Network Intrusion Detection & Forensics

with Bro

Matthias Vallentin
vallentin@berkeley.edu

BERKE1337

March 3, 2016

Outline

Detection vs. Blocking

Intrusion Prevention

- Inline
- Critical

Intrusion Detection

- Passive
- Independent

Deployment Styles

Host-based

- Scope: single machine
- Example: anti-virus (AV), system monitors (e.g., OSSEC)
- ✓ Access to internal system state (memory, disk, processes)
- ✓ Easy to block attacks
- ✗ High management overhead for large fleet of machines
- ✗ Expensive analysis can decrease performance

Network-based

- Scope: entire network
- Example: Bro, Snort, Suricata
- ✓ Network-wide vantage-point
- ✓ Easy to manage, best bang for the buck
- ✗ Lack of visibility: tunneling, encryption (TLS)
- ✗ All eggs in one basket

Detection Terminology

	Alert	**No Alert**
Attack	True Positive (TP)	False Negative (FN)
No Attack	False Positive (FP)	True Negative (TN)

Detection Styles

Four main styles

1. Misuse detection
2. Anomaly detection
3. Specification-based detection
4. Behavioral detection

Misuse Detection

Goal

Detect **known** attacks via *signatures*/*pattern* or *black lists*

Pros

- ✓ Easy to understand, readily shareable
- ✓ FPs: management likes warm fuzzy feeling

Cons

- ✗ Polymorphism: unable to detect new attacks or variants
- ✗ Accuracy: finding sweetspot between FPs and FNs is *hard*

Example

Snort, regular expression matching

Anomaly Detection

Goal

Flag **deviations** from a known profile of “normal”

Pros

- ✓ Detect wide range of attacks
- ✓ Detect novel attacks

Cons

- ✗ High FP rate
- ✗ Efficacy depends on training data purity

Example

Look at distribution of characters in URLs, learn some are rare

Specification-Based Detection

Goal

Describe what constitutes allowed activity via *policy* or *white list*

Pros

- ✓ Can detect novel attacks
- ✓ Can have low FPs

Cons

- ✗ Expensive: requires significant development
- ✗ Churn: must be kept up to date

Example

Firewall

Behavioral Detection

Goal

Look for **evidence** of compromise, rather than the attack itself

Pros

- ✓ Works well when attack is hard to describe
- ✓ Finds novel attacks, cheap to detect, and low FPs

Cons

- ✗ Misses unsuccessful attempts
- ✗ Might be too late to take action

Example

```
unset $HISTFILE
```

Outline

Broverview

History

- Created by Vern Paxson, 1996
- Since then monitors the border of LBNL
- At the time, difficult to use, expert NIDS

Today

- *Much* easier to use than 10 years ago
- Established open-source project, backed by Free Software Consortium
- Widely used in industry and academia
- General-purpose tool for network analysis
  - "The scripting language for your network"
  - Supports all major detection styles
- Produces a wealth of actionable logs by default

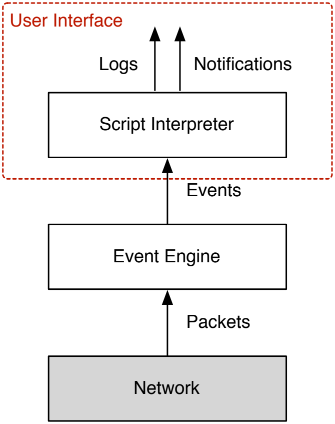
The Bro Network Security Monitor

Architecture

- Real-time network analysis framework
- Policy-neutral at the core
- Highly stateful

