An Introduction to Monitoring Encrypted Network Traffic with Joy

Blake Anderson and David McGrew
Cisco Live US 2016 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

https://github.com/davidmcgrew/joy
make
install

```
sh$ sudo apt-get install build-essential python-dev python-numpy
python-setuptools python-scipy libatlas-dev libatlas3-base
sh$ sudo easy_install -U scikit-learn
sh$ sudo apt-get install whois
```

Joy Components

pcap2flow
query.py
model.py
saltUI

Using pcap2flow to process pcaps into flow data files

```
sh$ pcap2flow bidir=1 http=1 dns=1 tls=1 dist=1 capture2.pcap >
capture2.gz
sh$ zless capture2.gz
{"version":"1.1","interface":"none","promisc":0,"daemon":0,"output":"
none","outputdir":"none","info":"none","count":0,"upload":"none","key
file":"none","retain":0,"bidir":1,"num_pkts":0,"type":1,"zeros":0,"di
st":1,"cdist":"none","entropy":0,"wht":0,"hd":0,"tls":1,"classify":0,
"idp":0,"dns":1,"exe":0,"anon":"none","useranon":"none","bpf":"none",
"verbosity":0}
{"sa":"10.0.2.15","da":"23.56.181.48","pr":6,"sp":43286,"dp":80,"ob":
12834,"op":93,"ib":173193,"ip":163,"ts":1465315516.880170,"te":146531
5547.566770,"ottl":64,"ittl":64,"otcp_win":14600,"itcp_win":65535,"ot
cp_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}, …
```

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 to process JSON flow data files

We can get output that looks like Netflow:

```
sh$ query.py capture2.gz --summary | less
source address   destination address prot   sport  dport obytes   opkts ibytes   ipkts        date      time   seconds
     10.0.2.15        172.217.1.196     6   51932    443    710       8    311       9 2016-06-07 22:40:29  59.019
     10.0.2.15          4.31.198.44     6   36350     80   1262      17  11068      16 2016-06-07 22:41:16  33.277
     10.0.2.15         64.102.6.247    17   44385     53     32       1    214       1 2016-06-07 22:40:29  0.0
     10.0.2.15         64.102.6.247    17   57384     53     52       2    900       2 2016-06-07 22:40:58  0.0
     10.0.2.15          4.31.198.44     6   36344     80   1095      18  23760      27 2016-06-07 22:40:58  16.05
     10.0.2.15          64.102.6.247    17   60086     53     26       1    444       1 2016-06-07 22:40:58   0.0
```

But we can also get a lot more information:

```
sh$ query.py capture2.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",
   "otcp_wscale": 7,
   "dp": 443,
   "ittl": 64,
   "otcp_tstamp": 1,
   "sp": 51932,
   "packets": [
      {
         "b": 517,
         "ipt": 32,
         "dir": ">"
      },
      {
         "b": 168,
         "ipt": 28,
         "dir": "<"
      },
      {
         "b": 87,
         "ipt": 0,
         "dir": ">"
      },
      {
         "b": 65,
         "ipt": 0,
```

The --`select` option will select certain fields to be printed.

```
query.py capture2.gz --select "sa, da, dp"
```

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

```
query.py capture2.gz --select "sa, da, dp" --where "pr=17"
```

