CTSC

CENTER FOR TRUSTWORTHY
SCIENTIFIC CYBERINFRASTRUCTURE
The NSF Cybersecurity Center of Excellence

Using awk to analyze Bro logs

Mark Krenz
BroCon 2017
September 12th, 2017

Center for Trustworthy Cyberinfrastructure

The NSF Cybersecurity Center of Excellence

CTSC’s mission is to provide the NSF community a coherent understanding of cybersecurity’s role in producing trustworthy science and the information and know-how required to achieve and maintain effective cybersecurity programs.

CENTER FOR APPLIED CYBERSECURITY RESEARCH
INDIANA UNIVERSITY
Pervasive Technology Institute

NCSA

PITTSBURGH SUPERCOMPUTING CENTER

THE UNIVERSITY of WISCONSIN MADISON

NSF

Speaker Bio - Mark Krenz

- Lead Security Analyst at Indiana University CACR (5 years)
- Part of the CTSC group
- System Administrator for 20 years
- Have worked in various sectors (private, government, academic)
- Creator of popular Twitter feed @climagic that : *https://twitter.com/climagic*

Agenda

- Give a brief introduction to:
  - The command line (This won't hurt, I promise)
  - Regular expressions
  - The awk command
- Provide you with real solutions to finding data in your Bro logs
  - Network Statistics
  - Security Incident Detection
  - Complex Analysis
  - $urprise!

THESE SLIDES WILL BE MADE AVAILABLE AFTER THE TALK

Color Coding Used For Commands In Slides

- commands
- options for commands
- filenames
- awk script
- output from commands
- | >   >>   (output redirection characters)
- comment text or prompt, don't type this

Common Commands for Processing Bro Logs

- cat, less, head and tail
- grep
- bro-cut
- sort
- uniq
- wc
- sed
- awk
- *many others…*

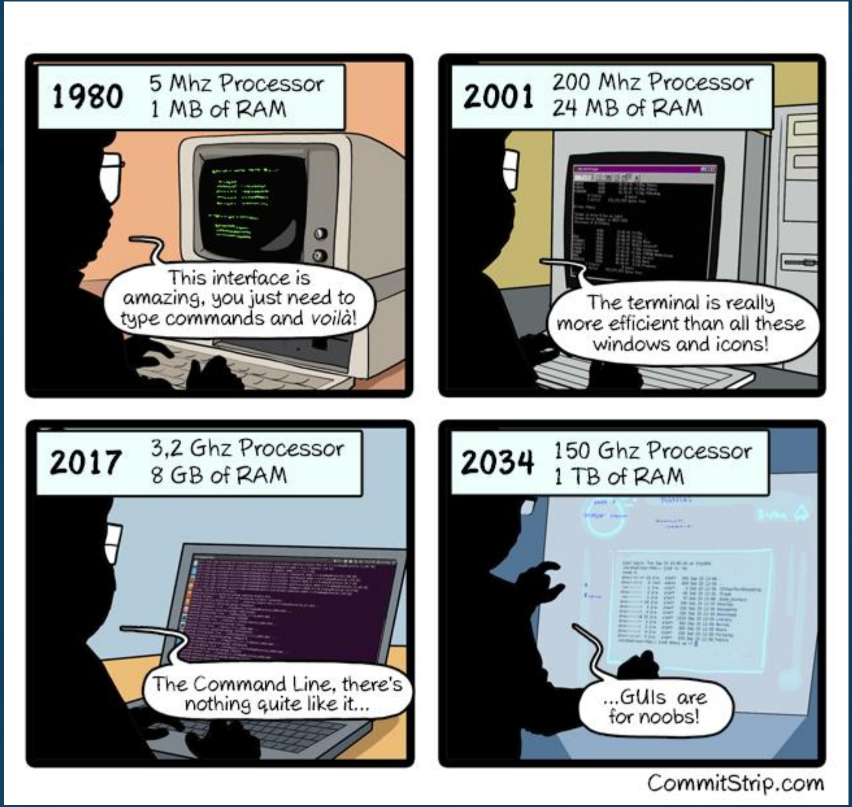

1980
5 Mhz Processor
1 MB of RAM
This interface is amazing, you just need to type commands and voilà!
2001
200 Mhz Processor
24 MB of RAM
The terminal is really more efficient than all these windows and icons!
2017
3,2 Ghz Processor
8 GB of RAM
The Command Line, there's nothing quite like it…
2034
150 Ghz Processor
1 TB of RAM
…GUIs are for noobs!
CommitStrip.com