Key components

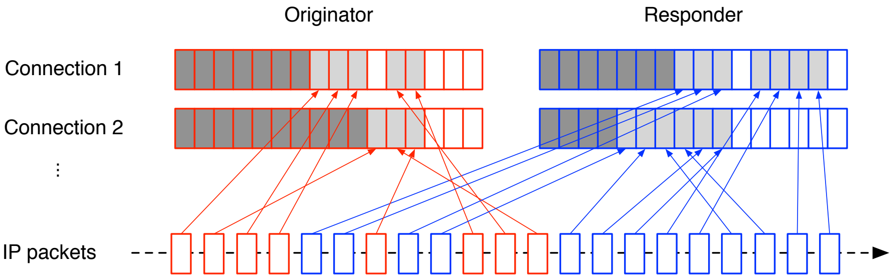
1. Event engine
   - TCP stream reassembly
   - Protocol analysis
   - Policy-neutral
2. Script interpreter
   - Construct & generate logs
   - Apply site policy
   - Raise alarms



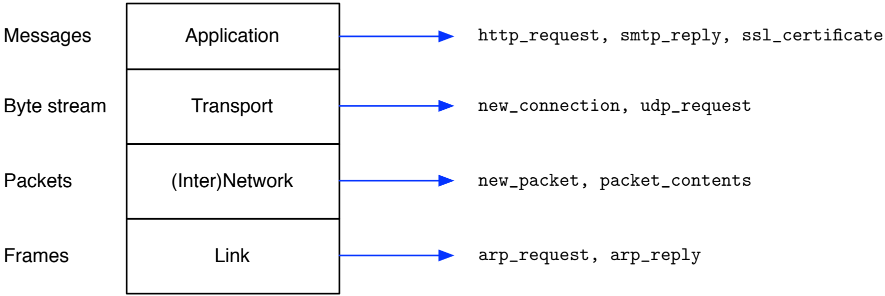
TCP Reassembly in Bro

Abstraction: from packets to byte streams

- Elevate packet data into byte streams
- Separate for connection originator and responder
- Passive TCP state machine: mimic endpoint semantics



Bro's Event Engine



Bro event and data model

- **Rich-typed**: first-class networking types (`addr`, `port`, ...)
- **Deep**: across the whole network stack
- **Fine-grained**: detailed protocol-level information
- **Expressive**: nested data with container types (aka. semi-structured)

Bro Logs

Events → Scripts → Logs

- **Policy-neutral** by default: no notion of good or bad
  - Forensic investigations highly benefit from *unbiased* information
  - Hence no use of the term “alert” → **NOTICE** instead
- **Flexible** output formats:
  1. ASCII
  2. Binary (coming soon)
  3. Custom