High entropy flows can be identified by using the Byte Distribution

```
sh$ query.py capture2.gz --select "sa, da, dp, bd" --where
"entropy(bd)>7.93"
{
"select": [
      {  "sa":  10.0.2.15 ,  "da":  4.31.198.44 ,  "dp":  80 ,  "bd":
[182, 40, 30, 39, 36, 31, 34, 44, 27, 33, 39, 39, 32, 39, 36, 49, 25,
34, 31, 28, 29, 40, 37, 37, 40, 40, 36, 38, 40, 38, 41, 57, 40, 27,
27, 44, 42, 33, 32, 40, 29, 34, 38, 51, 36, 46, 46, 49, 34, 42, 42,
38, 41, 49, 37, 36, 42, 46, 45, 36, 46, 47, 49, 60, 22, 30, 28, 36,
21, 34, 41, 46, 37, 47, 35, 43, 28, 41, 37, 58, 27, 34, 37, 42, 32,
40, 39, 52, 39, 40, 51, 50, 38, 42, 45, 62, 28, 38, 34, 48, 37, 55,
31, 46, 36, 60, 43, 59, 34, 50, 50, 57, 35, 43, 40, 47, 44, 50, 38,
47, 55, 43, 40, 40, 51, 44, 53, 61, 24, 30, 38, 35, 26, 36, 31, 39,
31, 35, 35, 34, 44, 40, 43, 51, 30, 38, 43, 33, 28, 41, 44, 41, 23,
29, 45, 37, 37, 36, 43, 52, 22, 32, 30, 42, 38, 44, 34, 50, 36, 44,
35, 46, 37, 47, 42, 60, 28, 37, 36, 36, 50, 51, 43, 54, 36, 44, 45,
40, 43, 39, 54, 57, 32, 42, 35, 45, 45, 39, 45, 53, 33, 37, 34, 42,
32, 41, 37, 53, 29, 39, 45, 46, 40, 38, 44, 58, 32, 39, 50, 45, 38,
43, 42, 47, 46, 53, 57, 54, 35, 44, 35, 41, 31, 42, 40, 46, 36, 49,
41, 45, 67, 67, 39, 43, 41, 44, 54, 42, 84, 44, 44, 55, 70, 52, 63,
68] }
   ]
}
```

Looking at TLS

```
sh$ query.py capture2.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_isid":
"5ff9b9c4356c4c102f3139daa4f58cfa968c5ff49b201fab81b287853e1c8c3a",
      "tls_osid":
"5ff9b9c4356c4c102f3139daa4f58cfa968c5ff49b201fab81b287853e1c8c3a",
      "tls_ext": [
         {
            "data": "00",
            "length": 1,
            "type": "ff01"
         },
     "tls_orandom":
"4e6fbeb21d3549c945fd52f17dc4190f195561117b970866dfd8651225691e59",
      "cs": [
         "c02b",
         "c02f",
         "c00a",
         "c009",
         "c013",
         "c014",
         "c012",
         "c007",
         "c011",
         "0033",
         "0032",
         "0045",
         "0039",
         "0038",
         "0088",
         "0016",
         "002f",
         "0041",
         "0035",
         "0084",
         "000a",
         "0005",
         "0004"
      ],
      "s_tls_ext": [
         {
            "data": "00",
            "length": 1,
            "type": "ff01"
         },
         {
            "data": "026832087370647 92f332e3108687474702f312e31",
            "length": 21,
            "type": "3374"
         },
         {
            "data": "0100",
            "length": 2,
            "type": "000b"
         }
      ],
      "scs": "c02f"
   }
```

Looking at TLS security levels

```
sh$ query.py capture2.gz --where "dp=443" --select "sa, da,
seclevel(tls)"
{
"name": [
      {  "sa":  "10.0.2.15" ,  "da":  "172.217.1.196" ,
"seclevel(tls)":  "recommended" } ,
      {  "sa":  "10.0.2.15" ,  "da":  "172.217.1.196" }
   ]
}
```

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 its Byte Distribution:

```
sh$ pcap2flow bidir=1 dist=1 classify=1 capture2.pcap > capture2.gz
sh$ query.py capture2.gz  --select "da, dp, p_malware" --where
"p_malware > 0.01"
{ "select": [
  {  "da":  4.31.198.44 ,  "dp":  80 , "p_malware":  0.038026 } ,
  {  "da":  4.31.198.44 ,  "dp":  80 ,  "p_malware":  0.035637 } ,
  {  "da":  4.31.198.44 ,  "dp":  80 ,  "p_malware":  0.011685 }
  ]
}
```

Example showing a similar run on a malicious PCAP:

```
sh$ pcap2flow bidir=1 dist=1 classify=1
133d773a8ca64c24bd81b594cf5240dd.pcap > malware.gz
sh$ 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:

```
sh$ pcap2flow bidir=1 dist=1 malware/*.pcap >
malware_train/malware.gz
sh$ pcap2flow bidir=1 dist=1 benign/*.pcap > benign_train/benign.gz
sh$ python ../joy/saltUI/model.py -m -l -t -p malware_train/ -n
benign_train/ -o params.txt

