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.

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.

detect-bruteforcing.bro

```
module FTP;

export {
        redef enum Notice::Type += {
                ## Indicates a host bruteforcing FTP logins by watching for
                ## many rejected usernames or failed passwords.
                Bruteforcing
        };

        ## How many rejected usernames or passwords are required before being
        ## considered to be bruteforcing.
        const bruteforce_threshold: double = 20 &redef;

        ## The time period in which the threshold needs to be crossed before
        ## being reset.
        const bruteforce_measurement_interval = 15mins &redef;
}
```

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.

detect-bruteforcing.bro

```
event ftp_reply(c: connection, code: count, msg: string, cont_resp: bool)
        {
        local cmd = c$ftp$cmdarg$cmd;
        if ( cmd == "USER" || cmd == "PASS" )
                {
                if ( FTP::parse_ftp_reply_code(code)$x == 5 )
                        SumStats::observe("ftp.failed_auth", [$host=c$id$orig_h], [$str=cat(c$id$resp_h)]);
                }
        }
```

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.

detect-bruteforcing.bro

```
event bro_init()
        {
        local r1: SumStats::Reducer = [$stream="ftp.failed_auth", $apply=set(SumStats::UNIQUE), $unique_max
        SumStats::create([$name="ftp-detect-bruteforcing",
                          $epoch=bruteforce_measurement_interval,
                          $reducers=set(r1),
                          $threshold_val(key: SumStats::Key, result: SumStats::Result) =
                                {
                                return result["ftp.failed_auth"]$num+0.0;
                                },
                          $threshold=bruteforce_threshold,
                          $threshold_crossed(key: SumStats::Key, result: SumStats::Result) =
                                {
                                local r = result["ftp.failed_auth"];
                                local dur = duration_to_mins_secs(r$end-r$begin);
                                local plural = r$unique>1 ? "s" : "";
                                local message = fmt("%s had %d failed logins on %d FTP server%s in %s", key
                                NOTICE([$note=FTP::Bruteforcing,
                                        $src=key$host,
                                        $msg=message,
                                        $identifier=cat(key$host)]);
                                }]);
        }
```

Below is the final code for our script.

detect-bruteforcing.bro

```
##! FTP brute-forcing detector, triggering when too many rejected usernames or
##! failed passwords have occurred from a single address.

@load base/protocols/ftp
@load base/frameworks/sumstats

@load base/utils/time

module FTP;

export {
        redef enum Notice::Type += {
                ## Indicates a host bruteforcing FTP logins by watching for too
                ## many rejected usernames or failed passwords.
                Bruteforcing
        };

        ## How many rejected usernames or passwords are required before being
        ## considered to be bruteforcing.
        const bruteforce_threshold: double = 20 &redef;

        ## The time period in which the threshold needs to be crossed before
        ## being reset.
        const bruteforce_measurement_interval = 15mins &redef;
}


event bro_init()
        {
        local r1: SumStats::Reducer = [$stream="ftp.failed_auth", $apply=set(SumStats::UNIQUE), $unique_max
        SumStats::create([$name="ftp-detect-bruteforcing",
                          $epoch=bruteforce_measurement_interval,
                          $reducers=set(r1),
                          $threshold_val(key: SumStats::Key, result: SumStats::Result) =
                                {
```

```
                                return result["ftp.failed_auth"]$num+0.0;
                                },
                        $threshold=bruteforce_threshold,
                        $threshold_crossed(key: SumStats::Key, result: SumStats::Result) =
                                {
                                local r = result["ftp.failed_auth"];
                                local dur = duration_to_mins_secs(r$end-r$begin);
                                local plural = r$unique>1 ? "s" : "";
                                local message = fmt("%s had %d failed logins on %d FTP server%s in %s", key
                                NOTICE([$note=FTP::Bruteforcing,
                                        $src=key$host,
                                        $msg=message,
                                        $identifier=cat(key$host)]);
                                }]);
        }

event ftp_reply(c: connection, code: count, msg: string, cont_resp: bool)
        {
        local cmd = c$ftp$cmdarg$cmd;
        if ( cmd == "USER" || cmd == "PASS" )
                {
                if ( FTP::parse_ftp_reply_code(code)$x == 5 )
                        SumStats::observe("ftp.failed_auth", [$host=c$id$orig_h], [$str=cat(c$id$resp_h)]);
                }
        }
```

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

```
#separator \x09
#set_separator    ,
#empty_field      (empty)
#unset_field      -
#path     notice
#open     2018-05-23-00-22-00
#fields   ts      uid     id.orig_h       id.orig_p       id.resp_h
#types    time    string  addr    port    addr    port    string  strin
1389721084.522861 -       -       -       -       -       -       -
#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.

Internal Pages