CSE543 Computer and Network Security

Module: Internet Malware

Professor Trent Jaeger
Worms

• A worm is a self-propagating program.
• As relevant to this discussion
  1. Exploits some vulnerability on a target host …
  2. (often) embeds itself into a host …
  3. Searches for other vulnerable hosts …
  4. Goto (1)

• Q: Why do we care?
The Danger

• What makes worms so dangerous is that infection grows at an exponential rate

  ‣ A simple model:
    • \( s \) (search) is the time it takes to find vulnerable host
    • \( i \) (infect) is the time is take to infect a host

  ‣ Assume that \( t=0 \) is the worm outbreak, the number of hosts at \( t=j \) is

\[
2^{j/(s+i)}
\]

• For example, if \( s+i = 1 \), what is it at time \( t=32 \)?
The result

![Graph showing data points ranging from 0 to 5,000,000,000]
The Morris Worm

• Robert Morris, a 23 doctoral student from Cornell
  ‣ Wrote a small (99 line) program
  ‣ November 3rd, 1988
  ‣ Simply disabled the Internet

• How it did it
  ‣ Reads /etc/password, they tries the obvious choices and dictionary, /usr/dict words
  ‣ Used local /etc/hosts.equiv, .rhosts, .forward to identify hosts that are related
    • Tries cracked passwords at related hosts (if necessary)
    • Uses whatever services are available to compromise other hosts
  ‣ Scanned local interfaces for network information
  ‣ Covered its tracks (set is own process name to sh, prevented accurate cores, re-forked itself)
Code Red

• Exploited a Microsoft IIS web-server vulnerability
  ‣ A vanilla buffer overflow (allows adversary to run code)
  ‣ Scans for vulnerabilities over random IP addresses
  ‣ Sometimes would deface the served website
• July 16th, 2001 - outbreak
  ‣ CRv1 - contained bad randomness (fixed IPs searched)
  ‣ CRv2 - fixed the randomness,
    • added DDOS of www.whitehouse.gov
    • Turned itself off and on (on 1st and 19th of month, attack 20-27th, dormant 28-31st)
  ‣ August 4 - Code Red II
    • Different code base, same exploit
    • Added local scanning (biased randomness to local IPs)
    • Killed itself in October of 2001
Worms and infection

• The effectiveness of a worm is determined by how good it is at identifying vulnerable machines
  ‣ Morris used local information at the host
  ‣ Code Red used what?

• Multi-vector worms use lots of ways to infect
  ‣ E.g., network, DFS partitions, email, drive by downloads …
  ‣ Another worm, Nimda did this

• Lots of scanning strategies
  ‣ Signpost scanning (using local information, e.g., Morris)
  ‣ Random IP - good, but waste a lot of time scanning “dark” or unreachable addresses (e.g., Code Red)
  ‣ Local scanning - biased randomness
  ‣ Permutation scanning - instance is given part of IP space
Other scanning strategies

• The doomsday worm: a flash worm
  ‣ Create a hit list of all vulnerable hosts
    • Staniford et al. argue this is feasible
    • Would contain a 48MB list
  ‣ Do the infect and split approach
  ‣ Use a zero-day vulnerability

• Result: saturate the Internet is less than 30 seconds!
Worms: Defense Strategies

- (Auto) patch your systems: most, if not all, large worm outbreaks have exploited known vulnerabilities (with patches)
- Heterogeneity: use more than one vendor for your networks
- Shield (Ross): provides filtering for known vulnerabilities, such that they are protected immediately (analog to virus scanning)

- Filtering: look for unnecessary or unusual communication patterns, then drop them on the floor
  - This is the dominant method, getting sophisticated (Arbor Networks)
Denial of Service

• Intentional prevention of access to valued resource
  ‣ CPU, memory, disk (system resources)
  ‣ DNS, print queues, NIS (services)
  ‣ Web server, database, media server (applications)

• This is an attack on availability (fidelity)

• Note: launching DOS attacks is easy

• Note: preventing DOS attacks is hard
  ‣ Mitigation the path most frequently traveled
Canonical DOS - Request Flood

• Attack: request flooding
  ‣ Overwhelm some resource with legitimate requests
  ‣ e.g., web-server, phone system
Example: SMURF Attacks

- This is one of the deadliest and simplest of the DOS attacks (called a *naturally amplified* attack)
  - Send a large number PING packet networks on the broadcast IP addresses (e.g., 192.168.27.254)
  - Set the source packet IP address to be your victim
  - All hosts will reflexively respond to the ping at your victim
  - … and it will be crushed under the load.
  - Fraggle: UDP based SMURF
Distributed denial of service

• DDOS: Network oriented attacks aimed at preventing access to network, host or service
  ‣ Saturate the target’s network with traffic
  ‣ Consume all network resources (e.g., SYN)
  ‣ Overload a service with requests
    • Use “expensive” requests (e.g., “sign this data”)  
  ‣ Can be extremely costly (e.g, Amazon)

• Result: service/host/network is unavailable

• Frequently distributed via other attack

• Note: IP is often hidden (spoofed)
D/DOS (generalized by Mirkovic)

• Send a stream of packets/requests/whatever …
  ‣ many PINGS, HTML requests, ...

