# **Bro IDS**

An Intrusion Detection System (IDS) allows you to detect suspicious activities happening on your network as a result of a past or active attack. Because of its programming capabilities, Bro can easily be configured to behave like traditional IDSs and detect common attacks with well known patterns, or you can create your own scripts to detect conditions specific to your particular case.

In the following sections, we present a few examples of common uses of Bro as an IDS.

# BRO NETWORK SECURITY MONITOS

#### TABLE OF CONTENTS

- Detecting an FTP Brute-force Attack and Notifying
- Other Attacks
  - Detecting SQL Injection Attacks
  - Checking files against known malware hashes

# **Detecting an FTP Brute-force Attack and Notifying**

For the purpose of this exercise, we define FTP brute-forcing as too many rejected usernames and passwords occurring from a single address. We start by defining a threshold for the number of attempts, a monitoring interval (in minutes), and a new notice type.

#### NEXT PAGE

MIME Type Statistics

#### PREVIOUS PAGE

```
1
                                      detect-bruteforcing.bro
                                                                                   Monitoring HTTP Traffic with Bro
2
 3
    module FTP;
 4
                                                                                   SEARCH
 5
    export {
 6
             redef enum Notice::Type += {
                     ## Indicates a host bruteforcing FTP logins by watching fo
 7
 8
                     ## many rejected usernames or failed passwords.
 9
                     Bruteforcing
10
             };
11
12
             ## How many rejected usernames or passwords are required before being
13
             ## considered to be bruteforcing.
14
             const bruteforce threshold: double = 20 &redef;
15
16
             \#\# The time period in which the threshold needs to be crossed before
17
             ## being reset.
18
             const bruteforce_measurement_interval = 15mins &redef;
19
```

Using the ftp\_reply event, we check for error codes from the 500 series for the "USER" and "PASS" commands, representing rejected usernames or passwords. For this, we can use the FTP::parse\_ftp\_reply\_code function to break down the reply code and check if the first digit is a "5" or not. If true, we then use the Summary Statistics Framework to keep track of the number of failed attempts.

```
1
                                                    detect-bruteforcing.bro
 2
 3
    event ftp reply(c: connection, code: count, msg: string, cont resp: bool)
 4
5
            local cmd = c$ftp$cmdarg$cmd;
 6
             if ( cmd == "USER" || cmd == "PASS" )
 7
                     {
 8
                     if (FTP::parse_ftp_reply_code(code)$x == 5 )
 9
                             SumStats::observe("ftp.failed auth", [$host=c$id$orig h], [$str=cat(c$id$resp h)]);
10
11
```

Next, we use the SumStats framework to raise a notice of the attack when the number of failed attempts exceeds the specified threshold during the measuring interval.

```
1
                                                                         detect-bruteforcing.bro
2
 3
    event bro_init()
4
            {
5
            local r1: SumStats::Reducer = [$stream="ftp.failed_auth", $apply=set(SumStats::UNIQUE), $unique_max
 6
            SumStats::create([$name="ftp-detect-bruteforcing",
 7
                               $epoch=bruteforce measurement interval,
 8
                               $reducers=set(r1),
9
                               $threshold val(key: SumStats::Key, result: SumStats::Result) =
10
11
                                     return result["ftp.failed auth"]$num+0.0;
12
                                     },
13
                               $threshold=bruteforce threshold,
14
                               $threshold crossed(key: SumStats:::Key, result: SumStats:::Result) =
15
16
                                     local r = result["ftp.failed auth"];
17
                                     local dur = duration to mins secs(r$end-r$begin);
18
                                     local plural = r$unique>1 ? "s" : "";
19
                                     local message = fmt("%s had %d failed logins on %d FTP server%s in %s", key
20
                                     NOTICE([$note=FTP::Bruteforcing,
21
                                              $src=key$host,
22
                                              $msg=message,
23
                                              $identifier=cat(key$host)]);
24
                                     }]);
25
```

#### Below is the final code for our script.

```
1
                                                                         detect-bruteforcing.bro
2
 3
    ##! FTP brute-forcing detector, triggering when too many rejected usernames or
 4
    ##! failed passwords have occurred from a single address.
 5
 6
    @load base/protocols/ftp
 7
    @load base/frameworks/sumstats
8
9
    @load base/utils/time
10
11
    module FTP;
12
13
   export {
14
            redef enum Notice::Type += {
1.5
                     ## Indicates a host bruteforcing FTP logins by watching for too
16
                     ## many rejected usernames or failed passwords.
17
                     Bruteforcing
18
            };
19
20
            ## How many rejected usernames or passwords are required before being
2.1
            ## considered to be bruteforcing.
22
            const bruteforce threshold: double = 20 &redef;
23
24
            ## The time period in which the threshold needs to be crossed before
25
            ## being reset.
26
            const bruteforce_measurement_interval = 15mins &redef;
27
28
29
30
   event bro_init()
31
32
            local r1: SumStats::Reducer = [$stream="ftp.failed_auth", $apply=set(SumStats::UNIQUE), $unique_max
33
            SumStats::create([$name="ftp-detect-bruteforcing",
34
                               $epoch=bruteforce measurement interval,
35
                               $reducers=set(r1),
36
                               $threshold val(key: SumStats:::Key, result: SumStats:::Result) =
37
```

```
return result["ftp.failed auth"]$num+0.0;
38
39
                                     },
40
                               $threshold=bruteforce_threshold,
41
                               $threshold crossed(key: SumStats:::Key, result: SumStats:::Result) =
42
43
                                     local r = result["ftp.failed_auth"];
                                     local dur = duration_to_mins_secs(r$end-r$begin);
44
                                     local plural = r$unique>1 ? "s" : "";
45
46
                                     local message = fmt("%s had %d failed logins on %d FTP server%s in %s", key
47
                                     NOTICE([$note=FTP::Bruteforcing,
48
                                             $src=key$host,
49
                                             $msg=message,
50
                                             $identifier=cat(key$host)]);
51
                                     }]);
52
53
54
    event ftp_reply(c: connection, code: count, msg: string, cont_resp: bool)
55
56
            local cmd = c$ftp$cmdarg$cmd;
57
            if ( cmd == "USER" || cmd == "PASS" )
58
59
                    if (FTP::parse_ftp_reply_code(code)$x == 5 )
                             SumStats::observe("ftp.failed_auth", [$host=c$id$orig_h], [$str=cat(c$id$resp_h)]);
60
61
                     }
62
```

# bro -r ftp/bruteforce.pcap protocols/ftp/detect-bruteforcing.bro

```
4
  1
    #separator \x09
    #set_separator
    #empty_field
                   (empty)
  4
    #unset field
  5
    #path notice
    #open 2018-05-23-00-22-00
  6
     #fields ts uid id.orig_h id.orig_p
                                                  id.resp h
  8
    #types time string addr port
                                      addr port
                                                   string strin
  9
    1389721084.522861 - -
 10
    #close
           2018-05-23-00-22-00
```

As a final note, the detect-bruteforcing.bro script above is included with Bro out of the box. Use this feature by loading this script during startup.

## Other Attacks

### **Detecting SQL Injection Attacks**

## Checking files against known malware hashes

Files transmitted on your network could either be completely harmless or contain viruses and other threats. One possible action against this threat is to compute the hashes of the files and compare them against a list of known malware hashes. Bro simplifies this task by offering a detect-MHR.bro script that creates and compares hashes against the Malware Hash Registry maintained by Team Cymru. Use this feature by loading this script during startup.

Copyright 2016, The Bro Project. Last updated on May 22, 2018. Created using Sphinx 1.6.6.

© 2014 The Bro Project.

Internal Pages