Log Example

conn.log

```
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2016-01-06-15-28-58
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto service duration orig_bytes resp_bytes conn_..
#types time string addr port addr port enum string interval count count string bool bool count string
1258531.. Cz7SRx3.. 192.168.1.102 68 192.168.1.1 67 udp dhcp 0.163820 301 300 SF - - 0 Dd 1 329 1 328 (empty)
1258531.. CTeURV1.. 192.168.1.103 137 192.168.1.255 137 udp dns 3.780125 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258531.. CUAVTq1.. 192.168.1.102 137 192.168.1.255 137 udp dns 3.748647 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258531.. CYoxAZ2.. 192.168.1.103 138 192.168.1.255 138 udp - 46.725380 560 0 S0 - - 0 D 3 644 0 0 (empty)
1258531.. CvabDq2.. 192.168.1.102 138 192.168.1.255 138 udp - 2.248589 348 0 S0 - - 0 D 2 404 0 0 (empty)
1258531.. CViJEOm.. 192.168.1.104 137 192.168.1.255 137 udp dns 3.748893 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258531.. CSC2Hd4.. 192.168.1.104 138 192.168.1.255 138 udp - 59.052898 549 0 S0 - - 0 D 3 633 0 0 (empty)
1258531.. Cd3RNm1.. 192.168.1.103 68 192.168.1.1 67 udp dhcp 0.044779 303 300 SF - - 0 Dd 1 331 1 328 (empty)
1258531.. CEwuIl2.. 192.168.1.102 138 192.168.1.255 138 udp - - - - S0 - - 0 D 1 229 0 0 (empty)
1258532.. CXxLc94.. 192.168.1.104 68 192.168.1.1 67 udp dhcp 0.002103 311 300 SF - - 0 Dd 1 339 1 328 (empty)
1258532.. CIFDQJV.. 192.168.1.102 1170 192.168.1.1 53 udp dns 0.068511 36 215 SF - - 0 Dd 1 64 1 243 (empty)
1258532.. CXFISh5.. 192.168.1.104 1174 192.168.1.1 53 udp dns 0.170962 36 215 SF - - 0 Dd 1 64 1 243 (empty)
1258532.. CQJw4C3.. 192.168.1.1 5353 224.0.0.251 5353 udp dns 0.100381 273 0 S0 - - 0 D 2 329 0 0 (empty)
1258532.. ClfEd43.. fe80::219:e3ff:fee7:5d23 5353 ff02::fb 5353 udp dns 0.100371 273 0 S0 - - 0 D 2 369 0 0
1258532.. C67zf02.. 192.168.1.103 137 192.168.1.255 137 udp dns 3.873818 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258532.. CG1FKF1.. 192.168.1.102 137 192.168.1.255 137 udp dns 3.748891 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258532.. CNFkeF2.. 192.168.1.103 138 192.168.1.255 138 udp - 2.257840 348 0 S0 - - 0 D 2 404 0 0 (empty)
1258532.. Cq4eis4.. 192.168.1.102 1173 192.168.1.1 53 udp dns 0.000267 33 497 SF - - 0 Dd 1 61 1 525 (empty)
1258532.. CHpqv31.. 192.168.1.102 138 192.168.1.255 138 udp - 2.248843 348 0 S0 - - 0 D 2 404 0 0 (empty)
1258532.. CFoJjT3.. 192.168.1.1 5353 224.0.0.251 5353 udp dns 0.099824 273 0 S0 - - 0 D 2 329 0 0 (empty)
1258532.. Cc3Ayyz.. fe80::219:e3ff:fee7:5d23 5353 ff02::fb 5353 udp dns 0.099813 273 0 S0 - - 0 D 2 369 0 0
```

Example: Matching URLs

Example

```
event http_request(c: connection, method: string, path: string) {
   if (method == "GET" && path == "/etc/passwd")
     NOTICE(SensitiveURL, c, path);
}
```

Example: Tracking SSH Hosts

Example

```
global ssh_hosts: set[addr];

event connection_established(c: connection) {
   local responder = c$id$resp_h; # Responder's address
   local service = c$id$resp_p;   # Responder's port

   if (service != 22/tcp)
       return; # Not SSH.

   if (responder in ssh_hosts)
       return; # We already know this one.

   add ssh_hosts[responder]; # Found a new host.
   print "New SSH host found", responder;
}
```

Example: Kaminsky Attack

1. Issue: vulnerable resolvers do not randomize DNS source ports
2. Identify relevant data: DNS, resolver address, UDP source port
3. Jot down your analysis ideas:
   - "For each resolver, no connection should reuse the same source port"
   - "For each resolver, connections should use random source ports"
4. Express analysis:
   - "Count the number of unique source ports per resolver"
5. Use your toolbox:
   - ```
     bro-cut id.resp_p id.orig_h id.orig_p < dns.log \
       | awk '$1 == 53 { print $2, $3 }' \ # Basic DNS only
       | sort | uniq -d  \ # Duplicate source ports
       | awk '{ print $1 }' | uniq # Extract unique hosts
     ```
6. Know your limitations:
   - No measure of PRNG quality (Diehard tests, Martin-Löf randomness)
   - Port reuse occurs eventually → false positives
7. Close the loop: write a Bro script that does the same
     ```

Example: Kaminsky Attack Detector

Example

```
const local_resolvers = { 7.7.7.7, 7.7.7.8 }
global ports: table[addr] of set[port] &create_expire=1hr;

event dns_request(c: connection, ...) {
    local resolver = c$id$orig_h; # Extract source IP address.
    if (resolver !in local_resolvers)
      return; # Do not consider user DNS requests.

    local src_port = c$id$orig_p; # Extract source port.
    if (src_port !in ports[resolver]) {
      add ports[resolver][src_port]:
      return;
    }

