CS 552
Computer Security

Richard Martin
Aspects of Security

- **Confidentiality**: can a 3rd party see it?
- **Authentication**: Who am I talking to?
- **Non-repudiation**: can you claim you didn’t send it even if you really did?
- **Integrity**: was it altered before I got it?
- **Authorization**: Are you allowed to perform the action (method)?
- **Auditing**: what happened, when, by who?
Outline

• Basics:
  – Stack smashing
  – worms, viruses

• Papers:
  – Basic Security
  – Portscan detection
  – Denial of Service traceback
Stack Vulnerability

• Any program written without memory protection (i.e., pointers) + writable stacks

• Recall from architecture/compilers:
  – local variables in a function/procedure/method are kept on a stack
  – So is the return address of the program counter.

• “Stack smashing” Give bad data to a program that:
  – Writes executable code into the program somewhere (most often the stack)
  – corrupts return address on stack to jump to code
#include <stdio.h>
char shellcode[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\x
b0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\n
xd8\x40\xcd\x80\xe8\xd0\xc0\xf0\xff\xff\xff/bin/sh";
char large string[128];
int i;
long *long_ptr;
int main() {
    char buffer[96];
    long ptr = (long *)large_string;
    for (i=0; i<32; i++)
        (long_ptr+i) = (int)buffer;
    for (i=0; i<(int)strlen(shellcode); i++)
        large_string[i] = shellcode[i];
    strcpy(buffer, large string);
    return 0; }

Stack Smashing Attack
int doSomething(int variable1);
    int arg1, arg2;
    char nextCommand[MAX_COMMAND];
    char inputFromWorld[MAX_INPUT];
...
    newsock_fd = accept(sockfd);
...
    read(newsock_fd,inputFromWorld,MAX_INPUT);
    sscanf(inputFromWorld,"%d %s %d ",&arg1,
    nextCommand, &arg2);
Overview of Protocol Security

• TCP syn impersonation
• Routing attacks
  • Source routing
  • Protocols:
    – RIP/EGP
  • ICMP
• Services and Server programs
  • Finger
  • Email (SMTP, POP)
  • Name lookup (DNS)
  • File transfer (FTP)
  • Network management (SNMP)
  • Remote Boot (TFTP, BTOOP)
TCP syn impersonation

• Assume attacker has compromised a machine, X
• Goal:
  – X can impersonate machine T to server S
    • Even if X has not compromised T
    • Even if X can’t observe traffic of S->T
• Strategy:
  – Send S a fake message
  – Guess Sequence number of S->T
  – Send fake data, get S to believe packet is from S
TCP Syn attack

Normal Syn/Ack sequence

C→S: SYN(ISN_c)
S→C: SYN(ISNs), ACK(ISNc)
C→S: ACK(ISNs)
C→S: data

Attacker based

X→S: SYN(ISNx), SRC=T
S→T: SYN(ISNs), ACK(ISNx)
X→S: ACK(ISNs), SRC=T
X→S: ACK(ISNs), SRC=T, bad_data
TCP syn attacks

• Success of attack depends on 2 weaknesses:
  – Initial guess of sequence number ISNs
  – S doing something with the bad data
    • Higher level of authentication
Routing Attacks

• Insert bogus routing entries for
  – Entire network
  – Single host

• Re-route packets for target system
  – Allows packet-interposition
Practical Security

• What does it mean to be secure?
  – Definitions from slide
Retrospective 1989-2005

- Authentication at all levels
  - “end-to-end” principal applies?
- DDoS
  - Never considered
- Auditing importance
- End-to-end confidentiality
  - Ssh, HTTPS
- Worry about bad input
  - Buffer overruns, SQL attacks
- Human impact on security
  - Too hard/complicated-> no security
  - Phishing
- Economic factors
  - Spam
Portscanning Intro

• Port Scanning: Reconnaissance
  – Hackers will scan host/hosts for vulnerable ports as potential avenues of attack

• Not clearly defined
  – Scan sweeps
    • Connection to a few addresses, some fail?
  – Granularity
    • Separate sources as one scan?
  – Temporal
    • Over what timeframe should activity be tracked
  – Intent
    • Hard to differentiate between benign scans and scans with malicious intent
Prior Detection Techniques

• Malformed Packets
  – Packets used for “stealth scanning”