Image source: http://www.commitstrip.com/en/2016/12/22/terminal-forever/

Command syntax (awk)

Pattern:

```
awk [options] <'program'> [file1] [file2] [...]
```

Starter program keywords:

- `{print $0}` (action statements)
- `$1, $2, ..., $NF`
- `$2=="foo",  $2!="foo"`
- `$3~/^[Bb]etty$/`
- `true || false,  true && true`
- (do this first) before doing this
- `variable=value`

How a command pipeline works

- Read in data, send output to next command
- Example (show list of id.orig_h IPs ordered by count)

```
$ zcat conn.log.gz | awk -F\\t '{print $3}' | sort | uniq -c | sort -rn
```

```
155489 172.16.0.10
  2836 172.16.0.5
  1456 172.16.0.13
   813 172.16.0.2
    64 172.16.0.7
```

Brief regular expression primer

- A regex can be used to match patterns of text data.
- Use " " or ' ' to protect expression from shell interpretation.
- `.` – matches any single character
- `\.` – Matches a literal . (use a \ before any special character)
- `.*` – matches any character zero or more times
- `.+` – matches any character 1 or more times
- `^` – Matches the beginning of the line.
- `$` – Matches the end of the line
- `[a-z]` – matches any letter between a and z in 1 position
- `[a-zA-Z0-9]` – Matches any alphanumeric in ASCII
- `[^0-9]` – Matches any character that is not 0 through 9.
- `[0-9]{1,3}` – Matches any character 0 - 9 between 1 and 3 times

CTSC

Regex Precision Is Important

Use ^2\.4\.150\.1$ to search for the IP 2.4.150.1

Why shouldn't I just run this?

```
grep "2.4.150.1" access_log
```

Because it will also match :

```
22.4.150.15
204.150.100.10
```

and these values:

```
2E4150A1
/script.php?id=12948150218
```

Detect Hosts Searching For Exploitable Code

Which IP had the most HTTP 404 Not Found errors?

- What is a 404 not found error?
  - HTTP status return code to the client
- What logs track this information?
  - Bro's http.log
- What field is it in the bro log?
  - status_code
- How can we match a number in a log? *
  - awk, grep, sed, search
- How can we generate a top list? *
  - Collect like groups (sort)
  - Count the number of items in each group (uniq -c)
  - Order the counts. (sort -n)

CTSC

Recon Detection Command (404s)

