Bro: Actively defending so that you can do other stuff

Aashish Sharma
& LBNL Cyber Security Team
Lawrence Berkeley National Lab
The goal of this talk is to provide you with detailed insights into deployment of dynamic firewalling capabilities of Bro while we continue to keep the network open, unrestricted and functional.
“Ah! Idea is simple- when you find something is worth blocking Block it”

- Implementation Complexity
  - False positives?
  - Can we rely on external data feeds and blindly block indicators?
- Checks and balances used to weed out false positives.
Desired goals

- Reliability
- Redundancy
- Accuracy and accountability
- Fast reaction time - BRO
  - Must block really fast scanners
  - Must block really slow scanners
- Fast reaction time - ACLD
- Smart ACL mgmt - keep scanners blocked only until active - no unnecessary acl consumption
- State management
  - Block sooner if they come back (catch-n-release)
  - Very very long state management (bloom filters)
Mechanisms for active-defense

- Scan detections
  - old-scan suite
  - new scan policies (based on sumstats)
- Deep-blocks
  - HTTP, domains, user-agents
  - IRC
  - SMTP
- Application Specific blocking
  - NTP
  - Heartbleed
  - SIP
- Intel-framework
  - Intelligence feeds
  - Local blacklists
  - Local domains
- + other similar
Cyber Security: Border Access Control and Visibility

Legend

- network traffic
- copy of network traffic
- flow data
- block commands
- network equipment
- tapping equipment
- cyber security equipment
- Lab computers

This diagram is for illustrative purposes only, additional technical details require a narrative.

Jay Krous, July 30, 2012
Catch-n-release

- Catch-n-release is an ACL management system
- Acts like a dynamic firewall for the institution
- It works on the top of the scan detection algorithms and acts as a reaction-framework working in conjunction with intrusion detection.
- The premise of catch-n-release is
  - No point keeping a scanning IP blocked if it has ‘moved-on’ from scanning our network space.
Catch-n-release

- This allows us to expire stale blocks and efficiently use limited ACL’s for the current threat.
- At the same time, this allows us to make our scan-detection heuristics very sensitive i.e. once a previously known scanner hits again, it gets blocks immediately with a much lower scan threshold.
- Additionally, catch-n-release system provides us with capability of a dynamic block and expiration timer. A persistent scanner gets incrementally blocked for a much larger duration.

- Smart ACL mgmt - keep scanners blocked only until active - no unnecessary acl consumption
- State management
  - Block sooner if they come back (catch-n-release)
  - Very very long state management (bloom filters)
function do_catch_release_drop(a: addr, msg: string) {
    do_direct_drop(a, msg);
    if (drop_info[a]$dropped_code != 0)
        return;

    local di = drop_info[a];

    local t = (persistent_offender_time != 0 sec &&
        di$tot_drop_time >= persistent_offender_time) ?
            long_drop_time : drop_time;

    di$tot_drop_time += t;
    di$last_timeout = t;

    schedule t { restore_dropped_address(a) };
}
<table>
<thead>
<tr>
<th>fields</th>
<th>ts</th>
<th>uid</th>
<th>host</th>
<th>action</th>
<th>reason</th>
<th>dropped_code</th>
<th>port_scanned</th>
<th>tot_drop_count</th>
<th>tot_restore_count</th>
<th>actual_restore_count</th>
<th>tot_dropped_count</th>
<th>interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>tot drop count</td>
<td>172800.000000</td>
<td>86400.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 18 00:00:01</td>
<td>2</td>
<td>1</td>
<td>172800.000000</td>
<td>86400.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Connection logs