• Connections to ports/hosts per unit time
  – Checks whether a source hits more than X ports on Y hosts in Z time

• Failed connections
  – Malicious connections will have a higher ratio of failed connection attempts
Bro NIDS

- Current algorithm in use for years
- High efficiency
- Counts local connections from remote host
- Differentiates connections by service
- Sets threshold
- Blocks suspected malicious hosts
Flaws in Bro

• Skewed for little-used servers
  – Example: a private host that one worker remotely logs into from home
• Difficult to choose probabilities
• Difficult to determine never-accessed hosts
  – Needs data to determine appropriate parameters
Threshold Random Walk (TRW)

• Objectives for the new algorithm:
  – Require performance near Bro
  – High speed
  – Flag as scanner if no useful connection
  – Detect single remote hosts
Data Analysis

- Data analyzed from two sites, LBL and ICSI
  - Research laboratories with minimal firewalls
  - LBL: 6000 hosts, sparse host density
  - ICSI: 200 hosts, dense host density

<table>
<thead>
<tr>
<th></th>
<th>LBL</th>
<th>ICSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Total inbound connections</td>
<td>15,614,500</td>
<td>161,122</td>
</tr>
<tr>
<td>2. Size of local address space</td>
<td>131,836</td>
<td>512</td>
</tr>
<tr>
<td>3. Active hosts</td>
<td>5,906</td>
<td>217</td>
</tr>
<tr>
<td>4. Total unique remote hosts</td>
<td>190,928</td>
<td>29,528</td>
</tr>
<tr>
<td>5. Scanners detected by Bro</td>
<td>122</td>
<td>7</td>
</tr>
<tr>
<td>6. HTTP worms</td>
<td>37</td>
<td>69</td>
</tr>
<tr>
<td>7. other_bad</td>
<td>74,383</td>
<td>15</td>
</tr>
<tr>
<td>8. remainder</td>
<td>116,386</td>
<td>29,437</td>
</tr>
</tbody>
</table>
Separating Possible Scanners

- Which of remainder are likely, but undetected scanners?
  - Argument nearly circular
  - Show that there are properties plausibly used to distinguish likely scanners in the remainder
  - Use that as a ground truth to develop an algorithm against
Data Analysis (cont.)

• First model
  – Look at remainder hosts making failed connections
  – Compare all of remainder to known bad
  – Hope for two modes, where the failed connection mode resembles the known bad
  – No such modality exists
Data Analysis (cont.)

• Second model
  – Examine ratio of hosts with failed connections made to successful connections made
  – Known bad have a high percentage of failed connections
  – Conclusion: remainder hosts with <80% failure are potentially benign
  – Rest are suspect
Variables

\( Y_i \) Trial \( i \); 0=connection succeeded, 1=failed

\( \theta_0 \) Probability connection succeed given source is begin

\( \theta_1 \) Probability connection succeed given source is a scanner

\( \alpha \) Ideal false positive upper bound

\( \beta \) Ideal detector lower bound
• Detect failed/succeeded connections
• Sequential Hypothesis Testing
  – Two hypotheses: benign (H_0) and scanner (H_1)
  – Probabilities determined by the equations
  – \( \Theta_0 > \Theta_1 \) (benign has higher chance of succeeding connection)
  – Four outcomes: detection, false positive, false negative, nominal

\[
\begin{align*}
\Pr[Y_i = 0|H_0] &= \Theta_0, & \Pr[Y_i = 1|H_0] &= 1 - \Theta_0 \\
\Pr[Y_i = 0|H_1] &= \Theta_1, & \Pr[Y_i = 1|H_1] &= 1 - \Theta_1
\end{align*}
\]
Thresholds

• Choose Thresholds
  – Set upper and lower thresholds, \( n_0 \) and \( n_1 \)
  – Calculate likelihood ratio
  – Compare to thresholds

\[
\Lambda(Y) \equiv \frac{Pr[Y|H_1]}{Pr[Y|H_0]} = \prod_{i=1}^{n} \frac{Pr[Y_i|H_1]}{Pr[Y_i|H_0]}
\]
Figure 3. Flow diagram of the real-time detection algorithm.
Choosing Thresholds

