BRO IDS
Intrusion Detection

Andrew Pyles

William and Mary

September 15, 2010
Background: **Typical Scenario**
Background: Typical Scenario

![Diagram of network setup with internet, switch, app server, firewall, and spare server connected by lines indicating network traffic and new attack.]
What can we do to prevent this attack in real-time?
Bro introduction

- BRO language input
- TCP/UDP events
- Fixed filter
- Network abstraction
  Any network interface

Network abstraction
Any network interface

Internet

libpcap

Event Engine

Policy Script Interpreter

Network
libpcap

- Used in tcpdump (Lawrence Livermore labs)
- Filter 'compiled', sent to kernel packet filter
- Example: **port http or port ftp or port 1234**
- Can capture TCP packets with **SYN, FIN, RST**
- Snapshot length configurable: Bro captures everything
- Tcpdump captures 68 bytes default **why**?
libpcap (cont’d): headers

**IP Header**

- 4-bit version
- 4-bit header length
- 8-bit type of service (TOS)
- 16-bit total length (in bytes)
- 16-bit identification
- 0 D M F
- 13-bit fragment offset
- 8-bit time to live (TTL)
- 8-bit protocol
- 16-bit header checksum
- 32-bit source IP address
- 32-bit destination IP address
- options (if any)
- data

**UDP Header**

- 16-bit source port number
- 16-bit destination port number
- 16-bit UDP length
- 16-bit UDP checksum
- data (if any)

**TCP Header**

- 16-bit source port number
- 16-bit destination port number
- 32-bit sequence number
- 32-bit acknowledgment number
- 4-bit header length
- reserved (6 bits)
- 16-bit window size
- 16-bit TCP checksum
- 16-bit urgent pointer
- options (if any)
- data (if any)

Reprinted from TCP/IP Illustrated, Volume 1: The Protocols
by W. Richard Stevens.
Copyright © 1994 by Addison-Wesley Publishing Company, Inc.
TCP review

TCP (Transmission Control Protocol) provides a reliable end to end service that delivers packets over the Internet. Packets are delivered in sequence without loss or duplication.

- **Client Node**
- **Server Node**
- **Client Application**
- **Server Application**

Client initiates TCP connection:
- Client Application creates a Socket
- The socket is created in the Closed state
- Initial sequence number is set to 0
- Application wishes to communicate with a destination server using a TCP connection. The application opens a socket for the connection in active mode. In this mode, a TCP connection will be attempted with the server.
- Typically, the client will use a well known port number to communicate with the remote server. For example, HTTP uses port 80.

Active Open:
- Server Application creates a Socket
- Client Application creates Socket
- The socket is created in the Closed state
- Initial sequence number is set to 0
- Application wishes to communicate with a destination server using a TCP connection. The application opens a socket for the connection in active mode. In this mode, a TCP connection will be attempted with the server.
- Typically, the client will use a well known port number to communicate with the remote server. For example, HTTP uses port 80.
TCP review

Client initiated three way handshake to establish a TCP connection:

Client sets the SYN bit in the TCP header to request a TCP connection. The sequence number field is set to 0. Since the SYN bit is set, this sequence number is used as the initial sequence number.

Socket transitions to the SYN Sent state.

Client receives the SYN_ACK TCP segment.

Client now acknowledges the first segment, thus completing the three way handshake. The receive window is set to 5000. Ack sequence number is set to 101, this means that the next expected sequence number is 101.

At this point, the client assumes that the TCP connection has been established.

LEG: Short data transfer.
Structure

Network

libpcap

Event Engine

Policy Script Interpreter

TCP/UDP events

Fixed filter

Network abstraction
Any network interface

BRO language input

Internet

Network abstraction
Any network interface
TCP:

- Track SYN/FIN/RST bits in tcp header
- If SYN, connection_attempt==true
- If SYN + acknowledgment, connection_established==true
- If FIN, connection_finished==true
- If RST, connection_aborted==true
Event Engine: UDP

UDP:
- Simpler
- Track source, destination IP
- Track source port, destination port
- Example: IP: 1.2.3.4:5000 > 4.3.2.1:5001
- 4.3.2.1:5001 sends to 1.2.3.4:5000
Policy script interpreter

- What do we do with events?
- Interpret user input from BRO language
- Record output to disk syslog
- Take external action (external script)
Bro language

Atomic types:
- bool
- int
- count
- double
- string

Strings
- Different from C:
- Example: USER nice\0User root
- Note: \0 == NUL
Non traditional data types:

- time: **absolute time**
- interval: **time difference**
- port: **TCP / UDP** (distinct)
- port example: **80/tcp**
- addr: **ip address**
- hostname: **list of addr**
Bro language

