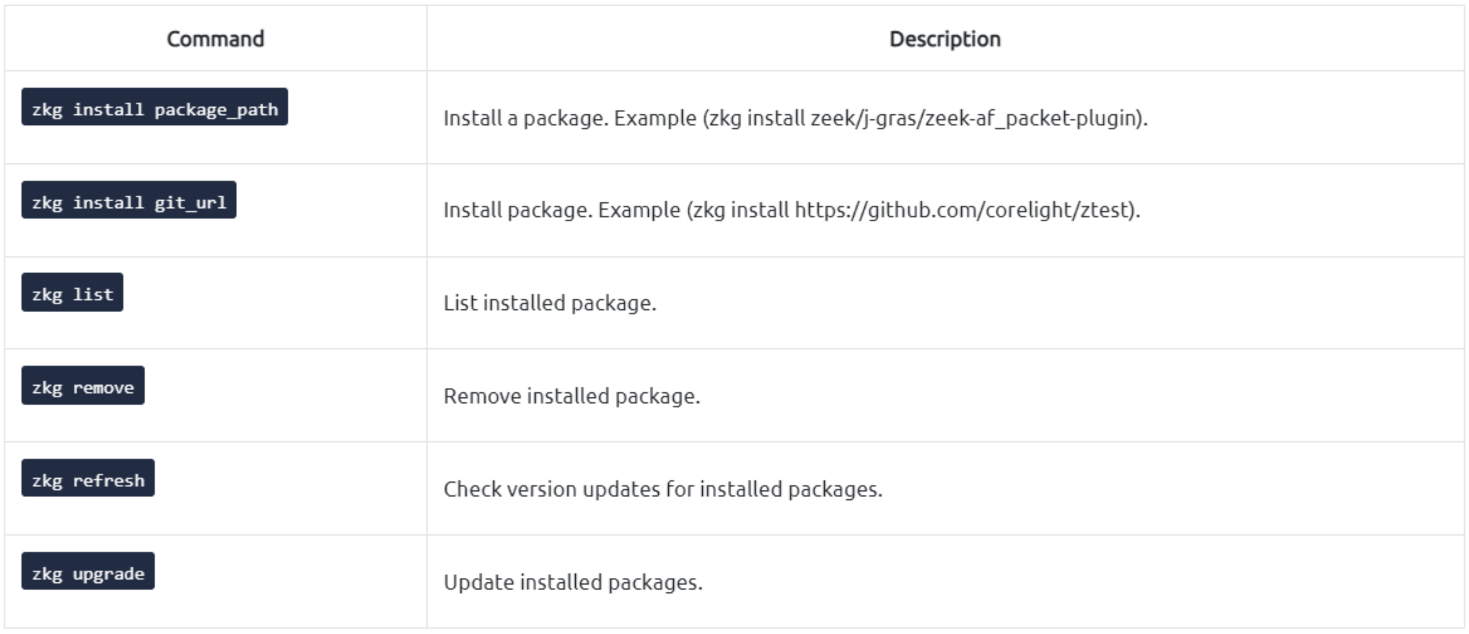
The above output shows that Zeek detected the listed domain and created the intel.log file. This is one of the easiest ways of using the intelligence framework.

## ***Zeek Packacges***

### **Scripts 204 | Package Manager**

Zeek Package Manager helps users install third-party scripts and plugins to extend Zeek functionalities with ease. The package manager is installed with Zeek and available with the `zkg` command. Users can install, load, remove, update and create packages with the "zkg" tool. You can read more on and view available packages here and here. Please note that you need root privileges to use the "zkg" tool.

Basic usage of zkg;

