Bro Logging

Contents

Once Bro has been deployed in an environment and monitoring live traffic, it will, in its default configuration, begin to produce human-readable ASCII logs. Each log file, produced by Bro's Logging Framework, is populated with organized, mostly connection-oriented data. As the standard log files are simple ASCII data, working with the data contained in them can be done from a command line terminal once you have been familiarized with the types of data that can be found in each file. In the following, we work through the logs general structure and then examine some standard ways of working with them.

Working with Log Files

Generally, all of Bro's log files are produced by a corresponding script that defines their individual structure. However, as each log file flows through the Logging Framework, they share a set of structural similarities. Without breaking into the scripting aspect of Bro here, a bird's eye view of how the log files are produced progresses as follows. The script's author defines the kinds of data, such as the originating IP address or the duration of a connection, which will make up the fields (i.e., columns) of the log file. The author then decides what network activity should generate a single log file entry (i.e., one line). For example, this could be a connection having been completed or an HTTP `GET` request being issued by an originator. When these behaviors are observed during operation, the data is passed to the Logging Framework which adds the entry to the appropriate log file.

As the fields of the log entries can be further customized by the user, the Logging Framework makes use of a header block to ensure that it remains self-describing. This header entry can be see by running the Unix utility `head` and outputting the first lines of the file:

```
# bro -r wikipedia.trace
```

```
#separator \x09
#set_separator    ,
#empty_field      (empty)
#unset_field      -
#path     conn
#open     2018-05-23-00-22-38
#fields   ts      uid     id.orig_h       id.orig_p       id.resp_h
#types    time    string  addr    port    addr    port    enum    strin
1300475167.096535 CHhAvVGS1DHFjwGM9       141.142.220.202 5353    224.0
1300475167.097012 ClEkJM2Vm5giqnMf4h      fe80::217:f2ff:fed7:cf65
```


SEARCH

Search

```
1300475167.099816 C4J4Th3PJpwUYZZ6gc      141.142.220.50  5353    224.0
1300475168.853899 CmES5u32sYpV7JYN        141.142.220.118 43927   141.1
1300475168.854378 CP5puj4I8PtEU4qzYg      141.142.220.118 37676   141.1
1300475168.854837 C37jN32gN3y3AZzyf6      141.142.220.118 40526   141.1
1300475168.857956 C0LAHyvtKSQHyJxIl       141.142.220.118 32902   141.1
[...]
```

As you can see, the header consists of lines prefixed by `#` and includes information such as what separators are being used for various types of data, what an empty field looks like and what an unset field looks like. In this example, the default TAB separator is being used as the delimiter between fields (`\x09` is the tab character in hex). It also lists the comma as the separator for set data, the string `(empty)` as the indicator for an empty field and the `-` character as the indicator for a field that hasn't been set. The timestamp for when the file was created is included under `#open`. The header then goes on to detail the fields being listed in the file and the data types of those fields, in `#fields` and `#types`, respectively. These two entries are often the two most significant points of interest as they detail not only the field names but the data types used. When navigating through the different log files with tools like `sed`, `awk`, or `grep`, having the field definitions readily available saves the user some mental leg work. The field names are also a key resource for using the bro-cut utility included with Bro, see below.

Next to the header follows the main content. In this example we see 7 connections with their key properties, such as originator and responder IP addresses (note how Bro transparently handles both IPv4 and IPv6), transport-layer ports, application-layer services (- the `service` field is filled in as Bro determines a specific protocol to be in use, independent of the connection's ports), payload size, and more. See `Conn::Info` for a description of all fields.

In addition to `conn.log`, Bro generates many further logs by default, including:

`dpd.log`
A summary of protocols encountered on non-standard ports.

`dns.log`
All DNS activity.

`ftp.log`
A log of FTP session-level activity.

`files.log`
Summaries of files transferred over the network. This information is aggregated from different protocols, including HTTP, FTP, and SMTP.

`http.log`
A summary of all HTTP requests with their replies.

`known_certs.log`
SSL certificates seen in use.

`smtp.log`
A summary of SMTP activity.

`ssl.log`
A record of SSL sessions, including certificates being used.