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

non-zero parameters:    21
```

and then we generate the parameters that do use the byte distribution:

```
sh$ python ../joy/saltUI/model.py -m -l -t -d -p malware_train/ -n
benign_train/ -o params_bd.txt
Num Positive:   599
Num Negative:   293

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

non-zero parameters:    24
```

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

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

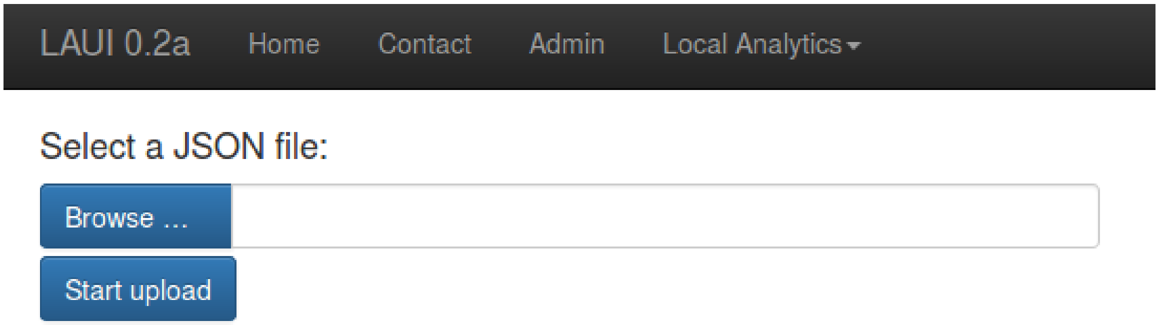
with:

```
classifier Malware      logreg   params.txt    params_bd.txt
```

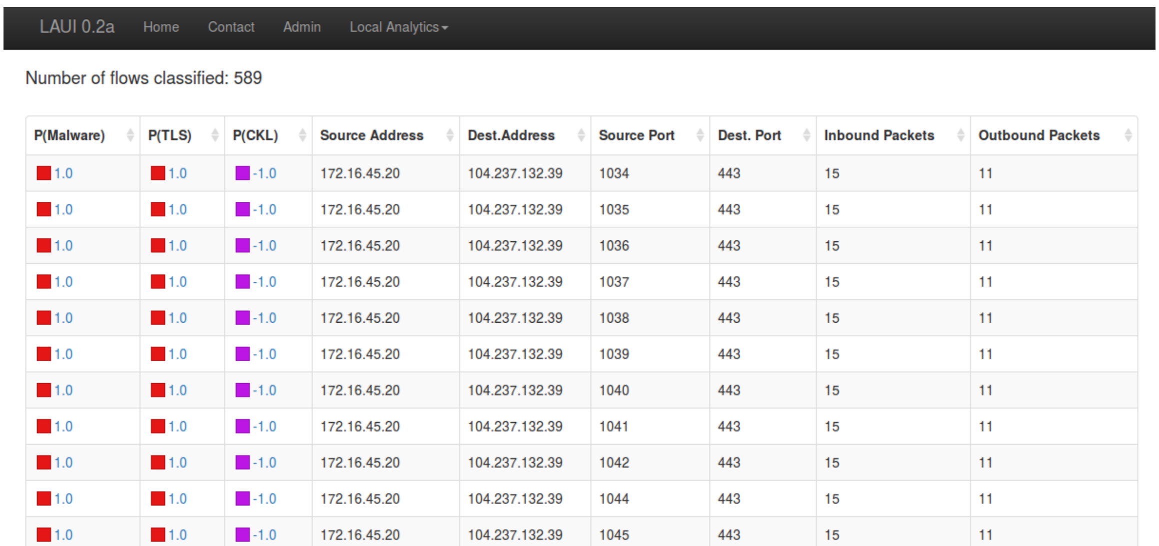
In the saltUI directory, start the UI with:

```
python server.py
```

And point your browser to http://localhost:8080/home, and click on `Local Analytics` -> `Upload JSON File`:



Upload a malicious, processed pcap, and the results using the new classifiers will be displayed:



Number of flows classified: 589

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