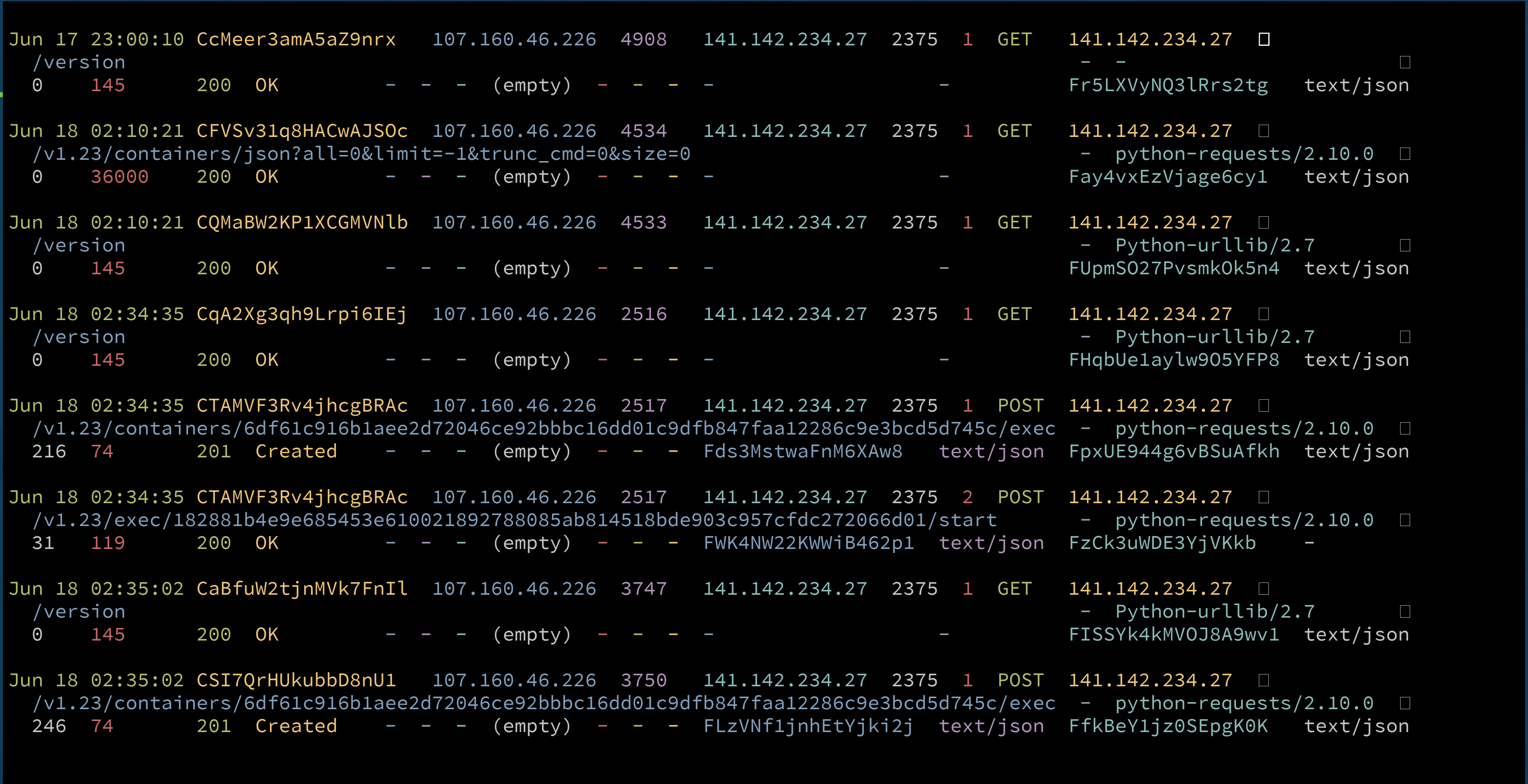
```
$ cat http.log |
bro-cut id.orig_h status_code |
awk -F\\t '$2=="404"' |
sort | uniq -c | sort -n |
tail -n 1

165 64.39.106.131   404

$ dig +short -x 64.39.106.131

sn031.s01.sea01.qualys.com
```

Detect If Web App Tried To Read Filesystem

Do any successful queries to Wordpress code contain filesystem paths in the query string?

- Where do wordpress requests get logged?
  - Bro's http.log
- What should I search for?
  - Filesystem path indicators like '/', '..', '/etc' or
  - Specific filenames like my.cnf, passwd, .htaccess
- How can I figure out if the exploit attempt worked?
  - HTTP return status (if 404, then probably not; 200 only means potentially)
  - Does the file referenced exist?

