An Introduction to Monitoring Encrypted Network Traffic with Joy

Philip Perricone, Bill Hudson, Blake Anderson, David McGrew
Cisco Live Berlin 2017 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

```
cl-berlin@Joy$ sudo apt-get install build-essential python-dev python-numpy
python-setuptools python-scipy

cl-berlin@Joy$ sudo easy_install -U scikit-learn

cl-berlin@Joy$ sudo apt-get install whois
```

Joy Components

joy
query.py
model.py
saltUI

Walkthrough

Using joy to process pcaps into flow data files

```
cl-berlin@Joy:~/joy$ bin/joy bidir=1 http=1 dns=1 tls=1 dist=1
../benign/capture.pcap > capture.gz
cl-berlin@Joy:~/joy$ zless capture.gz

{"version":"1.2","interface":"none","promisc":0,"output":"none",
"outputdir":"none","info":"none","count":0,"upload":"none","keyfile":"none",
"retain":0,"bidir":1,"num_pkts":0,"type":1,"zeros":0,"dist":1,"cdist":"none",
"entropy":0,"hd":0,"tls":1,"classify":0,"idp":0,"exe":0,"anon":"none",
"useranon":"none","bpf":"none","wht":0,"example":0,"dns":1,"ssh":0,"ip_id":0,
"salt":0,"verbosity":0}
{"sa":"10.0.2.15","da":"151.101.1.69","pr":6,"sp":37558,"dp":80,"ob":461,
"op":25,"ib":28652,"ip":28,"ts":1465338230.479314,"te":1465338261.040187,
"ottl":64,"ittl":64,"otcp_win":14600,"itcp_win":65535,"otcp_syn":40,
"otcp_nop":1,"otcp_mss":1460,"itcp_mss":1460,"otcp_wscale":7,"otcp_sack":1,
"otcp_tstamp":1,"packets":[{"b":461,"dir":">","ipt":55},

                                    ...
```

The first line is metadata that describes the options used to convert the pcap into JSON. The second line is a JSON description of a flow:

```
       {
          "sa":"10.0.2.15",         # source address
          "da":"23.56.181.48",      # destination address
          "pr":6,                   # protocol (TCP)
          "sp":43286,               # source port
          "dp":80,                  # destination port (HTTP)
          "ob":12834,               # number outbound bytes
          "op":93,                  # number outbound packets
          "ib":173193,              # number inbound bytes
          "ip":163,                 # number inbound packets
          "ts":1465315516.880170,   # time start (seconds since epoch)
          "te":1465315547.566770,   # time end (seconds since epoch)
          "ottl":64,                # outbound IP TTL
          "ittl":64,                # inbound IP TTL
          "otcp_win":14600,
          "itcp_win":65535,
          "otcp_syn":40,
          "otcp_nop":1,
          "otcp_mss":1460,
          "itcp_mss":1460,
          "otcp_wscale":7,
          "otcp_sack":1,
          "otcp_tstamp":1,
          "packets":[
             {"b":299,"dir":">","ipt":19},
             {"b":1248,"dir":"<","ipt":86},
             ...
```

Using query.py to process JSON flow data files

We can get output that looks like Netflow:

```
cl-berlin@Joy:~/joy$ ./query.py capture.gz --summary | less

source address  destination address prot  sport  dport obytes  opkts ibytes  ipkts       date     time  seconds
     10.0.2.15         151.101.1.69    6  37558     80    461     25  28652     28 2016-06-07 22:23:50  30.561
     10.0.2.15         151.101.1.69    6  37568     80    488      7    396      6 2016-06-07 22:23:50  30.234
     10.0.2.15         72.21.91.121    6  41646    443   1279     20  10080     19 2016-06-07 22:23:50  30.599
     10.0.2.15         72.21.91.121    6  41645    443   1279     19   9876     18 2016-06-07 22:23:50  30.599
     10.0.2.15         72.21.91.121    6  41647    443   1279     20   9460     19 2016-06-07 22:23:50  30.554
     10.0.2.15         72.21.91.121    6  41643    443   1730     21  10590     21 2016-06-07 22:23:50  30.662
     10.0.2.15         72.21.91.121    6  41648    443    828     16   8073     15 2016-06-07 22:23:50  30.594
     10.0.2.15         72.21.91.121    6  41644    443   1730     22  11259     22 2016-06-07 22:23:50  30.694
     10.0.2.15          172.217.1.4    6  60364    443    449     10   3915     12 2016-06-07 22:23:39  61.03
     10.0.2.15        66.150.118.20    6  58790     80    539      8    357      7 2016-06-07 22:23:51  40.371
     10.0.2.15       172.232.36.95    6  43882    443   1041     14   4704     13 2016-06-07 22:23:51  40.495
     10.0.2.15         23.23.170.52    6  56237    443   1026     14   4295     14 2016-06-07 22:23:51  40.219

                                                ...
```