• Choose two constants, alpha and beta
  – Probability of false positive (P_f) <= alpha
  – Detection probability (P_d) >= beta
  – Typical values: alpha = 0.01, beta = 0.99

• Thresholds can be defined in terms of P_f and P_d or alpha and beta
  – n_1 <= P_d/P_f
  – n_0 >= (1-P_d)/(1-P_f)
  – Can be approximated using alpha and beta
  – n_1 = beta/alpha
  – n_0 = (1-beta)/(1-alpha)
Evaluation Methodology

• Used the data from the two labs
• Knowledge of whether each connection is established, rejected, or unanswered
• Maintains 3 variables for each remote host
  – \( D_s \), the set of distinct hosts previously connected to
  – \( S_s \), the decision state (pending, \( H_0 \), or \( H_1 \))
  – \( L_s \), the likelihood ratio
Evaluation Methodology (cont.)

• For each line in dataset
  – Skip if not pending
  – Determine if connection is successful
  – Check whether is already in connection set; if so, proceed to next line
  – Update D_s and L_s
  – If L_s goes beyond either threshold, update state accordingly
## Results

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th>LBL</th>
<th>ICSI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Count</td>
<td>$P_D$</td>
<td>$\bar{N}$</td>
</tr>
<tr>
<td>scan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>122</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$H_1$</td>
<td>122</td>
<td>1.000</td>
<td>4.0</td>
</tr>
<tr>
<td>worm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>32</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$H_1$</td>
<td>27</td>
<td>0.844</td>
<td>4.5</td>
</tr>
<tr>
<td>PENDING</td>
<td>5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>other bad</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>13257</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$H_1$</td>
<td>13059</td>
<td>0.985</td>
<td>4.0</td>
</tr>
<tr>
<td>$H_0$</td>
<td>15</td>
<td>-</td>
<td>5.1</td>
</tr>
<tr>
<td>PENDING</td>
<td>183</td>
<td>-</td>
<td>11</td>
</tr>
<tr>
<td>benign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>2811</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$H_1$</td>
<td>33</td>
<td>-</td>
<td>8.1</td>
</tr>
<tr>
<td>$H_0$</td>
<td>2343</td>
<td>-</td>
<td>4.1</td>
</tr>
<tr>
<td>PENDING</td>
<td>435</td>
<td>-</td>
<td>14</td>
</tr>
<tr>
<td>suspect</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>692</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$H_1$</td>
<td>659</td>
<td>0.952</td>
<td>4.1</td>
</tr>
<tr>
<td>PENDING</td>
<td>33</td>
<td>-</td>
<td>7</td>
</tr>
</tbody>
</table>
TRW Evaluation

- Efficiency – true positives to rate of H1
- Effectiveness – true positives to all scanners
- N – Average number of hosts probed before detection

<table>
<thead>
<tr>
<th>Trace</th>
<th>Measures</th>
<th>TRW</th>
<th>Bro</th>
<th>Snort</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBL</td>
<td>Efficiency</td>
<td>0.963</td>
<td>1.000</td>
<td>0.615</td>
</tr>
<tr>
<td></td>
<td>Effectiveness</td>
<td>0.960</td>
<td>0.150</td>
<td>0.126</td>
</tr>
<tr>
<td></td>
<td>overline(N)</td>
<td>4.08</td>
<td>21.40</td>
<td>14.06</td>
</tr>
<tr>
<td>ICSI</td>
<td>Efficiency</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>Effectiveness</td>
<td>0.992</td>
<td>0.029</td>
<td>0.029</td>
</tr>
<tr>
<td></td>
<td>overline(N)</td>
<td>4.06</td>
<td>36.91</td>
<td>6.00</td>
</tr>
</tbody>
</table>

Table 8. Comparison of the efficiency and effectiveness across TRW, Bro, and Snort
TRW Evaluation (cont.)

- TRW is far more effective than the other two
- TRW is almost as efficient as Bro
- TRW detects scanners in far less time
Potential Improvements

- Leverage Additional Information
  - Factor for specific services (e.g. HTTP)
  - Distinguish between unanswered and rejected connections
  - Consider time local host has been inactive
  - Consider rate
  - Introduce correlations (e.g. 2 failed in a row worse than 1 fail, 1 success, 1 fail)
  - Devise a model on history of the hosts
Improvements (cont.)