Compromise 2: http.log



```
Jun 17 23:00:10 CcMeer3amA5aZ9nrx    107.160.46.226  4908   141.142.234.27  2375  1  GET   141.142.234.27  □
  /version                                                                                 -  -              □
  0     145      200  OK          -  -  -  (empty)  -  -  -  -                    -         Fr5LXVyNQ3lRrs2tg   text/json

Jun 18 02:10:21 CFVSv31q8HACwAJSOc  107.160.46.226  4534   141.142.234.27  2375  1  GET   141.142.234.27  □
  /v1.23/containers/json?all=0&limit=-1&trunc_cmd=0&size=0                                 -  python-requests/2.10.0  □
  0     36000    200  OK          -  -  -  (empty)  -  -  -  -                    -         Fay4vxEzVjage6cy1   text/json

Jun 18 02:10:21 CQMaBW2KP1XCGMVNlb  107.160.46.226  4533   141.142.234.27  2375  1  GET   141.142.234.27  □
  /version                                                                                 -  Python-urllib/2.7     □
  0     145      200  OK          -  -  -  (empty)  -  -  -  -                    -         FUpmSO27PvsmkOk5n4  text/json

Jun 18 02:34:35 CqA2Xg3qh9Lrpi6IEj  107.160.46.226  2516   141.142.234.27  2375  1  GET   141.142.234.27  □
  /version                                                                                 -  Python-urllib/2.7     □
  0     145      200  OK          -  -  -  (empty)  -  -  -  -                    -         FHqbUe1aylw9O5YFP8  text/json

Jun 18 02:34:35 CTAMVF3Rv4jhcgBRAc  107.160.46.226  2517   141.142.234.27  2375  1  POST  141.142.234.27  □
  /v1.23/containers/6df61c916b1aee2d72046ce92bbbc16dd01c9dfb847faa12286c9e3bcd5d745c/exec  -  python-requests/2.10.0  □
  216   74       201  Created     -  -  -  (empty)  -  -  -  Fds3MstwaFnM6XAw8   text/json  FpxUE944g6vBSuAfkh  text/json

Jun 18 02:34:35 CTAMVF3Rv4jhcgBRAc  107.160.46.226  2517   141.142.234.27  2375  2  POST  141.142.234.27  □
  /v1.23/exec/182881b4e9e685453e610021892788085ab814518bde903c957cfdc272066d01/start      -  python-requests/2.10.0  □
  31    119      200  OK          -  -  -  (empty)  -  -  -  FWK4NW22KWWiB462p1  text/json  FzCk3uWDE3YjVKkb    -

Jun 18 02:35:02 CaBfuW2tjnMVk7FnIl  107.160.46.226  3747   141.142.234.27  2375  1  GET   141.142.234.27  □
  /version                                                                                 -  Python-urllib/2.7     □
  0     145      200  OK          -  -  -  (empty)  -  -  -  -                    -         FISSYk4kMVOJ8A9wv1  text/json

Jun 18 02:35:02 CSI7QrHUkubbD8nU1   107.160.46.226  3750   141.142.234.27  2375  1  POST  141.142.234.27  □
  /v1.23/containers/6df61c916b1aee2d72046ce92bbbc16dd01c9dfb847faa12286c9e3bcd5d745c/exec  -  python-requests/2.10.0  □
  246   74       201  Created     -  -  -  (empty)  -  -  -  FLzVNf1jnhEtYjki2j  text/json  FfkBeY1jz0SEpgK0K   text/json
```

Recon detection command (web app)

```
$ awk -F\\t '$10~/\.\.\//' http.log
1486703681.865315    C57Abb4C4F651y171f    172.16.17.106
42470    36.158.63.186    80    1    GET
www.acmewidgets.com
/wp-admin/admin-ajax.php?action=revslider_show_image&img=..
/../.my.cnf    -    Mozilla/5.0    0    3    200    OK    -
-    -    (empty)    -    -    -    --FiU9vrD2d9PPvMQJc
-
```

Detect If A New Exploit Hit Us In The Past?

Given that the recent Intel AMT vulnerability has been hidden in chips since 2010, can we find any indication of previous attacks against our network?

- What are we looking for?
  - meta data about traffic to tcp ports 16992 and 16993
- Where can we find this?
  - Bro's conn.log
- How can we be sure the connections were successful?
  - Check that the conn_status column in conn.log is not "`S0`".
- Make a list of potential attackers first, save it to a file.
- Then investigate the overall activity of the potentials.

Recon detection command (Intel AMT)

```
$ zcat 201[0-7]-*/conn.*.log.gz |
  cat - current/conn.log |
  awk -F\\t '($6==16992 || $6==16993) && $12!="S0"
  {print $3}' > potential-attackers.txt
$ zgrep -F -f potential-attackers.txt
201[0-7]-*/conn.*.log.gz current/conn.log
```

Image source: https://upload.wikimedia.org/wikipedia/en/3/3a/Hacker_inside.jpg

Logins Vs. Non-work Time

Can we analyze a log to show entries of login activity outside of normal working hours?

- What service do we want to check against?
  - SSH
- What logs provide this information?
  - Bro's ssh.log
- How to compare time of day?
  - Use bro-cut to convert ts column to parsable local time.
  - Use awk's substr() function to get the hour of the day from the timestamp.

Break It Down: Getting A Sub-string

```
substr(<string>, <starting index*>, <length of substring>)
(*starting index is from 1, not 0.)

substr("this is easy", 9, 4);
easy

($1 = 2017-01-24T04:03:58-0400)
substr($1,12,2)
04
```

Break It Down: Making Comparisons

```
$0!~/^#/   (Don't print lines starting with comment characters)

$4=="T" && $5=="INBOUND" (Successful inbound logins)

if (true) { do something } else { do something else }

if (hour < 9 || hour >= 17) { print } (♬ Not Workin' 9 to 5 ♬)

true && true || false { print }
```