• Send a few malformed packets
  ‣ causing failures or expensive error handling
  ‣ low-rate packet dropping (TCP congestion control)
  ‣ “ping of death”

• Abuse legitimate access
  ‣ Compromise service/host
  ‣ Use its legitimate access rights to consume the rights for domain (e.g., local network)
  ‣ E.g., First-year graduate student runs a recursive file operation on root of NFS partition
The canonical DDOS attack

(master) → Internet → (zombies) → (router) → LAN → (target)

(adversary) → Internet → (zombies)
The canonical DDOS attack
The canonical DDOS attack

Internet

(master)

(zombies)

(adversary)

(router)

(LAN)

(target)
The canonical DDOS attack

Internet

(master)

(zombies)

(router)

(adversary)

(LAN)

(target)
The canonical DDOS attack

(master)  
(adversary)  
(zombies)  
(router)  
(LAN)  
(target)
The canonical DDOS attack

(master) → (zombies) → (router) → (target)

Internet

(adversary)
The canonical DDOS attack
The canonical DDOS attack

Internet

(master)

(adversary)

(zombies)

(router)

(LAN)

(target)
DDOS and the E2E argument

• E2E (a simplified version): We should design the network such that all the intelligence is at the edges.
  ‣ So that the network can be more robust and scalable
  ‣ Many think is the main reason why the Internet works

• Downside:
  ‣ Also, no real ability to police the traffic/content
  ‣ So, many security solutions break this E2E by cracking open packets (e.g., application level firewalls)
  ‣ DDOS is real because of this …
DOS Prevention - Reverse-Turing Tests

- **Turing test**: measures whether a human can tell the difference between a human or computer (AI)
- **Reverse Turning tests**: measures whether a user on the internet is a person, a bot, whatever?
- CAPTCHA - completely automated public Turing test to tell computers and humans apart
  - contorted image humans can read, computers can’t
  - image processing pressing SOA, making these harder
- Note: often used not just for DOS prevention, but for protecting “free” services (email accounts)
DOS Prevention - Puzzles

• Make the solver present evidence of “work” done
  ‣ If work is proven, then process request
  ‣ Note: only useful if request processing significantly more work than

• Puzzle design
  ‣ Must be hard to solve
  ‣ Easy to Verify

• Canonical Example
  ‣ Puzzle: given all but k-bits of r and h(r), where h is a cryptographic hash function
  ‣ Solution: Invert h(r)
  ‣ Q: Assume you are given all but 20 bits, how hard would it be to solve the puzzle?
Pushback

• Initially, detect the DDOS
  ‣ Use local algorithm, ID-esque processing
  ‣ Flag the sources/types/links of DDOS traffic

• Pushback on upstream routers
  ‣ Contact upstream routers using PB protocol
  ‣ Indicate some filtering rules (based on observed flows)

• Repeat as necessary towards sources
  ‣ Eventually, all (enough) sources will be filtered

• Q: What is the limitation here?
Pushback

• Initially, detect the DDOS
  ‣ Use local algorithm, ID-esque processing
  ‣ Flag the sources/types/links of DDOS traffic

• Pushback on upstream routers
  ‣ Contact upstream routers using PB protocol
  ‣ Indicate some filtering rules (based on observed flows)

• Repeat as necessary towards sources
  ‣ Eventually, all (enough) sources will be filtered

• Q: What is the limitation here?
Traceback

• Routers forward packet data to source
  ‣ Include packets and previous hop …
  ‣ At low frequency (1/20,000) …

• Targets reconstruct path to source (IP unreliable)
  ‣ Use per-hop data to look at
  ‣ Statistics say that the path will be exposed

• Enact standard
  ‣ Add filters at routers along the path
DDOS Reality

• None of the “protocol oriented” solutions have really seen any adoption
  ‣ too many untrusting, ill-informed, mutually suspicious parties must play together well (*hint*: human nature)
  ‣ solution have many remaining challenges

• Real Solution
  ‣ Large ISP police their ingress/egress points very carefully
  ‣ Watch for DDOS attacks and filter appropriately
    • e.g., BGP (routing) tricks, blacklisting, whitelisting
  ‣ Products in existing net that coordinate view from many points in the network to identify upswings in traffic ...
  ‣ Interestingly, this is the same way they deal with *worms* ...
Botnet Story
Botnets

- A botnet is a network of software robots (bots) run on zombie machines which run are controlled by command and control networks
  - IRCbots - command and control over IRC
  - Bot herder - owner/controller of network
  - "scrumping" - stealing resources from a computer

- Surprising Factoid: the IRC server is exposed.
What are botnets being used for?