`weird.log`

A log of unexpected protocol-level activity. Whenever Bro's protocol analysis encounters a situation it would not expect (e.g., an RFC violation) it logs it in this file. Note that in practice, real-world networks tend to exhibit a large number of such "crud" that is usually not worth following up on.

As you can see, some log files are specific to a particular protocol, while others aggregate information across different types of activity. For a complete list of log files and a description of its purpose, see Log Files.

Using `bro-cut`

The `bro-cut` utility can be used in place of other tools to build terminal commands that remain flexible and accurate independent of possible changes to the log file itself. It accomplishes this by parsing the header in each file and allowing the user to refer to the specific columnar data available (in contrast to tools like `awk` that require the user to refer to fields referenced by their position). For example, the following command extracts just the given columns from a `conn.log` :

```
# cat conn.log | bro-cut id.orig_h id.orig_p id.resp_h duration
141.142.220.202   5353    224.0.0.251     -
fe80::217:f2ff:fed7:cf65  5353    ff02::fb        -
141.142.220.50    5353    224.0.0.251     -
141.142.220.118   43927   141.142.2.2     0.000435
141.142.220.118   37676   141.142.2.2     0.000420
141.142.220.118   40526   141.142.2.2     0.000392
141.142.220.118   32902   141.142.2.2     0.000317
141.142.220.118   59816   141.142.2.2     0.000343
141.142.220.118   59714   141.142.2.2     0.000375
141.142.220.118   58206   141.142.2.2     0.000339
[...]
```

The corresponding `awk` command will look like this:

```
# awk '/^[^#]/ {print $3, $4, $5, $6, $9}' conn.log
141.142.220.202 5353 224.0.0.251 5353 -
fe80::217:f2ff:fed7:cf65 5353 ff02::fb 5353 -
141.142.220.50 5353 224.0.0.251 5353 -
141.142.220.118 43927 141.142.2.2 53 0.000435
141.142.220.118 37676 141.142.2.2 53 0.000420
141.142.220.118 40526 141.142.2.2 53 0.000392
141.142.220.118 32902 141.142.2.2 53 0.000317
141.142.220.118 59816 141.142.2.2 53 0.000343
141.142.220.118 59714 141.142.2.2 53 0.000375
141.142.220.118 58206 141.142.2.2 53 0.000339
[...]
```

While the output is similar, the advantages to using bro-cut over `awk` lay in that, while `awk` is flexible and powerful, `bro-cut` was specifically designed to work with Bro's log files. Firstly, the `bro-cut` output includes only the log file entries, while the `awk` solution needs to skip the header manually. Secondly, since `bro-cut` uses the field descriptors to identify and extract data, it allows for flexibility independent of the format and contents of the log file. It's not uncommon for a Bro configuration to add extra fields to various log files as required by the

environment. In this case, the fields in the `awk` command would have to be altered to compensate for the new position whereas the `bro-cut` output would not change.

> **Note**
>
> The sequence of field names given to `bro-cut` determines the output order, which means you can also use `bro-cut` to reorder fields. That can be helpful when piping into, e.g., `sort`.

As you may have noticed, the command for `bro-cut` uses the output redirection through the `cat` command and `|` operator. Whereas tools like `awk` allow you to indicate the log file as a command line option, bro-cut only takes input through redirection such as `|` and `<`. There are a couple of ways to direct log file data into `bro-cut`, each dependent upon the type of log file you're processing. A caveat of its use, however, is that all of the header lines must be present.

> **Note**
>
> `bro-cut` provides an option `-c` to include a corresponding format header into the output, which allows to chain multiple `bro-cut` instances or perform further post-processing that evaluates the header information.

In its default setup, Bro will rotate log files on an hourly basis, moving the current log file into a directory with format `YYYY-MM-DD` and gzip compressing the file with a file format that includes the log file type and time range of the file. In the case of processing a compressed log file you simply adjust your command line tools to use the complementary `z*` versions of commands such as `cat` (`zcat`) or `grep` (`zgrep`).