CTSC

Logins Vs. Non-work Time

```
(Check for inbound successful logs not between 9am and 5pm)
$ cat ssh.log |
 bro-cut -C -d ts id.orig_h id.resp_h auth_success direction |
 awk -F\\t '$0!~/^#/ && $4=="T" && $5 == "INBOUND"
 { hour=int(substr($1,12,2)); if (hour < 9 || hour >= 17)
  {print}}' | less -S
```

```
2017-04-01T06:45:18-0400  154.19.91.90     10.0.4.26   T      INBOUND
2017-04-01T06:47:13-0400  154.19.91.90     10.0.1.5    T      INBOUND
2017-04-01T19:05:44-0400  154.19.91.90     10.0.1.5    T      INBOUND
```

Logins Vs. Non-work Time

- Alternate way using modulus of epoch time.
- % is modulus operator, gives you the remainder after division.
- Unix epoch time modulo 86400 will give you the same time of day no matter what the day

```
$ cat ssh.log |
  awk -F\\t '$8=="T" && $9 == "INBOUND" &&
  ($1 % 86400 < 43200 || $1 % 86400 > 75600) {print}' |
  less -S
```

Break It Down: Awk Array Primer

- An array stores a set of values. ['a', 10, "kiwi", "192.168.0.5", ...]
- You can perform operations on the array and it's values.
- `a[0] = "Tabitha Gallagher"` (store a value by numeric index)
- `u['tgallagher'] = "Tabitha Gallagher"` (store a value by text key)
- `conns['10.0.0.2 66.8.54.3'] = 203` (Complex key made of IPs)
- `u['ishort']['name'] = "Ira Short"` (multi-dimensional array)
- `length(u)` (Get the number of keys in the 'users' array)

Detecting Brute Force Success

Can we track failure or success for a service?

- Bro's ssh.log

How to keep track of failures?

- If failure, increment a value in an array for that IP pair
- If "success" and fail count has passed a threshold? Then print.
- Delete successful connection pairs to reset count.

Detect Brute Force Success (After 20+ tries)

```
$ zcat 2017-*/ssh*.gz | cat - current/ssh.log |
 bro-cut -d -C ts uid id.orig_h id.resp_h auth_success |
 awk -F\\t '{ pairkey=$3 ":" $4;
if ($5 != "T") { fails[pairkey]++;
} else {
   if (fails[pairkey] > 20) {
      print $0 " after " fails[pairkey] " tries";
   }
   delete fails[pairkey]; }
}}'
2017-08-02T05:15:04-0500   CyAM04646e0f7ad4  42.81.18.7  107.16.2.47   T after 5082 tries
```

Attack Reinforcement Detection

How can we detect when someone installs a backdoor?

- What type of service is being backdoored?
  - SSH
- How could we tell if it's been backdoored?
  - SSH server version number change
  - Server side binary file size or checksum
- What logs can we use for software version change?
  - Bro's software.log
- What tool can we use to detect a change?
  - awk: Store the last version seen and compare with current line's version
  - `if (lastversion != $4) { print; lastversion=$4 }`

Attack Reinforcement Detection

From software.log:

```
Jul 27 19:32:19 141.142.227.45 22 SSH::SERVER OpenSSH_6.6.1p1
Jul 27 20:29:39 141.142.227.45 22 SSH::SERVER OpenSSH_6.6.1p1
Jul 27 22:27:53 141.142.227.45 22 SSH::SERVER OpenSSH_6.6.1p1
Jul 27 23:30:34 141.142.227.45 22 SSH::SERVER OpenSSH_6.5.1p1
```

Attack Reinforcement Detection Command

```
$ cat software.log |
bro-cut -C -d ts host host_p unparsed_version |
awk -F\\t '$2=="141.142.227.45" && $3=="22"
{ if (lastversion != $4) { print; lastversion=$4 } }'

Jul 27 22:27:53 141.142.227.45 22 OpenSSH_6.6.1p1
Jul 27 23:30:34 141.142.227.45 22 OpenSSH_6.5.1p1
```

MySQL Log Analysis Command

Checking for a large number of returned rows

```
$ cat mysql.log |
bro-cut -C -d ts id.orig_h id.resp_h success rows |
awk -F\\t '$3=="T" && $4 > 1000 { print }'
```

MySQL Log Analysis Command

SQL queries coming from odd networks or hosts

```
$ cat mysql.log |
bro-cut -C -d ts id.orig_h id.resp_h success |
awk -F\\t '$2 !~ /^172\.16\.50\./ && $3=="T" { print }'
```