Activities we have seen

**Stealing CD Keys:**

```plaintext
ying!ying@ying.2.tha.yang PRIVMSG #atta :BGR|0981901486 $getcdkeys
BGR|0981901486!nmavmkmymam@212.91.170.57 PRIVMSG #atta :Microsoft Windows
Product ID CD Key: (55274-648-5295662-23992).
BGR|0981901486!nmavmkmymam@212.91.170.57 PRIVMSG #atta :[CDKEYS]: Search completed.
```

**Reading a user's clipboard:**

```plaintext
B][]!Guardian@globalop.xxx.xxx PRIVMSG ##chem## :~getclip
Ch3m|784318!~zbhibvn@xxx-7CCCB7AA.click-network.com PRIVMSG ##chem## :-
[Clipboard Data]- Ch3m|784318!~zbhibvn@xxx-7CCCB7AA.click-network.com PRIVMSG ##chem## :If You think the refs screwed the seahawks over put your name down!!!
```

**DDoS someone:**

```plaintext
devil!evil@admin.of.hell.network.us PRIVMSG #t3rr0r0Fc1a :!pfflood 82.147.217.39
443 1500 s7n|2K503827!s7s@221.216.120.120 PRIVMSG #t3rr0r0Fc1a :\002Packets\002
\002D\002one \002;\002>\n s7n|2K503827!s7s@221.216.120.120 PRIVMSG #t3rr0r0Fc1a flooding....\n```

**Set up a web-server (presumably for phishing):**

```plaintext
[DeXTeR]!alexo@185-130-136-193.broadband.actcom.net.il PRIVMSG [Del]29466
:.http 7564 c:\ [Del]38628!zaazbob@born113.athome233.wau.nl PRIVMSG _[DeXTeR]
:[HTTPD]: Server listening on IP: 10.0.2.100:7564, Directory: c:\.
```
IRC botnets

- An army of compromised hosts ("bots") coordinated via a command and control center (C&C). The perpetrator is usually called a "botmaster".

"A botnet is comparable to compulsory military service for windows boxes"

-- Bjorn Stromberg
Bots usually require some form of **authentication** from their botmaster.
Lots of bots out there

• Level of botnet threat is supported by the conjecture that large numbers of bots are available to inflict damage

• Press Quotes
  ‣ “Three suspects in a Dutch crime ring hacked 1.5 million computers worldwide, setting up a “zombie network””, Associated Press
  ‣ “The bot networks that Symantec discovers run anywhere from 40 systems to 400,000”, Symantec
Measuring botnet size

- Two main categories
  - **Indirect** methods: inferring botnet size by exploiting the side-effects of botnet activity (e.g., DNS requests)
  - **Direct** methods: exploiting internal information from monitoring botnet activity
Indirect Methods

• Mechanism
  ‣ DNS blacklists
  ‣ DNS snooping

• What does it provide?
  ‣ DNS footprint

• Caveats
  ‣ DNS footprint is only a lower bound of the actual infection footprint of the botnet
  ‣ DNS records with small TTLs
  ‣ DNS servers blocking external requests (~50%)
DNS Blacklist

• The value of a bot is related to its status on the DNS blacklists
  ‣ Compromised hosts often used as SMTP servers for sending spam.
  ‣ DNS blacklists are lists maintained by providers that indicate that SPAM has been received by them.
  ‣ Organizations review blacklists before allowing mail from a host.

• A "clean" bot (not listed) is worth a lot

• A listed bot is largely blocked from sending SPAM
• **Observation**: bot controllers/users need to query for BL status of hosts to determine value.

• **Idea**: if you watch who is querying (and you can tell the difference from legitimate queries), then you know something is a bot

• Understanding the in/out ratio:

\[ \lambda_n = \frac{d_{n,\text{out}}}{d_{n,\text{in}}} \]

- \( d_{n,\text{out}} \): #queries by host
- \( d_{n,\text{in}} \): #queries for host

• **Q**: what does a high ratio mean? Low?
Results
Direct Methods

• Mechanisms
  ‣ Infiltrate botnets and directly count online bots
  ‣ DNS redirection (by Dagon et al.)

• What do they provide?
  ‣ Infection footprint & effective size (infiltration)
  ‣ Infection footprint (DNS redirection)

• Caveats
  ‣ Cloning (infiltration)
  ‣ Counting IDs vs. counting IPs (infiltration)
  ‣ Measuring membership in DNS sinkhole (DNS redirection)
  ‣ Botmasters block broadcasts on C&C channel (infiltration) (~48%)
• DNS redirection “sinkhole”
  ‣ Identify, then self poison DNS entries of IRC servers
• DNS cache hits
  ‣ Idea: query for IRC server to see if in cache
  ‣ If yes, at least one bot in the network within the TTL (see [14])
  ‣ Limitations: TTL, not all servers answer, lower bound on bots
Botnet size, what does it mean?

- **Infection Footprint**: the total number of infected bots throughout a botnet’s lifetime
  - Relevance: how wide spread the botnet infection

- **Effective Botnet Size**: the number of bots simultaneously connected to the command and control channel
  - Relevance: the botnet capacity to execute botmaster commands (e.g., flood attacks)

- An Example:
  - While a botnet appeared to have a footprint of 45,000 bots, the number of online bots (i.e. its effective size) was < 3,000
Botnet footprint estimates

- Redirection results:
  - Botnets with up to 350,000 infected hosts [Dagon et al.]
Making a Bot

• How are bots being created?
  ‣ What kinds of attacks can be used?
Buffer Overflow