But we can also get a lot more information:

```
cl-berlin@Joy:~/joy$ ./query.py capture.gz | less

{
   "itcp_mss": 1460,
   "ip": 9,
   "i_probable_os": "FreeBSD / OS X",
   "ib": 311,
   "pr": 6,
   "otcp_syn": 40,
   "otcp_win": 14600,
   "ts": 1465339229.163859,
   "ottl": 64,
   "te": 1465339288.182719,
   "otcp_nop": 1,
   "itcp_win": 65535,
   "otcp_mss": 1460,
   "otcp_sack": 1,
   "da": "172.217.1.196",

                           ...
```

The --select option will select certain fields to be printed:

```
cl-berlin@Joy:~/joy/saltUI$ ./query.py capture.gz --select "sa, da, dp"
```

The --where option will filter the flows to meet certain criteria.

```
cl-berlin@Joy:~/joy/saltUI$ ./query.py capture.gz --select "sa, da, dp"
--where "pr=17"
```

High entropy flows can be identified by using the Byte Distribution:

```
cl-berlin@Joy:~/joy$ ./query.py capture.gz --select "sa, da, dp, bd"
--where "entropy(bd)>7.99"
{
"name": [
    {"sa": "10.0.2.15", "da": "172.217.1.4", "dp": 443,
     "bd": [186, 70, 64, 97, 82, 69, 68, 61, 67, 63, 63, 63, 64, 60, 64,
63, 63, 64, 63, 66, 61, 61, 65, 78, 62, 61, 63, 60, 60, 64, 63, 58, 66, 63,
62, 62, 63, 66, 62, 67, 64, 62, 63, 64, 61, 63, 69, 63, 68, 65, 64, 61, 62,
65, 62, 66, 63, 63, 63, 60, 67, 60, 60, 63, 63, 58, 62, 61, 61, 62, 63,
62,60, 62, 62, 63, 60, 62, 65, 64, 64, 60, 57, 60, 63, 68, 61, 60, 61, 61,
63, 57, 59, 65, 64, 65, 62, 60, 61, 63, 60, 64, 61, 63, 66, 62, 60, 57, 62,
62, 62, 67, 65, 63, 63, 59, 66, 61, 62, 63, 63, 62, 64, 62, 64, 63, 65, 63,
64, 62, 61, 62, 59, 61, 65, 65, 60, 64, 63, 62, 60, 68, 63, 64, 61, 57, 63,
62, 58, 64, 64, 60, 61, 61, 62, 57, 64, 60, 64, 63, 61, 61, 60, 61, 59, 61,
61, 59, 63, 62, 58, 57, 62, 60, 64, 60, 59, 63, 63, 60, 62, 60, 61, 62, 62,
61, 60, 63, 64, 63, 63, 67, 63, 60, 60, 60, 62, 64, 63, 63, 62, 63, 61, 63,
63, 62, 63, 63, 63, 64, 62, 62, 59, 61, 64, 64, 63, 63, 61, 76, 62, 64, 60,
62, 63, 62, 64, 61, 63, 59, 65, 60, 62, 60, 61, 58, 61, 63, 62, 61, 64, 62,
64, 61, 63, 61, 61, 60, 62, 64, 64, 65, 62, 63, 63, 62] }
    ]
}
```