Aggregate types:

- record: arbitrary collection of elements
- table: set of indices AND a yield type
- set: table type with no yield
- file: open files for reading and writing
- list: zero or more instances of a variable
- pattern: regular expression
Bro language

Operators
- C-like operators: ’’+/-*/%!/&&||?:
- assignment : ’=’

Variables
- Scoping: local, global
- const IRC = 6666/tcp, 6667/tcp, 6668/tcp; : set[port]
Bro language

example

snippets from solution:
@load scan  # generic scan detection mechanism
@load drop
redef Drop::can_drop_connectivity = T;
redef Drop::use_catch_release = T;
redef Drop::drop_connectivity_script = "drop.sh";
redef Drop::restore_connectivity_script = "restore.sh";
...

# If a port scan report and block
redef notice_policy += {
  [$pred(a: notice_info) =
    { return a$note == Scan::PortScan &&
      ! is_local_addr(a$src); },
  $result = NOTICE_DROP,
  $priority = 1],
};
Implementation

Design:

• C++
• Single threaded
• high performance REGEX library
• Timers : calendar queues : O(1)
Implementation

Interpret or compile?

- future plans: compiler
- Binpac : ( Binary Protocol compiler )
- bif : ( bro compiler )
Attacks on the monitor

Overload attack:

Too much traffic:
- **solution**: Throw hardware at it

Force event flood:
- **light weight** Events in bro
- policy script pre-knowledge

Elevate logging recording level:
- policy script pre-knowledge

Event: **net_stats_update**
Attacks on the monitor

Crash attack:

Attack:
- crash monitor
- intrusion

Defense:
- watchdog timer: packet processed within $T$ seconds?
- careful coding
- **failsafe**: if (crash) tcpdump
Attacks on the monitor

**Subterfuge attack:**
- Attack: Mislead monitor

**Embedded NUL string:**
- send: USER nice\0Attack string

**Fragmented IP datagrams**
- TCP byte stream integrity
- Extract payload from each TCP packet
- re-assemble in proper order
Attacks on the monitor: subterfuge

Subterfuge sample attack:

- **Attacker**
  - seq= 6 ... 9
  - ttl=20
  - USER
  - ttl=12
  - nice
  - 10 .. 13
  - ttl=20
  - root

- **Monitor**
  - (10 hops)

- **Victim**
  - (18 hops)
  - ttl expires
  - USER root
  - ?
  - USER nice
  - ?
  - USER root
  - ?
Attacks on the monitor: subterfuge

Subterfuge attack:

Defense:

- Observe retransmitted packets
- TCP: same data retransmit
  - Can we trust TCP?
- sound alarm if different
Application-specific processing

**plain text examples**
- Finger
- `ident`
- FTP
- Portmapper
- Telnet, Rlogin

**plain text protocols:**
- catch suspicious usernames
- track suspicious protocol behavior
Application-specific processing: FTP

Goals:

ftp-request event

- Track distinct FTP sessions
- Sensitive username check: root
- Sensitive files check: /etc/passwd
- Send alert if abnormalities present

Threats:

- sToR vs. STOR: case insensitivity
- FTP Bounce attack: active mode
- Multiple request pipelining
Application-specific processing

**current status:**

- 162 policy scripts available
- drop.bro: **suspicious behavior:** drop user
- bittorrent.bro: **track bittorrent traffic**
- bt-tracker.bro: **BT tracker traffic**
- ssh-stepping.bro: **track ssh sessions**
- synflood.bro: **track SYN flood**
Usage notes

- bro -i eth0 <policy script>

**drop.bro:**

if (condition)
- run user defined script
- catch and release
  - drop, wait time T, restore
- unconditional drop

**Sample drop script drop.sh:**

# log dropped connections
echo "droppping ip: $1 " » /var/log/drop.log
/sbin/iptables -I INPUT -s $i -j DROP

**Sample restore script restor.sh:**

# restore connections from ipaddress
/sbin/iptables -D INPUT -s $1 -j DROP
How do you monitor > 1gigE interface?

**Problem statement:**
- Resource limitation
- Vulnerable to Overload attack

**Potential solution:**
- Use a multiplexor
- Break **Big** pipe into manageable units
- Currently deployed at Ohio State
Thanks!

Questions?
Goals:

FTP-request event

- Track distinct FTP sessions
- Sensitive username check: root
- Sensitive files check: /etc/passwd
- Send alert if abnormalities present

Threats:

- sToR vs. STOR: case insensitivity
- FTP Bounce attack: active mode
- Multiple request pipelining