Image source: https://xkcd.com/327/

CTSC

Large Exfiltration Of Data

Large outbound transfers from sensitive networks (172.17.50.0/24)

```
$ cat conn.log |
bro-cut -C -d ts id.orig_h id.resp_h resp_ip_bytes |
awk -F\\t '$3~/^172\.17\.50\./ && $4 > 100000000
{ print }'
```

Image source: http://en.rocketnews24.com/

Large Exfiltration Of Data

Large outbound transfers from sensitive networks (172.17.50.0/24)

```
$ cat conn.log |
bro-cut -C -d ts id.orig_h id.resp_h resp_ip_bytes |
awk -F\\t '$3~/^172\.17\.50\./ && $4 > 100000000
{ print }'
```

```
2017-05-26T13:08:32-0400      172.17.50.7  172.17.49.42    3020603598
2017-05-26T15:11:04-0400      172.17.50.7  16.58.192.193   5031339532
2017-05-26T18:09:24-0400      172.17.50.2  57.49.32.164    171755661
2017-05-26T22:15:40-0400      172.17.50.8  172.16.9.5      1420997210
```

CTSC

Detect Protocol Mismatch

Show instances of ssh running on port 80 or 443

```
$ cat conn.log |
bro-cut -C -d ts id.orig_h id.resp_h id.resp_p service |
awk -F\\t '($4 == 80 || $4 == 443) && $5 == "ssh"'
```

```
2017-05-02T04:03:34-0400      172.17.40.104     42.71.10.49  443  ssh
```

Image source: http://s2.quickmeme.com/

CTSC

Accessing Bro log columns by name (bawk)

Wouldn't it be great if you could just run awk commands like these?

```
$ bawk '$_b["id.resp_h"] ~ /10\.0\.1\./' http.log


$ bawk 'geoip( $_b["id.orig_h"] ) == "XZ"' ssh.log


$ cat http.log conn.log | bawk '{
if ( _log_path == "http" ) {
    if ( $_b["uri"] ~ /malwarestring/ ) {  uids[$_b["uid"]=1 }
} else if ( _log_path == "conn" && uids[$_b["uid"]] ) { print }'
```

Git it here: **https://github.com/deltaray/bawk**

CTSC

Accessing Bro log columns by name (bawk)

From */opt/bro/lib/bawk/getlogheaders.awk* :

```
/^#/ {
    if ($0~/^#fields/) {
        for (i=2; i<=NF; i++) {
            _b[$i]=i-1
        };
    }
   print; next;
}
```

Git it here: **https://github.com/deltaray/bawk**

Accessing Bro Log Columns By Name (bawk)

Finding potential video call users

```
$ zcat 2017-08-*/conn.00\:00\:00-00\:00\:00.log.gz |
bawk '
$_b["id.resp_p"] >= 3478 && $_b["id.resp_p"] <= 3481
 { caller[ $_b["id.orig_h"] ] = $_b["ts"] }
$_b["resp_ip_bytes"] > 2000000
  && ( $_b["ts"] - caller[ $_b["id.orig_h"] ] < 300
       && $_b["proto"]=="udp"
       && $_b["id.resp_p"] > 1023 )
  { print }'
```

Git it here: **https://github.com/deltaray/bawk**

Using scripts for complex commands

- Recommend using shell scripts to save complex and often reused analysis commands.
- Make the scripts adaptable through use of arguments.
- Run them regularly from cron

CTSC

CENTER FOR TRUSTWORTHY SCIENTIFIC CYBERINFRASTRUCTURE

The NSF Cybersecurity Center of Excellence

CTSC webinar series: trustedci.org/webinars

Mailing list: trustedci.org/ctsc-email-lists

We thank the National Science Foundation (grant 1547272) for supporting our work.

CTSC

CENTER FOR TRUSTWORTHY SCIENTIFIC CYBERINFRASTRUCTURE

The NSF Cybersecurity Center of Excellence

Thank You

NCSA:
ncsa.illinois.edu
@NCSAatIllinois

SWAMP:
continuousassurance.org
@SWAMPTEAM

CTSC:
trustedci.org
@TrustedCI

CACR:
cacr.iu.edu
@iucacr

Bro:
bro.org
@Bro_IDS

CLI Magic:
climagic.org
@climagic

Questions? Comments? Contact the presenter at mkrenz@iu.edu

We thank the National Science Foundation (grant 1547272) for supporting our work.

cray cray.