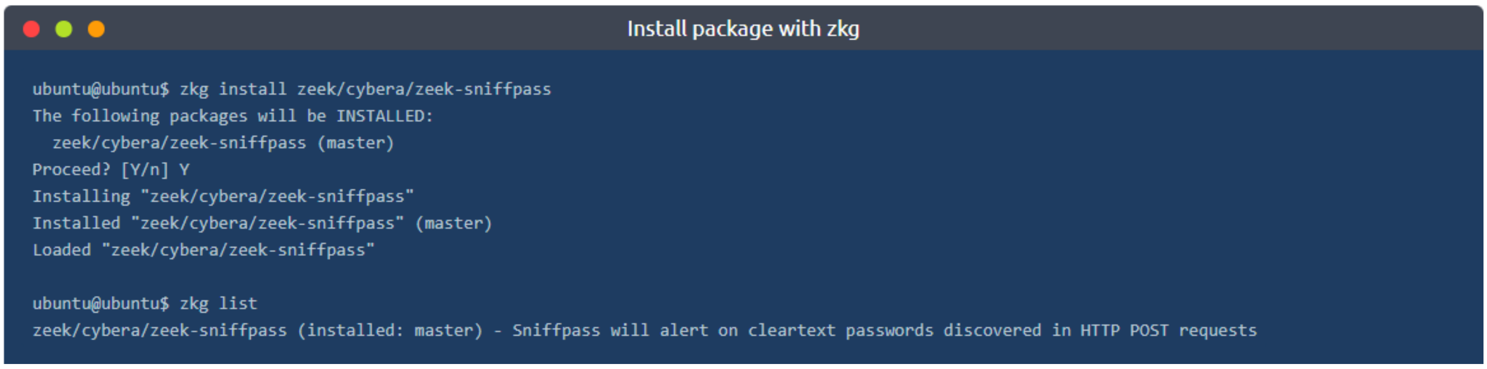


| Command | Description |
|---|---|
| `zkg install package_path` | Install a package. Example (zkg install zeek/j-gras/zeek-af_packet-plugin). |
| `zkg install git_url` | Install package. Example (zkg install https://github.com/corelight/ztest). |
| `zkg list` | List installed package. |
| `zkg remove` | Remove installed package. |
| `zkg refresh` | Check version updates for installed packages. |
| `zkg upgrade` | Update installed packages. |

There are multiple ways of using packages. The first approach is using them as frameworks and calling specific package path/directory per usage. The second and most common approach is calling packages from a script with the "@load" method. The third and final approach to using packages is calling their package names; note that this method works only for packages installed with the "zkg" install method.

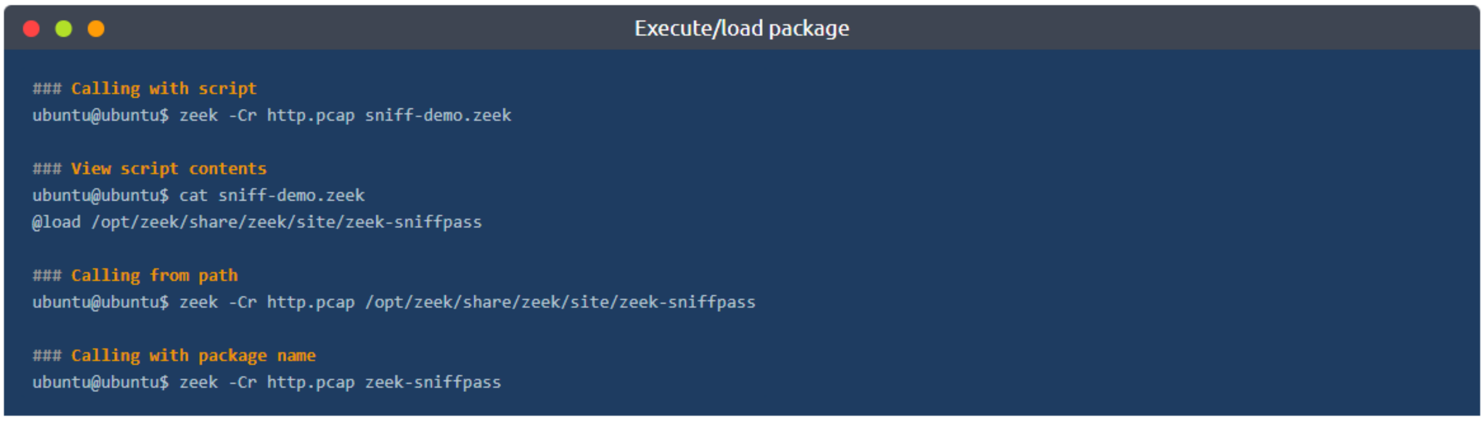
## Packages | Cleartext Submission of Password

Let's install a package first and then demonstrate the usage in different approaches.
**Note:** The package is installed in the given VM.



