An Introduction to Monitoring Encrypted Network Traffic with Joy

Philip Perricone, Bill Hudson, Blake Anderson, David McGrew
Cisco Live Barcelona 2018 Workbench

Abstract: *TLS encryption has become the standard form of Internet communication. In this session, we will demonstrate our open source project: Joy. It extends flow monitoring technologies by collecting much more detailed information about flows. This information can be used in conjunction with simple rules to detect obsolete cryptography on a network, or can be used by machine learning algorithms to detect malicious, encrypted flows. Both of these use cases will be highlighted.*

Technical Level: Introductory
Technology: Open Source, Security
Solutions: Analytics, Threat Defense
Session Type: DevNet
Session Length: 45 min

Obtaining Joy

Wait! These steps have already been completed for you!

git clone https://github.com/cisco/joy
./config
make

```
$ sudo apt-get install build-essential libssl-dev libpcap-dev \
libcurl4-openssl-dev

$ sudo apt-get install python-dev python-numpy python-setuptools \
python-scipy

$ sudo easy_install -U scikit-learn
```

Software Components

joy
sleuth
model.py

Walkthrough

Using Joy to process PCAP into flow JSON

```
$ ./bin/joy bidir=1 http=1 dns=1 tls=1 dist=1 \
../benign/capture.pcap > capture.gz

$ zless capture.gz
```

The first line is metadata that describes the program options used. The second line is a JSON description of a flow:

```
 {
       "sa":"10.0.2.15",                    # source address
       "da":"64.102.6.247",                 # destination address
       "pr":17,                             # protocol (UDP)
       "sp":55635,                          # source port
       "dp":53,                             # destination port (DNS)
       "bytes_out":52,                      # number outbound bytes
       "num_pkts_out":2,                    # number outbound packets
       "bytes_in":531,                      # number inbound bytes
       "num_pkts_in":2,                     # number inbound packets
       "time_start":1465338193.717080,
       "time_end":1465338193.762416,
       "packets":[
                   {"b":26,"dir":">","ipt":0},
                   {"b":26,"dir":">","ipt":0},
                   {"b":444,"dir":"<","ipt":32},

                                   ...
```

Using Sleuth

We can view all of the information with indentation by using the --pretty option:

```
$ ./sleuth capture.gz --pretty | less
```

The --select option will select certain keys to be printed.
(Example) Print the 5-tuple identifier of each flow:

```
$ ./sleuth capture.gz --select "sa, da, sp, dp, pr" | less
```

The --where option will filter the flows to meet specific conditions.
(Example) Print each flow where the protocol equals 17:

```
$ ./sleuth capture.gz --select "sa, da, pr" --where "pr=17" | less
```

The --split option will split the output into multiple streams according to the field elements specified.
(Example) Create a stream for each distinct (destination address, destination port) tuple:

```
$ ./sleuth capture.gz --select "da, dp" --groupby "da, dp" | less
```

The –sum option will compute the sum for the values of each field element specified.
(Example) Find which destinations had the most data sent to them (outbound bytes):

```
$ ./sleuth capture.gz --select "da, bytes_out" --groupby "da" \
--sum "bytes_out" | sort -k 2 -nr | less
```

The --dist option will calculate the distribution of each unique value for the selected field elements.
(Example) Show the distribution of TLS selected cipher suites:

```
$ ./sleuth capture.gz --pretty --select "tls{scs}" --dist | less
```

More Examples

Looking at TLS:

```
$ ./sleuth capture.gz --pretty --where "dp=443 "| less
```

TLS security levels:

```
$ ./sleuth capture.gz --tls_sec --select "sa, da, tls_sec" \
--where "dp=443" | less
```

TLS server certificate chains where one or more of the keys is less than 2048 bits:

```
$ ./sleuth capture.gz --pretty --select \
"tls{s_cert[{signature_key_size,signature_algo}]},da,linked_dns" \
--where "tls{s_cert[{signature_key_size}]}<2048" | less
```

Fingerprinting TLS, HTTP, and TCP:

```
$ ./sleuth capture.gz --pretty --fingerprint "tls,http,tcp" \
--select "sa,da,fingerprint" | less
```