<table>
<thead>
<tr>
<th>Time</th>
<th>IP Address</th>
<th>Port</th>
<th>Source Port</th>
<th>Source Address</th>
<th>Source Country</th>
<th>Destination Port</th>
<th>Destination Address</th>
<th>Destination Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug 17 02:13:08</td>
<td>C82agl398B0a0Y41c7</td>
<td>1.50.224.173</td>
<td>49093</td>
<td>131.243.149.71</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:13:20</td>
<td>CVrrtC14QaqpanQN2i2</td>
<td>1.50.224.173</td>
<td>49093</td>
<td>131.243.149.71</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:21:45</td>
<td>CagMsg30rS1tYe1tt7</td>
<td>1.50.224.173</td>
<td>38614</td>
<td>128.3.98.112</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:21:54</td>
<td>CjLDGv4DYhQc7SDSwf</td>
<td>1.50.224.173</td>
<td>38614</td>
<td>128.3.98.112</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:22:06</td>
<td>CGn23p4M5PKLPK5Rlf</td>
<td>1.50.224.173</td>
<td>38614</td>
<td>128.3.98.112</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:25:08</td>
<td>CpBpS11us0h1bjZib</td>
<td>1.50.224.173</td>
<td>46418</td>
<td>128.3.141.211</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:25:17</td>
<td>ChAoJa2kYV094ez7Qb</td>
<td>1.50.224.173</td>
<td>46418</td>
<td>128.3.141.211</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:25:29</td>
<td>CC7RMY19KRSDOXCKNb</td>
<td>1.50.224.173</td>
<td>46418</td>
<td>128.3.141.211</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:43:22</td>
<td>C7XTPT27qT9SB2fsza</td>
<td>1.50.224.173</td>
<td>56343</td>
<td>128.3.216.250</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aug 17 02:43:31</td>
<td>CmPzRU3hu0cG0gHbp1</td>
<td>1.50.224.173</td>
<td>56343</td>
<td>128.3.216.250</td>
<td>23</td>
<td>tcp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice Log

Aug 17 02:43:27 - - - - - - - - - - tcp
OldScan::AddressScan 1.50.224.173 has scanned □ hosts (23/tcp) [Drop::NoticeNone]
□ 1.50.224.173 □ 23 □ □ worker-1 Notice::ACTION_DROP
,Notice::ACTION_LOG 3600.000000 T - -

ACLD Log

Aug 17 02:43:27 NETS status=succeed cmd=nulterno id=wired device=XXXX
ip=1.50.224.173 ats=1408268607.940360 cts=1408268607.940412 cmt={bro@"}
no=OldScan::AddressScan msg=1.50.224.173 has scanned □ hosts 23/tcp

Accountability
HTTP Deep blocking

Blocks based on deeper visibility into Protocols

- redef HTTP::sensitive_URIs += /\blockme$/ ;
- Invoke event http_request

So if you go to cyber.lbl.gov/blockme , you will get blocked

- Some sensitive HTTP are “watch” worthy instead of “block worthy”
  - So HTTP::sensitive_URIs = NOTICE_DROP
  - HTTP::watched_URIs = NOTICE_EMAIL
HTTP Deep-blocks

```python
if ( n$note == HTTP::HTTP_SensitiveURI && n$id$orig_h !in LBNL::scan_hosts && n$id$orig_h !in Site::local_nets ) 
  add n$actions [Notice::ACTION_DROP];
if ( n$note == HTTP::HTTP_WatchedURI && n$id$orig_h !in LBNL::scan_hosts && n$id$orig_h !in Site::local_nets ) 
  add n$actions [Notice::ACTION_EMAIL];
if ( n$note == HTTP::HTTP_PageURI && n$id$orig_h !in LBNL::scan_hosts && n$id$orig_h !in Site::local_nets ) 
  add n$actions [Notice::ACTION_PAGE];
```

Aug 18 14:04:30 C9KBDh1GHaQwnFLCh6 72.36.117.14 28683 128.3.41.32 80 - -
  - tcp HTTP::HTTP_SensitiveURI GET http://128.3.41.32/blockme [Drop::NoticeNone]
  - 72.36.117.14 128.3.41.32 80 - worker-1 Notice::ACTION_LOG,
Notice::ACTION_DROP 3600.000000 T - - - - - - GET http://128.3.41.32/blockme -

Aug 18 14:04:30 acld: NETS status=success cmd=nullzero id=wired device=XXXXXXXXX ip=72.36.117.14
  ats=1408395870.753132 cts=1408395870.753194 cmt={bro@}\n  no=HTTP::HTTP_SensitiveURI msg=GET http://128.3.41.32/blockme
Application specific blocks - Heartbleed

Apr 8 18:10:02 acld: NETS status=success cmd=nullzero id=wired device=ip=213.37.136.14 ats=1397005802.785484 cts=1397005802.785538 cmt=SSL heartbleed scanner

Apr 9 01:58:17 acld: NETS status=success cmd=nullzero id=wired device=ip=107.3.148.68 ats=1397033897.178419 cts=1397033897.178482 cmt={bro@} no=Heartbleed::SSL_Heartbeat_Attack msg=An TLS heartbleed attack was detected! Record length 3, payload length 16384

Aug 17 20:07:42 acld: NETS status=success cmd=nullzero id=wired device=ip=195.159.233.44 ats=1408331262.393718 cts=1408331262.393783 cmt={bro@} no=Heartbleed::SSL_Heartbeat_Attack msg=Heartbeat before ciphertext. Probable attack or scan. Length: 96, is_orig: 1