Working with Timestamps

`bro-cut` accepts the flag `-d` to convert the epoch time values in the log files to human-readable format. The following command includes the human readable time stamp, the unique identifier, the HTTP `Host`, and HTTP `URI` as extracted from the `http.log` file:

```
# bro-cut -d ts uid host uri < http.log
2011-03-18T19:06:08+0000  CUM0KZ3MLUfNB0cl11      bits.wikimedia.org
2011-03-18T19:06:08+0000  CwjjYJ2WqgTbAqiHl6      upload.wikimedia.org
2011-03-18T19:06:08+0000  C3eiCBGOLw3VtHfOj       upload.wikimedia.org
2011-03-18T19:06:08+0000  Ck51lg1bScffFj34Ri      upload.wikimedia.org
2011-03-18T19:06:08+0000  CtxTCR2Yer0FR1tIBg      upload.wikimedia.org
[...]
```

Often times log files from multiple sources are stored in UTC time to allow easy correlation. Converting the timestamp from a log file to UTC can be accomplished with the `-u` option:

```
# bro-cut -u ts uid host uri < http.log
2011-03-18T19:06:08+0000  CUM0KZ3MLUfNB0cl11      bits.wikimedia.org
2011-03-18T19:06:08+0000  CwjjYJ2WqgTbAqiHl6      upload.wikimedia.org
2011-03-18T19:06:08+0000  C3eiCBGOLw3VtHfOj       upload.wikimedia.org
2011-03-18T19:06:08+0000  Ck51lg1bScffFj34Ri      upload.wikimedia.org
2011-03-18T19:06:08+0000  CtxTCR2Yer0FR1tIBg      upload.wikimedia.org
[...]
```

The default time format when using the `-d` or `-u` is the `strftime` format string `%Y-%m-%dT%H:%M:%S%z` which results in a string with year, month, day of month, followed by hour, minutes, seconds and the timezone offset. The default format can be altered by using the `-D` and `-U` flags, using the standard `strftime` syntax. For example, to format the timestamp in the US-typical "Middle Endian" you could use a format string of: `%d-%m-%YT%H:%M:%S%z`

```
# bro-cut -D %d-%m-%YT%H:%M:%S%z ts uid host uri < http.log
18-03-2011T19:06:08+0000  CUM0KZ3MLUfNB0cl11      bits.wikimedia.org
18-03-2011T19:06:08+0000  CwjjYJ2WqgTbAqiHl6      upload.wikimedia.org
18-03-2011T19:06:08+0000  C3eiCBGOLw3VtHfOj       upload.wikimedia.org
18-03-2011T19:06:08+0000  Ck51lg1bScffFj34Ri      upload.wikimedia.org
18-03-2011T19:06:08+0000  CtxTCR2Yer0FR1tIBg      upload.wikimedia.org
[...]
```

See `man strfime` for more options for the format string.

Using UIDs

While Bro can do signature-based analysis, its primary focus is on behavioral detection which alters the practice of log review from "reactionary review" to a process a little more akin to a hunting trip. A common progression of review includes correlating a session across multiple log files. As a connection is processed by Bro, a unique identifier is assigned to each session. This unique identifier is generally included in any log file entry associated with that connection and can be used to cross-reference different log files.

A simple example would be to cross-reference a UID seen in a `conn.log` file. Here, we're looking for the connection with the largest number of bytes from the responder by redirecting the output for `cat conn.log` into bro-cut to extract the UID and the resp_bytes, then sorting that output by the resp_bytes field.

```
# cat conn.log | bro-cut uid resp_bytes | sort -nrk2 | head -5
CwjjYJ2WqgTbAqiHl6        734
CtxTCR2Yer0FR1tIBg        734
Ck51lg1bScffFj34Ri        734
CLNN1k2QMum1aexUK7        734
CykQaM33ztNt0csB9a        733
```

Taking the UID of the first of the top responses, we can now crossreference that with the UIDs in the `http.log` file.

```
# cat http.log | bro-cut uid id.resp_h method status_code host uri | gr
CUM0KZ3MLUfNB0cl11        208.80.152.118  GET     304     bits.wikimedi
```

As you can see there are two HTTP `GET` requests within the session that Bro identified and logged. Given that HTTP is a stream protocol, it can have multiple `GET` / `POST` /etc requests in a stream and Bro is able to extract and track that information for you, giving you an in-depth and structured view into HTTP traffic on your network.

Internal Pages