TLS library inferences using fingerprints:

```
$ ./sleuth capture.gz --pretty --fingerprint "tls" --select \
"sa,da,inferences" --where "dp=443" | less
```

SSH brute-force attack:

```
$ ./sleuth /home/joy-user/malware/kali-password-attack_hydra-eof.pcap \
--select "sa, da, dp, bytes_out" --where "dp=22" --sum bytes_out | less
```

Comparing the “sum_over” and “bytes_out” value to the normal ssh usage example, we see much higher values:

```
$ ./sleuth /home/joy-user/joy/test/pcaps/kali-normal-ssh.pcap \
--select "sa, da, dp, bytes_out" --where "dp=22" --sum bytes_out | less
```

Identify malware flows based on SPLT and BD

The built-in classifier can detect malware based on its Sequence of Packet Lengths and Times (SPLT) behavior and it's Byte Distribution:

```
$ ./bin/joy bidir=1 dist=1 classify=1 ../benign/capture.pcap > capture.gz

$ ./sleuth capture.gz --select "da, dp, p_malware" \
--where "p_malware > 0.01"

{"p_malware": 0.084199, "dp": 80, "da": "151.101.1.69"}
{"p_malware": 0.020982, "dp": 443, "da": "172.217.1.195"}
```

Example showing a similar run on a malicious PCAP:

```
$ ./bin/joy bidir=1 dist=1 classify=1 \
../malware/133d773a8ca64c24bd81b594cf5240dd.pcap > malware.gz

$ ./sleuth malware.gz --select "da, dp, p_malware" \
--where "p_malware > 0.99"

{"p_malware": 0.996547, "dp": 443, "da": "86.59.21.38"}
{"p_malware": 0.998782, "dp": 1900, "da": "239.255.255.250"}
{"p_malware": 0.995214, "dp": 443, "da": "108.61.179.216"}
{"p_malware": 0.998643, "dp": 9001, "da": "5.9.123.81"}
{"p_malware": 0.991867, "dp": 443, "da": "109.238.6.25"}
                        ...
```

Making a classifier

We can use model.py to train logistic regression classifiers from two data directories; one containing malicious output and one containing benign output. The arguments to model.py are:

```
  -p POS_DIR, --pos_dir POS_DIR
                        Directory of Positive Examples (JSON Format)
  -n NEG_DIR, --neg_dir NEG_DIR
                        Directory of Negative Examples (JSON Format)
  -m, --meta            Parse Metadata Information
  -l, --lengths         Parse Packet Size Information
  -t, --times           Parse Inter-packet Time Information
  -d, --dist            Parse Byte Distribution Information
  -o OUTPUT, --output OUTPUT
                        Output file for parameters
```

Joy uses two classifiers, one with only metadata and packet lengths/times, and one that also contains the byte distribution. To generate the parameter file that does not use the byte distribution, we run:

```
$ ./bin/joy bidir=1 dist=1 ../malware/*.pcap > ../malware_train/malware.gz

$ ./bin/joy bidir=1 dist=1 ../benign/*.pcap > ../benign_train/benign.gz

$ python analysis/model.py -m -l -t -p ../malware_train/ \
-n ../benign_train/ -o params.txt

Num Positive:     2217
Num Negative:     1499

Features Used:
      Metadata          (7)
      Packet Lengths          (100)
      Packet Times            (100)
Total Features:   207

non-zero parameters:    47
```

Next we generate the parameters that use the byte distribution:

```
$ python analysis/model.py -m -l -t -d -p ../malware_train/ \
-n ../benign_train/ -o params_bd.txt

Num Positive:     2217
Num Negative:     1499

Features Used:
      Metadata          (7)
      Packet Lengths          (100)
      Packet Times            (100)
      Byte Distribution (256)
Total Features:   463

non-zero parameters:    49
```

Using the new classifier

We can use the newly made classifiers that we trained in the previous step. To do so, we need tell Joy to use the new classifier parameters that were generated by providing the “model” option as an argument to the program like this:

```
$ ./bin/joy bidir=1 dist=1 classify=1 model="params.txt:params_bd.txt" \
../everything > new_class.gz
```

Finally, as we learned in many of the prior examples, we can use sleuth to examine the output:

```
$ ./sleuth new_class.gz --select "da, dp, p_malware" \
--where "p_malware > 0.99"
```

END