Aug 18 22:29:09 acld: NETS status=success cmd=nullzero id=wired device=ip=176.58.123.151 ats=1408426149.497628 cts=1408426149.497682 cmt={bro} no=NTP::NTP_Monlist_Queries msg=NTP monlist queries

Aug 18 22:29:25 acld: NETS status=ignore cmd=nullzero id=wired device=ip=176.58.123.151 ats=1408426165.680533 cts=1408426165.680594 cmt={bro} no=0ldScan::AddressScan msg=176.58.123.151 has scanned [ ] hosts 123/udp 38484681 monlistqueries

```c
event ntp_message(c: connection, msg: ntp_msg, excess: string) {
    if ( ..... ((msg$code != NTP_CONTROL) || (msg$code != NTP_PRIVATE)) )
        return;

    if (((msg$code == NTPPRIVATE) || (msg$code == NTP_CONTROL)) && msg$precision == MON_GETLIST_1) {
        if ( c$id$orig_h !in ntp_host ) {
            NOTICE([$note=NTP::NTP_Monlist_Queries,
                    $conn=c,
```

Blacklist and intel feeds

- Hits are often irrelevant
- Count the number of hits to get a better insight into the activity
Wish-list continues

- Persistence - restarts should not matter
- Ability to add new heuristics very fast
- Identify and Remove false positives quickly and suppress them in future
- Should Percolate false positives up quickly ( .gov, .edu, etc)
IP_BAN

- Block connections form list of known Bad IPs
- What if the list has a very high churn rate?
- Bulk block and then bulk unblock
  - continuously
- Rationale
  - Block the IP only if it connects to you
  - No point blocking an IP and then Unblocking it after its removed from the list
- Can we do this **independently** while other DMZ boxes are doing their ‘thing’
Extents of blocking - some numbers
ACL Rate Graph

Day ending August 17, 2014 @ 3:05AM

Week ending August 17, 2014 @ 3:05AM
ACLD rate graph for Month of August 2014

Month ending August 17, 2014 @ 3:05AM

ROUTE/min

Thu 24 Jul  Thu 31 Jul  Thu 7 Aug  Thu 14 Aug

NULLZERO
Challenges

- Atomicity in blocking
- Coexistence with other blocking services
- Race conditions
Unblocking failure condition

Found an instance of brute forcers are slipping through the cracks

- SEC Log analyzer would apply a block and maintain state that block is applied
- Bro-A would apply a block, fails but thinks block is in place for next N hours
- Bro-B existing timer would expire and would unblock.
- SEC continues to thinks block is in place so no more blocking (to control block rate)
- Bro-A keep thinking blocking is in place so no more blocking

- Bad guys get a free whack for few hours
● Race conditions on blocking and unblocking
  o Bro boxes should share state?
  o One Bro is authoritative until it fails?
  o How does this work with other tools - SEC etc?
● and for unblocking?
  o who is authoritative?
  o How much do we rely on whitelist due to Additional overheads to maintain manually
Solution to avoid race conditions on blocking and unblocking

Well, use ACLD return code !!

- success block = 0
- failed block = 1
- already blocked = 2

Now based on this return code, bro decides to act for unblocking

```c
function do_restore(a: addr, force: bool)
{
    if ( a !in drop_info )
        return;

    local di = drop_info[a];

    if (di$dropped_code == ACL_EXISTING)
    {
        debug_log(fmt("Dropped_by_else, not restoring: %s, %s", a, di));
        return;
    }

    if (di$dropped_code == ACL_FAILURE)
    {
        debug_log(fmt("FAILED ACL when attempted drop, not restoring: %s, %s", a, di));
        return;
    }
```
when ( local result = Exec::run([{$cmd=drop_command}]) )
{
    if ( result$exit_code == ACL_EXISTING ) {
        ## check if previous drop was also dropped by this host
        if ( di$dropped_code == 0 ) {
            do_notice([{$note=Drop::AddressAlreadyDropped, $src=a, $msg=fmt("%s\s\s", a, msg)]});
        } else {
            di$dropped_code = ACL_EXISTING ;
            do_notice([{$note=Drop::AddressAlreadyDropped_By_Else, $src=a, $msg=fmt("%s\s\s", a, msg)]});
        }
    } else if ( result$exit_code == ACL_FAILURE ) {
        di$dropped_code = ACL_FAILURE ;
    } else if ( result$exit_code == ACL_SUCCESS ) {
        di$dropped_code = ACL_SUCCESS ;

        if ( di$tot_drop_count == 0 )
            do_notice([{$note=Drop::AddressDropped, $src=a, $msg=fmt("%s\s\s", a, msg)]});
        else {
            local s = fmt("(%d times)\", di$tot_drop_count + 1);
            do_notice([{$note=Drop::RepeatAddressDropped, $src=a,
                       $n=di$tot_drop_count+1, $msg=fmt("%s\s\s \s", a, msg, s), $sub=s}]);
        }