- Managing State
  - Requires large amount of maintained states for tracking
  - However, capping the state is vulnerable to state overflow attacks
- How to Respond
  - What to do when a scanner is detected?
  - Is it worth blocking?
- Evasion and Gaming
  - Spoofed IPs
    - Institute “whitelists”
    - Use a honeypot to try to connect
  - Evasion (inserting legitimate connections in scan)
    - Incorporating other information, such as a model of what is normal for legitimate users and give less weight to connections not fitting the pattern
- Distributed Scans
  - Scans originating from more than one source
  - Difficult to fix in this framework
Summary

• TRW- based on ratio of failed/succeeded connections
• Sequential Hypothesis Testing
• Highly accurate
  – 4-5 vs 20 attempts on average. Meaningful?
• Quick Response
Modify routers to allow IP traceback

**Traceback problem**

**Goal**
- Given set of packets
- Determine path

**Assumptions**
- Most routers remain uncompromised
- Attacker sends many packets
- Route from attacker to victim remains relatively stable
Simple method

• Record path
  – Each router adds IP address to packet
  – Victim reads path from packet

• Problem
  – Requires space in packet
    • Path can be long
    • No extra fields in current IP format
      – Changes to packet format are not practical
Better idea

• Many packets
  – DDoS involves many packets on same path
• Store one link in each packet
  – Each router probabilistically stores own address
  – Fixed space regardless of path length
Edge Sampling

• Data fields
  – Edge: *start* and *end* IP addresses
  – Distance: number of hops since edge stored

• Marking procedure for router R
  with probability $p$
    write $R$ into start address
    write 0 into distance field
  else
    if distance ==0 write $R$ into end field
    increment distance field
Edge Sampling: picture

• Packet received
  – $R_1$ receives packet from source or another router
  – Packet contains space for start, end, distance
Edge Sampling: picture

• Begin writing edge
  – $R_1$ chooses to write start of edge
  – Sets distance to 0

![Diagram showing edge sampling process](attachment:image.png)
Edge Sampling

• Finish writing edge
  – $R_2$ chooses not to overwrite edge
  – Distance is 0
    • Write end of edge, increment distance to 1
Edge Sampling

• Increment distance
  – $R_3$ chooses not to overwrite edge
  – Distance $>0$
    • Increment distance to 2
Path reconstruction

• Extract identifiers from attack packets
• Build graph rooted at victim
  – Each (start,end,distance) tuple provides an edge
  – Eliminate edges with inconsistent distance
  – Traverse edges from root to find attack paths
• # packets needed to reconstruct path

\[ E(X) < \frac{\ln(d)}{p(1-p)^{d-1}} \]

where \( p \) is marking probability, \( d \) is length of path

Optimal \( p \) is \( 1/d \) ... can vary probability by distance
Node Sampling?

• Less data than edge sampling
  – Each router writes own address with probability $p$
• Infer order by number of packets
  – Router at distance $d$ has probability $p(1-p)^d$ of showing up in marked packet

• Problems
  – Need many packets to infer path order
  – Does not work well if many paths
Reduce Space Requirement

• XOR edge IP addresses
  – Store edge as start $\oplus$ end
  – Work backwards to get path:
    $(\text{start } \oplus \text{end}) \oplus \text{end} = \text{start}$

• Sample attack path

```
a \oplus b \quad b \oplus c \quad c \oplus d \quad d
```

Diagram:
```
 a  b  c  d  V
```

a $\rightarrow$ b $\rightarrow$ c $\rightarrow$ d $\rightarrow$ V
Creating Unique Edge-ids

- Edge-id fragments are not unique
  - with multiple attackers, multiple edge fragments with the same offset and distance
- Bit-interleave has code with IP address
Encoding Edge Fragments

offset  edge fragment
distance

2  5  8
IP Header Encoding

• Backwards compatibility
• Two problems
  – Writing same values into id fields of frags from different datagrams
  – Writing different values into id fields of frags of same datagrams
Fragmentation Issues

- Copy data into ICMP packet
- Check the checksum at higher level
- etc
Creating Unique Edge-ids

Address
0000...1111

Hash(Address)
0011...1100

Bit-interleave
00000101...11111010

send k fragments into network
Candidate Edge-ids

• Combine all permutations of fragments at each distance with disjoint offset values
• Check that the hash matches hash of the address
Details: where to store edge