```c
_start:
call main

int main() {
    char buf[8];
    gets(buf);
    printf("You typed: %s", buf);
}
```

EIP

0x0804321  call main

0x0804480  gets(buf);

0x0804484  printf("You typed: %s", buf);

0x0804484  }
Buffer Overflow

```c
_start:
    call main

int main() {
    char buf[8];
    gets(buf);
    printf("You typed: \%s", buf);
}
```

ESP:

```
0xbf000000
0xbf000004
0xbf000008
0xbf00000b
...
```

EIP:

```
0x80484321
0x8048480
0x804484
```

Return Address

```c
char buf[8];
gets(buf);
printf("You typed: %s", buf);
```
Buffer Overflow

```
_int main() {
    char buf[8];
    gets(buf);
    printf("You typed: %s", buf);
}
```

```
_start:
    call main

    int main() {
        char buf[8];
        gets(buf);
        printf("You typed: %s", buf);
    }
```

```
0xbf000000 0x80484321
0xbf000004
0xbf000008
0xbf00000b

_esp:
```

0x0804480 gets(buf); 0x12345678 0x90abcdef 0xbf000004

printf("You typed: %s", buf); 
```

```
EIP
```

```
ESP
```
Buffer Overflow

```c
int main() {
    char buf[8];
    gets(buf);
    printf("You typed: %s", buf);
}
```

EIP: 0x80484321

ESP:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xbf000000</td>
<td>0x80484321</td>
</tr>
<tr>
<td>0xbf000004</td>
<td></td>
</tr>
<tr>
<td>0xbf000008</td>
<td>0x12345678</td>
</tr>
<tr>
<td>0xbf00000b</td>
<td></td>
</tr>
</tbody>
</table>

Return Address: 0x80484321
Buffer Overflow

```c
_start:
    call main

int main() {
    char buf[8];
    gets(buf);
    printf("You typed: %s", buf);
}
```

Return Address

```c
0xb0000000 0x80484321 0x90abcdef 0xb0000004
```

buf

```c
char buf[8];
```

EIP

```c
0x804480
```

ESP

```c
0xb0000000 0xb0000004 0xb0000008 0xb000000b
```

...
Buffer Overflow

```c
_start:
  call main

int main() {
  char buf[8];
  gets(buf);
  printf("You typed: \%s", buf);
}
```

EIP

```
0x80484321 call main
```

ESP

```
0xbf000000
0xbf000004
0xbf000008
0xbf00000b
```

Return Address

```
0x80484321 0x90abcdef 0xbf000004
```

buf

```
0x12345678
```

Saturday, November 3, 12
Buffer Overflow

```c
_start:
    call main

t main() {
    char buf[8];
    gets(buf);
    printf("You typed: %s", buf);
}
```

Return Address

- buf
- 0x80484321
- 0x804480
- 0x804484
- 0xbf000004
- 0x12345678 0x90abcdef 0xbf000004