        ++di$tot_drop_count;
    }
} timeout 5secs {
    debug_log(fmt("Hm, didn't return:%s", drop_command));
}
Number of Uniq IPs dropped by DMZ bro boxes + overlap (if any)
Some things to keep in mind

- Bro respects blacklist IP and doesn’t unblocks
- Bro respects blocks put by other Bro boxes and doesn’t unblock
- Use of whitelists, local_nets, neighbor_nets
- Watch to account for functionality
  - alert if too many failures on blocking
  - alert if too many success on blocking
  - alert if rate of blocking changes etc
# Nightly Reports

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-08-18</td>
<td>2</td>
<td>24</td>
<td>10</td>
<td>21317 (10640)</td>
<td>31517 (15726)</td>
<td>118 (59)</td>
<td>256 (128)</td>
<td>1102</td>
<td>56</td>
</tr>
<tr>
<td>2014-08-17</td>
<td>0</td>
<td>12</td>
<td>14</td>
<td>33308 (16630)</td>
<td>46926 (23414)</td>
<td>142 (71)</td>
<td>250 (124)</td>
<td>708</td>
<td>34</td>
</tr>
<tr>
<td>2014-08-16</td>
<td>2</td>
<td>30</td>
<td>10</td>
<td>19766 (9669)</td>
<td>30705 (15292)</td>
<td>214 (107)</td>
<td>188 (94)</td>
<td>596</td>
<td>64</td>
</tr>
<tr>
<td>2014-08-15</td>
<td>6</td>
<td>20</td>
<td>12</td>
<td>20183 (10070)</td>
<td>40926 (20005)</td>
<td>370 (185)</td>
<td>274 (137)</td>
<td>758</td>
<td>62</td>
</tr>
<tr>
<td>2014-08-14</td>
<td>0</td>
<td>18</td>
<td>8</td>
<td>54785 (27382)</td>
<td>16250 (8125)</td>
<td>362 (181)</td>
<td>350 (175)</td>
<td>1138</td>
<td>50</td>
</tr>
<tr>
<td>2014-08-13</td>
<td>2</td>
<td>32</td>
<td>4</td>
<td>40456 (20217)</td>
<td>73455 (36697)</td>
<td>350 (175)</td>
<td>514 (257)</td>
<td>630</td>
<td>60</td>
</tr>
<tr>
<td>2014-08-12</td>
<td>2</td>
<td>40</td>
<td>8</td>
<td>37082 (18529)</td>
<td>62205 (31066)</td>
<td>340 (170)</td>
<td>232 (116)</td>
<td>632</td>
<td>76</td>
</tr>
<tr>
<td>2014-08-11</td>
<td>0</td>
<td>18</td>
<td>4</td>
<td>48916 (24448)</td>
<td>55861 (27914)</td>
<td>304 (152)</td>
<td>78 (39)</td>
<td>896</td>
<td>94</td>
</tr>
<tr>
<td>2014-08-10</td>
<td>0</td>
<td>14</td>
<td>4</td>
<td>17385 (8679)</td>
<td>12163 (6066)</td>
<td>326 (163)</td>
<td>86 (43)</td>
<td>596</td>
<td>108</td>
</tr>
<tr>
<td>2014-08-09</td>
<td>0</td>
<td>50</td>
<td>6</td>
<td>15964 (7965)</td>
<td>11756 (5858)</td>
<td>338 (169)</td>
<td>88 (44)</td>
<td>508</td>
<td>96</td>
</tr>
<tr>
<td>2014-08-08</td>
<td>0</td>
<td>22</td>
<td>4</td>
<td>9014 (4495)</td>
<td>10850 (5389)</td>
<td>314 (157)</td>
<td>84 (42)</td>
<td>484</td>
<td>156</td>
</tr>
<tr>
<td>2014-08-07</td>
<td>0</td>
<td>24</td>
<td>6</td>
<td>10664 (5326)</td>
<td>14986 (7464)</td>
<td>342 (171)</td>
<td>100 (50)</td>
<td>490</td>
<td>190</td>
</tr>
<tr>
<td>2014-08-06</td>
<td>0</td>
<td>16</td>
<td>4</td>
<td>12619 (6299)</td>
<td>15228 (7582)</td>
<td>408 (204)</td>
<td>120 (60)</td>
<td>544</td>
<td>108</td>
</tr>
<tr>
<td>2014-08-05</td>
<td>2</td>
<td>28</td>
<td>2</td>
<td>16452 (8219)</td>
<td>19854 (9906)</td>
<td>320 (160)</td>
<td>118 (59)</td>
<td>760</td>
<td>116</td>
</tr>
<tr>
<td>2014-08-04</td>
<td>0</td>
<td>28</td>
<td>6</td>
<td>21778 (10880)</td>
<td>30664 (15314)</td>
<td>312 (153)</td>
<td>136 (68)</td>
<td>708</td>
<td>136</td>
</tr>
</tbody>
</table>
Onestop :: LBLnet host block query page

