Network Intrusion Detection & Forensics
with Bro

Matthias Vallentin
vallentin@berkeley.edu

BERKE1337

March 3, 2016
1. Intrusion Detection 101

2. Bro

3. Network Forensics Exercises
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

<table>
<thead>
<tr>
<th></th>
<th>Alert</th>
<th>No Alert</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attack</strong></td>
<td>True Positive (TP)</td>
<td>False Negative (FN)</td>
</tr>
<tr>
<td><strong>No Attack</strong></td>
<td>False Positive (FP)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look for <strong>evidence</strong> of compromise, rather than the attack itself</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pros</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Works well when attack is hard to describe</td>
</tr>
<tr>
<td>✓ Finds novel attacks, cheap to detect, and low FPs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ Misses unsuccessful attempts</td>
</tr>
<tr>
<td>✗ Might be too late to take action</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>unset $HISTFILE</td>
</tr>
</tbody>
</table>
Outline

1. Intrusion Detection 101

2. Bro

3. Network Forensics Exercises
## Bro Overview

### 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
### 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**

<table>
<thead>
<tr>
<th>Time</th>
<th>UID</th>
<th>Orig.H</th>
<th>Orig.P</th>
<th>Resp.H</th>
<th>Resp.P</th>
<th>Proto</th>
<th>Service</th>
<th>Duration</th>
<th>Orig.Bytes</th>
<th>Resp.Bytes</th>
<th>Conn...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>192.168.1.102</td>
<td>68</td>
<td>192.168.1.1</td>
<td>67</td>
<td>udp</td>
<td>dhcp</td>
<td>0.163820</td>
<td>301</td>
<td>300</td>
<td>SF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.103</td>
<td>137</td>
<td>192.168.1.255</td>
<td>137</td>
<td>udp</td>
<td>dns</td>
<td>3.780125</td>
<td>350</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.102</td>
<td>137</td>
<td>192.168.1.255</td>
<td>137</td>
<td>udp</td>
<td>dns</td>
<td>3.748647</td>
<td>350</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.103</td>
<td>138</td>
<td>192.168.1.255</td>
<td>138</td>
<td>udp</td>
<td>-</td>
<td>46.725380</td>
<td>560</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.102</td>
<td>138</td>
<td>192.168.1.255</td>
<td>138</td>
<td>udp</td>
<td>-</td>
<td>2.248589</td>
<td>348</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.104</td>
<td>137</td>
<td>192.168.1.255</td>
<td>137</td>
<td>udp</td>
<td>dns</td>
<td>3.748893</td>
<td>350</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.104</td>
<td>138</td>
<td>192.168.1.255</td>
<td>138</td>
<td>udp</td>
<td>-</td>
<td>59.052989</td>
<td>549</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.103</td>
<td>68</td>
<td>192.168.1.1</td>
<td>67</td>
<td>udp</td>
<td>dhcp</td>
<td>0.044779</td>
<td>303</td>
<td>300</td>
<td>SF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.102</td>
<td>138</td>
<td>192.168.1.255</td>
<td>138</td>
<td>udp</td>
<td>-</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.104</td>
<td>68</td>
<td>192.168.1.1</td>
<td>67</td>
<td>udp</td>
<td>dhcp</td>
<td>0.002103</td>
<td>311</td>
<td>300</td>
<td>SF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.102</td>
<td>1170</td>
<td>192.168.1.1</td>
<td>53</td>
<td>udp</td>
<td>dns</td>
<td>0.068511</td>
<td>36</td>
<td>215</td>
<td>SF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.104</td>
<td>1174</td>
<td>192.168.1.1</td>
<td>53</td>
<td>udp</td>
<td>dns</td>
<td>0.170962</td>
<td>36</td>
<td>215</td>
<td>SF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.1</td>
<td>5353</td>
<td>224.0.0.251</td>
<td>5353</td>
<td>udp</td>
<td>dns</td>
<td>0.100381</td>
<td>273</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fe80::219:e3ff:fee7:5d23</td>
<td>5353</td>
<td>ff02::fb</td>
<td>5353</td>
<td>udp</td>
<td>dns</td>
<td>0.100371</td>
<td>273</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.103</td>
<td>137</td>
<td>192.168.1.255</td>
<td>137</td>
<td>udp</td>
<td>dns</td>
<td>3.873818</td>
<td>350</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.102</td>
<td>137</td>
<td>192.168.1.255</td>
<td>137</td>
<td>udp</td>
<td>dns</td>
<td>3.748891</td>
<td>350</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.102</td>
<td>137</td>
<td>192.168.1.255</td>
<td>137</td>
<td>udp</td>
<td>dns</td>
<td>3.748891</td>
<td>350</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.1.1</td>
<td>5353</td>
<td>224.0.0.251</td>
<td>5353</td>
<td>udp</td>
<td>dns</td>
<td>0.099824</td>
<td>273</td>
<td>0</td>
<td>S0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fe80::219:e3ff:fee7:5d23</td>
<td>5353</td>
<td>ff02::fb</td>
<td>5353</td>
<td>udp</td>
<td>dns</td>
<td>0.099813</td>
<td>273</td>
<td>0</td>
<td>S0</td>
</tr>
</tbody>
</table>
Example: Matching URLs

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

Example

```plaintext

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

```cpp
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
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 connections exist with a duration between 1 and 2 minutes?

Solution
bro-cut duration id.{orig,resp}_\{h,p\} < conn.log \n| awk '$1 >= 60 && $1 <= 120'
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
```
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
}
  size += $2
}
END {
  if (size != 0)
    print $1, size
}'
| sort -k 2
| head
```
### More HTTP Statistics

<table>
<thead>
<tr>
<th>MIME types</th>
</tr>
</thead>
<tbody>
<tr>
<td>- What are the distinct browsers in this trace?</td>
</tr>
<tr>
<td>- What are the distinct MIME types of the downloaded URLs?</td>
</tr>
</tbody>
</table>

**Solution**

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

<table>
<thead>
<tr>
<th>Web sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are the three most commonly accessed web sites?</td>
</tr>
</tbody>
</table>

**Solution**

```bash
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

27 / 29
Think!

What do you want to know?
That’s It!

FIN