- **Identification field**
  - Used for fragmentation
  - Fragmentation is rare
  - 16 bits
- **Store edge in 16 bits?**
  - Break into chunks
  - Store start and end

<table>
<thead>
<tr>
<th>offset</th>
<th>distance</th>
<th>edge chunk</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 3</td>
<td>7 8 15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Header Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Length</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flags</th>
<th>Fragment Offset</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Time to Live</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Header Checksum</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Source Address of Originating Host</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Destination Address of Target Host</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Padding</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IP Data</th>
</tr>
</thead>
</table>
Edge Sampling (Cont)

- The expected number of packets needed for the victim to reconstruct the entire path is at most \(\ln(d)/p(1-p)^{d-1}\)
  - Example: \(p=0.1, d=10\), reconstruction requires about 75 packets
  - This is related to the coupon-collection problem

- Edge sampling allows reconstruction of the whole attack tree

- Encoding start, end, and distance is a problem
  - Not backward compatible if we change the IP header!
  - There are ways around this
Digression: Coupon Collection

- Suppose you have $t$ types of coupons, $C_1, C_2, \ldots, C_t$
  - Each time you open baseball cards, you get a coupon of type $i$ with probability $1/t$
  - How many coupons do you need before you have a complete set?
    - Note that in real competitions, all types are not $1/t$
    - Call total number you need $N$ (a random variable)
  - Define a random variable $N_i$ indicating the number of draws you need to use when you hold $i-1$ coupon types and you want a new type
    - Then $N = N_1 + N_2 + \ldots + N_t$
Coupon Collection (cont)

- Then $E(N) = E(N_1) + \ldots + E(N_t)$
  - Linearity of expectation
- What is $E(N_i)$?
  - Probability of getting a new type if you have $i-1$ types is $(N-i+1)/N$, so expectation is $N/(N-i+1)$
    - For geometric random variables, expectation is the inverse of the parameter
    - If you have a fair die, it takes an expected 6 rolls to get a 4 (for example)
  - So $E(N) = 1 + N/(N-2) + N/(N-3) + \ldots + N$ or $N H_N$
    - Here $H_N$ is the Nth harmonic number
    - This is approximately $N \ln N$. 
Testing the Algorithm

• Uses Simulation
  – Create random paths
  – Originate attacks
• Mark P() = 1/25
• 1,000 random test runs
• Vary path lengths
Results
Advanced Marking Schemes for IP Traceback

- Approach by Savage et. al. is impractical when > 25 attackers

- Use Internet maps and alternative encoding techniques to reconstruct multiple attack paths with high probability

- Victims can either maintain a current map of the routers leading to them or they can create such maps after the attack for analysis.
- The edges are represented as the current router information XOR the information of an upstream router.
- To minimize the number of collisions and hence reduce the number of false positives due to many routers ending up at the same distance, a set of hashing functions is used.
Solution continued…

- A set of hash functions of size $2^w$ is used.
- A router with probability $q$ inserts its own information hashed with random hash function $h$, whose index $x$ is $w$ bits. Also distance is set to 0. Otherwise the router increments the distance and XORs the result of $h(x, \text{current IP})$ with the edge information in the packet.
- When the victim performs reconstruction it will perform a breadth first search and then match the decoded packets with the routers it is aware of in the tree. To reduce the number of false positives a $m$-threshold scheme can be used which requires a router to be identified as an attacker $m$ times.
- The reconstruction scheme can identify more than 2000 attacker sites with very few false positives and runs in $O(\sum_d |S_d| \cdot |E_{d+1}|)$ vs. Savage’s $O(\sum_d |S_d| \cdot |E_{d+1}|^8)$ ($S_d$ set of routers on the attack path at distance $d$, $E_d$ unique edges at distance $d$)
- Not all the packets have to be collected before the reconstruction can begin.
Solution continued…

- Routers can use time varying MACs (Message Authenticating Codes) to sign the packets. This will allow the victim to reconstruct the map by referencing the timestamp of the packet and the public site that contains the MAC keys for routers at different time periods.

**Analysis/Impact**

- The scheme does not require all of the packets to be received before the reconstruction can begin and a victim does not need the entire Internet map as long as most of the offending paths are on the map.
Results