EIP

ESP

Saturday, November 3, 12
Buffer Overflow

```c
_int main() {
    char buf[8];
    gets(buf);
    printf("You typed: %s", buf);
}
```

```c
_EIP
  call main

 ESP
    gets(buf);
    printf("You typed: %s", buf);
    buf
      0x90abcdef
      0x12345678
```
Buffer Overflow

_start:
0x0804321
    call main

int main() {
    char buf[8];
    gets(buf);
    printf("You typed: %s", buf);
}

0x12345678 0x90abcdef 0xbf000004

Buffer Overflow Prevention

• Canaries
  ‣ Integrity of return address protection

• Randomize Stack Base
  ‣ Difficult to plant code on stack and run it

• W xor X
  ‣ Can only write or execute memory
  ‣ Prevent execution on stack

• How would you circumvent W xor X?
  ‣ Return-oriented programming
Return-Oriented Programming

```c
_int main() {
    char buf[8];
    gets(buf);
    printf("You typed: \%s", buf);
}
```

EIP: 0x804321

ESP:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xbf000000</td>
<td>0xbf000004</td>
</tr>
<tr>
<td>0xbf000004</td>
<td>0x90abcdef</td>
</tr>
<tr>
<td>0xbf000008</td>
<td>0x12345678</td>
</tr>
</tbody>
</table>

Return Address: function in libc

buf

Return Address: function in libc

...
Return-Oriented Programming

• General approach for hijacking control flow
• First
  ‣ Attacker gains control of the process (stack pointer)
  ‣ Any way possible...
• Second,
  ‣ Attacker then chooses the code to execute (creates a stack)
  ‣ Keeps control of the process

• Enables Turing-complete programs to be executed from existing code
  ‣ Requirement: there is enough code
Return-Oriented Programming

- Use ESP as program counter
  - E.g., Store 5 at address 0x8048000
  - without introducing new code

```asm
pop %eax
ret
pop %ebx
ret
movl %eax, (%ebx)
ret
```

- Registers
  - %eax =
  - %ebx =

- Stack
  - G1
  - 5
  - jmp G2
  - 0x8048000
  - jump G3
  - ...

- Memory
  - 0x8048000 =

- Code
  - G1
    - pop %eax
      - ret
    - pop %ebx
      - ret
    - movl %eax, (%ebx)
      - ret
Return-Oriented Programming

- Use ESP as program counter
  - E.g., Store 5 at address 0x8048000
  - without introducing new code

Code

```
pop %eax
ret

pop %ebx
ret

movl %eax, (%ebx)
ret
```

Stack

```
G1
5
jmp G2
0x8048000
jump G3
...
```

Return Address

buf

Registers

- %eax = 5
- %ebx =

Memory

- 0x8048000 =
Return-Oriented Programming

- Use ESP as program counter
  - E.g., Store 5 at address 0x8048000
  - without introducing new code

```
G3
movl %eax, (%ebx)
ret
```

```
pop %eax
ret
pop %ebx
ret
```

```
G1
5
jmp G2
```

```
0x8048000
jump G3
... 
```

- Registers
  - %eax = 5
  - %ebx = 0x8048000

- Memory
  - 0x8048000 =

- Code

- Stack

- Return Address

- buf
Return-Oriented Programming

- Use ESP as program counter
  - E.g., Store 5 at address 0x8048000
  - without introducing new code

```
Code
pop %eax
ret

pop %ebx
ret

movl %eax, (%ebx)
ret

Stack
G1
5
jmp G2
0x8048000
jump G3
...

Code
pop %eax
ret

pop %ebx
ret

movl %eax, (%ebx)
ret

G3

Stack
Return Address
buf

Registers
%eax = 5
%ebx = 0x8048000

Memory
0x8048000 = 5
```
Return-Oriented Programming

• Myths debunked
  ‣ (1) Don’t need bad code for bad behavior
    • No code injection required
  ‣ (2) Return-oriented attacks are limited
    • Can run any code in the program
    • Small sequences of instructions can be found to perform useful operations (gadgets)
    • Result is Turing-complete
  ‣ (3) Instruction pointer determines process control flow
    • Can execute an entire program using stack pointer to choose instruction sequence

• Claim: If attacker can get control of ESP, they can run anything they want
ROP Summary

• All is not lost

• Some defenses can limit ROP execution
  ‣ Only some legal control flow changes in a program
    • Match call and return sites (Control Flow Integrity)
  ‣ Eliminate choice of gadgets
    • Remove return instructions from program
  ‣ Make it difficult to find gadgets
    • Randomize locations (Address Space Randomization)
    • Randomize instructions (Instruction Translation)

• Result: Arms race continues