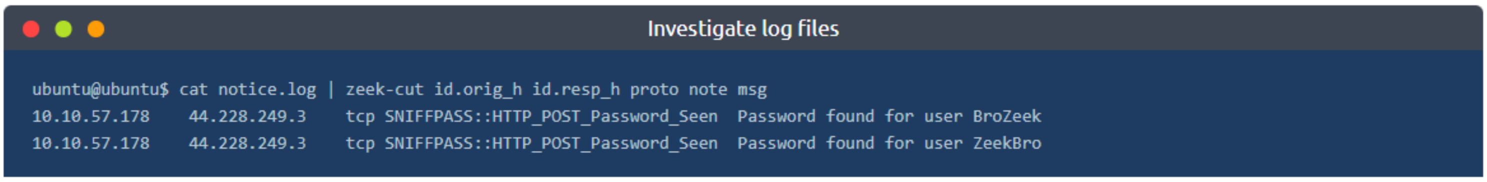
```
ubuntu@ubuntu$ zkg install zeek/cybera/zeek-sniffpass
The following packages will be INSTALLED:
  zeek/cybera/zeek-sniffpass (master)
Proceed? [Y/n] Y
Installing "zeek/cybera/zeek-sniffpass"
Installed "zeek/cybera/zeek-sniffpass" (master)
Loaded "zeek/cybera/zeek-sniffpass"

ubuntu@ubuntu$ zkg list
zeek/cybera/zeek-sniffpass (installed: master) - Sniffpass will alert on cleartext passwords discovered in HTTP POST requests
```

The above output shows how to install and list the installed packages. Now we successfully installed a package. As the description mentions on the above terminal, this package creates alerts for cleartext passwords found in HTTP traffic. Let's use this package in three different ways!



```
### Calling with script
ubuntu@ubuntu$ zeek -Cr http.pcap sniff-demo.zeek

### View script contents
ubuntu@ubuntu$ cat sniff-demo.zeek
@load /opt/zeek/share/zeek/site/zeek-sniffpass

### Calling from path
ubuntu@ubuntu$ zeek -Cr http.pcap /opt/zeek/share/zeek/site/zeek-sniffpass

### Calling with package name
ubuntu@ubuntu$ zeek -Cr http.pcap zeek-sniffpass
```

The above output demonstrates how to execute/load packages against a pcap. You can use the best one for your case. The "zeek-sniffpass" package provides additional information in the notice.log file. Now let's review the logs and discover the obtained data using the specific package.

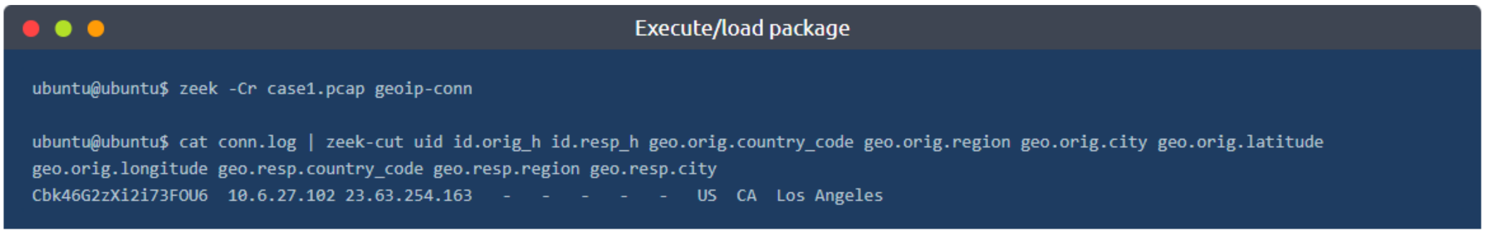


```
ubuntu@ubuntu$ cat notice.log | zeek-cut id.orig_h id.resp_h proto note msg
10.10.57.178    44.228.249.3    tcp SNIFFPASS::HTTP_POST_Password_Seen  Password found for user BroZeek
10.10.57.178    44.228.249.3    tcp SNIFFPASS::HTTP_POST_Password_Seen  Password found for user ZeekBro
```

The above output shows that the package found cleartext password submissions, provided notice, and grabbed the usernames. Remember, in **TASK-5** we created a signature to do the same action. Now we can do the same activity without using a signature file. This is a simple demonstration of the benefit and flexibility of the Zeek scripts.

## Packages | Geolocation Data

Let's use another helpful package called "geoip-conn". This package provides geolocation information for the IP addresses in the conn.log file. It depends on "GeoLite2-City.mmdb" database created by MaxMind. This package provides location information for only matched IP addresses from the internal database.



```
ubuntu@ubuntu$ zeek -Cr case1.pcap geoip-conn

ubuntu@ubuntu$ cat conn.log | zeek-cut uid id.orig_h id.resp_h geo.orig.country_code geo.orig.region geo.orig.city geo.orig.latitude
geo.orig.longitude geo.resp.country_code geo.resp.region geo.resp.city
Cbk46G2zXi2i73FOU6  10.6.27.102 23.63.254.163   -   -   -   -   -   US  CA  Los Angeles
```

Up to now, we've covered what the Zeek packages are and how to use them. There are much more packages and scripts available for Zeek in the wild. You can try ready or third party packages and scripts or learn Zeek scripting language and create new ones.