Looking at TLS:

```
cl-berlin@Joy:~/joy$ ./query.py capture.gz --where "dp=443" | less
 "tls": {
      "tls_irandom":
"57574d7d1e6a95a0304ccc2d71bb14765c2bd8a3cd966039bae034329471e462",
      "tls_iv": 5,
      "tls_ov": 5,
      "SNI": [
         "www.google.com"
      ],
      "srlt": [
         {
            "b": 512,
            "tp": "22:1",
            "ipt": 0,
            "dir": ">"
         },
      "tls_ext": [
         {
            "data": "00",
            "length": 1,
            "type": "ff01"
         },
     "tls_orandom":
"4e6fbeb21d3549c945fd52f17dc4190f195561117b970866dfd8651225691e59",
      "cs": [
         "c02b",
         "c02f",
         "c00a",
```

Looking at TLS security levels:

```
cl-berlin@Joy:~/joy$ ./query.py capture.gz --where "dp=443" --select "sa,
da, seclevel(tls)"

{
"name": [
{"sa": "10.0.2.15" , "da": "72.21.91.121" , "seclevel(tls)": "recommended"
},
                          ...
```

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:

```
cl-berlin@Joy:~/joy$ bin/joy bidir=1 dist=1 classify=1
../benign/capture.pcap > capture.gz
cl-berlin@Joy:~/joy$ ./query.py capture.gz --select "da, dp, p_malware"
--where "p_malware > 0.01"

{
"name": [
      {"da": "151.101.1.69", "dp": 80, "p_malware": 0.084},
      {"da": "172.217.1.195", "dp": 443, "p_malware": 0.021}
  ]
}
```

Example showing a similar run on a malicious PCAP:

```
cl-berlin@Joy:~/joy$ bin/joy bidir=1 dist=1 classify=1
../malware/133d773a8ca64c24bd81b594cf5240dd.pcap > malware.gz
cl-berlin@Joy:~/joy$ ./query.py malware.gz --select "da, dp, p_malware"
--where "p_malware > 0.99"

{
"name": [
        {  "da": 86.59.21.38,  "dp": 443 ,  "p_malware": 0.997 } ,
        {  "da": 108.61.179.216,  "dp": 443 ,  "p_malware": 0.995 } ,
        {  "da": 5.9.123.81,  "dp": 9001 ,  "p_malware": 0.999 } ,
        {  "da": 109.238.6.25,  "dp": 443 ,  "p_malware": 0.992 } ,
        {  "da": 176.31.159.231,  "dp": 443 ,  "p_malware": 0.998 } ,
                  ...
```

Making a classifier

We can use model.py to learn a 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 use the byte distribution, we run:

```
cl-berlin@Joy:~/joy$ bin/joy bidir=1 dist=1 ../malware/*.pcap >
../malware_train/malware.gz
cl-berlin@Joy:~/joy$ bin/joy bidir=1 dist=1 ../benign/*.pcap >
../benign_train/benign.gz
cl-berlin@Joy:~/joy$ python saltUI/model.py -m -l -t -p ../malware_train/
-n ../benign_train/ -o params.txt

Num Positive:     470
Num Negative:     276

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

non-zero parameters:    22
```

Then we generate the parameters that use the byte distribution:

```
cl-berlin@Joy:~/joy$ python saltUI/model.py -m -l -t -d -p
../malware_train/ -n ../benign_train/ -o params_bd.txt

Num Positive:    470
Num Negative:    276

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

non-zero parameters:    22
```

We can now visualize these classifiers by copying the params.txt and params_bd.txt files to the saltUI directory. Edit the saltUI/laui.cfg file by replacing line 16:

```
classifier Malware  logreg logreg_parameters.txt     logreg_parameters_bd.txt
```