    # If we reach this point, we have a duplicate source port.
    NOTICE(...);
}
```

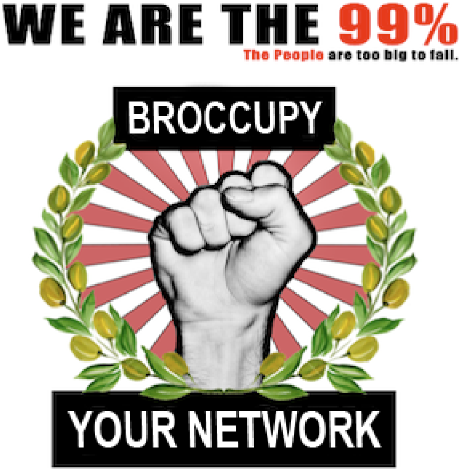
Outline

1. Intrusion Detection 101

2. Bro

3. Network Forensics Exercises

Your Turn!


WE ARE THE 99%
The People are too big to fail.
BROCCUPY
YOUR NETWORK


Bro

Ready, Set, Go!

Running Bro

Run Bro on the 2009-M57-day11-18 trace.

Solution

```
cd /tmp/berke1337
wget http://bit.ly/m57-trace
zcat 2009-M57-day11-18.trace.gz | bro -r -
```

Connection Statistics

Connection by duration

List the top-10 connections in decreasing order of duration, i.e., the longest connections at the beginning.

Solution

```
bro-cut duration id.{orig,resp}_{h,p} < conn.log | sort -rn |
```

Focus on a specific interval

How many connection exist with a duration between 1 and 2 minutes?

Solution

```
bro-cut duration id.{orig,resp}_{h,p} < conn.log \
  | awk '$1 >= 60 && $1 <= 120'
```

HTTP

HTTP servers

Find all IP addresses of web servers that send more than 1 KB back to a client.

Solution

```
bro-cut service resp_bytes id.resp_h < conn.log \
    | awk '$1 == "http" && $2 > 1000000 { print $3 }' \
    | sort -u
```

Non-standard HTTP servers

Are there any web servers on non-standard ports (i.e., 80 and 8080)?

Solution

```
bro-cut service id.resp_p id.resp_h < conn.log \
    | awk '$1=="http" && !($2==80 || $2==8080) { print $3 }' \
    | sort -u
```

Service Statistics

Service histogram

Show a breakdown of the number of connections by service.

Solution

```
bro-cut service < conn.log | sort | uniq -c | sort -n
```

Top destinations

Show the top 10 destination ports in descending order.

Solution

```
bro-cut id.resp_p < conn.log \
  | sort | uniq -c | sort -rn | head
```

Service Statistics (hard!)

Bulky hosts

What are the top 10 hosts (originators) that send the most traffic?

Solution

```
bro-cut id.orig_h orig_bytes < conn.log  \
    | sort                               \
    | awk '{ if (host != $1) {           \
                 if (size != 0)          \
                     print $1, size;     \
                  host=$1;               \
                  size=0                 \
              } else                     \
                  size += $2             \
            }                            \
            END {                        \
                if (size != 0)           \
                     print $1, size      \
                }'                       \
    | sort -k 2                          \
    | head
```

More HTTP Statistics

MIME types

- What are the distinct browsers in this trace?
- What are the distinct MIME types of the downloaded URLs?

Solution

```
bro-cut user_agent < http.log | sort -u
bro-cut mime_type < http.log | sort -u
```

Web sites

What are the three most commonly accessed web sites?

Solution

```
bro-cut host < http.log \
  | sort | uniq -c | sort -n | tail -n 3
```

HTTP Referral

Referer header

What are the top 10 referred hosts?

Solution

```
bro-cut referrer < http.log                                     \
    | awk 'sub(/[[:alpha:]]+:\/\//, "", $1)       \
           {                                                      \
               split($1, s, /\//);                           \
               print s[1]                                      \
           }'                                                     \
    | sort                                                        \
    | uniq -c                                                     \
    | sort -rn                                                    \
    | head
```

Think!

What do you want to know?

That's It!

FIN