Your public IP address is

IP or MAC address or HOSTNAME:

1.50.224.173  submit  clear

Report for 1.50.224.173 on Friday, August 15 2014 at 16:11:55 PDT

Wired:

  block: OK
  syslog:

  2014-08-17 02:43:27 nullzero 1.50.224.173 {bro@!} \"\" has scanned __ hosts 23/top \"\"

  whitelist: OK
  nullzero: YES
  port exceptions: NONE

Wireless:

  block: OK
  syslog: NONE
  whitelist: OK
  nullzero: OK

Site:

  blacklist: NO

Isolated/Denyboot:

  isolated: NO
  denyboot: NO

DHCP:

  lease: NO
connect.lbl.gov

Overview

The purpose of this site is to provide information to off-site Berkeley Lab users regarding the blocking of computers and instructions on unblocking.

How do I know if my computer or conference is blocked?

If you are having problems connecting to sites at Berkeley Lab (such as www.lbl.gov) but have no issues connecting to any other internet sites (such as www.google.com), it is likely that your machine or conference network is blocked.

Why am I blocked?

Berkeley Lab runs a sophisticated system to detect and block attacks to laboratory computers. Sometimes computers outside the lab are blocked for behaviors that appear to be attacks on the lab and it may be necessary for the Lab’s Security Team to whitelist the conference or outside machine’s IP address, which prevents it from being blocked.

How can I get unblocked?

Please acquire the public facing IP address of the conference network or blocked computer. The box below should show your current IP address:

Your current IP address:

192.17.162.97

Please send a request to the IT Help Desk by calling (510) 486-4357, or sending an email to help@lbl.gov and provide the following information:

- Your name
- Conference name and location (e.g. hotel or conference center)
- IP address (from the top of the page)
- Start and end date of the conference

If my computer is blocked, how can I see this page?

This page is hosted at Google (outside of the lab) in order to provide you the necessary information to get unblocked.
Stale management

- Strip out all IPs acld has operated on for past 7 days (cheap to do. also, exclude if either SUCCESS or IGNORE)

- This leaves the "stale candidates"
  - we iterate over each IP and check the logs back to "the beginning of time" for the last *successful* transaction …
  - Obtain the blocking-agent and block-reason … and these let you write different expire rule for "well-known cases" …

(This is relatively expensive, so we want to minimize how many IPs to lookups to-the-beginning-of-time).
Stale management

- blocking-agent = DMZ bro + block-reason = any -> AUTO_FREE: DMZ_ORPHANS

- block-agent = balh@bro + block-reason = SSH_BRUTEFORCE -> AUTO_FREE: SSH_BRUTEFORCE

This in turn leave "not-well understood cases" for human inspection "STALE_DIR" makes it trivial to add rules to do expires.

This has the advantage of the create() and free() processes do not need to be tightly coupled ... i.e. if you spin up a new process to say drop on ICMP timestamp scanning,

you don’t worry if you are ok with the default action (which in some cases is AUTO_FREE and in other escalate-to-eyeballs).
Never ending wish-list

1. Dynamic responses based on situation - eg.
   a. Change from ACL blocks to nullzero on thresholds on fly
   b. Expire blocks based on priorities (icmp sooner than ssh for example)
2. Prioritize a list of ports/IPs/nets to be aggressively blocked
3. Careful and slow in blocking a certain set
4. Mechanisms to handle established connection scanners/bruteforcers (RDP, SSH)
5. If possible figure out intentions why this scan specifically
6. Outsmart attackers over attackers so that they cannot easily guess/defeat block thresholds *(Dynamic thresholds)*
7. Smart Defenses against spoofing udp
8. Make bro tell me what attackers know after they scan us
Questions ?

asharma@lbl.gov