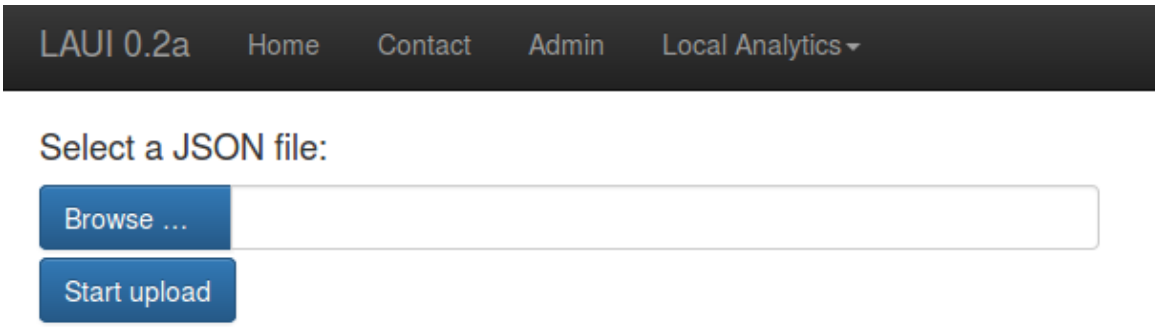
with:

```
classifier Malware      logreg  params.txt   params bd.txt
```

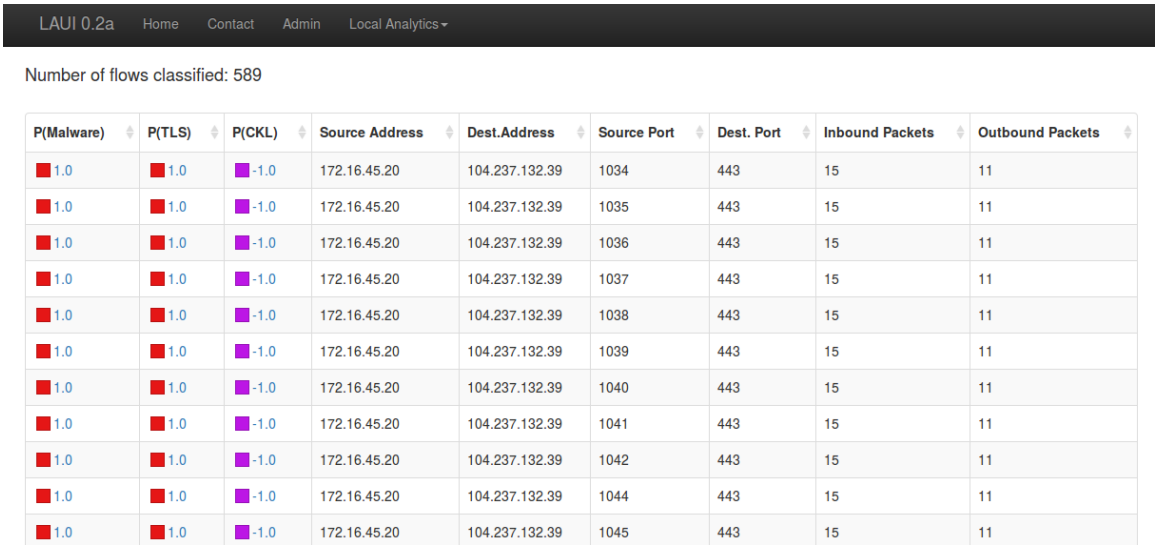
Move into the saltUI directory. Start the UI with:

```
cl-berlin@Joy:~/joy/saltUI$ python server.py
```

Point your browser to http://localhost:8080/home and click on Local Analytics -> Upload JSON File. You will then see the following page:



Click the "Browse…" button to choose the joy/malware.gz JSON file that we made earlier by processing pcaps (sitting under the root directory of joy). Finally, click the "Start upload" button. Using the new classifiers that we created, the results will be displayed:



P(Malware)	P(TLS)	P(CKL)	Source Address	Dest.Address	Source Port	Dest. Port	Inbound Packets	Outbound Packets
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1034	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1035	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1036	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1037	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1038	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1039	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1040	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1041	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1042	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1044	443	15	11
1.0	1.0	-1.0	172.16.45.20	104.237.132.39	1045	443	15	11