Announcements

- Lab 2 is due in 1 week. (You should have already started!)

- Next paper summary will be due next Thursday.
Viruses, Worms, & rootkits

Computer attacks have become a reality in the modern world.
Business & individuals lose millions to bugs that are preventable.
But what is really involved?
### Definitions

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virus</td>
<td>replicates itself into other executable code; when the infected code is executed, the virus also executes</td>
</tr>
<tr>
<td>worm</td>
<td>runs independently; propagate a complete working version of itself onto other hosts.</td>
</tr>
<tr>
<td>logic bomb</td>
<td>inserted into software by an attacker; lies dormant until some condition is met.</td>
</tr>
<tr>
<td>Trojan horse</td>
<td>appears to have a useful function, but also has a hidden, malicious function.</td>
</tr>
<tr>
<td>backdoor</td>
<td>any mechanism that bypasses a normal security check; sometimes intentional, sometimes accidental and exploited.</td>
</tr>
<tr>
<td>auto-rooter</td>
<td>a set of scripts used to break into machines remotely.</td>
</tr>
<tr>
<td>rootkit</td>
<td>a set of scripts used maintain admin access on a machine.</td>
</tr>
</tbody>
</table>

Taken From "Computer Security" by Stallings and Brown, Chapter 7, Table 7.1
Reality

Writing any of these requires a lot of system programming knowledge!

(Don't be a script kiddie!)
**Virus**

- Generally embedded in another piece of code (often compressed to avoid detection)

Many ways to classify them:

<table>
<thead>
<tr>
<th>classification</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot sector infector</td>
<td>infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.</td>
</tr>
<tr>
<td>file infector</td>
<td>infects files that the operating system or shell consider to be executable.</td>
</tr>
<tr>
<td>macro virus</td>
<td>infects files with macro code that is interpreted by an application.</td>
</tr>
</tbody>
</table>
Viruses generally hide somehow:

<table>
<thead>
<tr>
<th>classification</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encrypted virus</td>
<td>a portion of the virus creates an encryption key and encrypts the remainder of the virus; because the virus is encrypted, it's difficult to scan.</td>
</tr>
<tr>
<td>stealth virus</td>
<td>explicitly hides itself from detection; the entire virus is hidden.</td>
</tr>
<tr>
<td>polymorphic virus</td>
<td>mutates with every infection, making signature scan difficult.</td>
</tr>
<tr>
<td>metamorphic virus</td>
<td>rewrites itself with every infection, making signature scan difficult.</td>
</tr>
</tbody>
</table>
Worms

In contrast, worms generally run independently and propagate themselves via network connections.

Today, we'll examine one particular worm which exploits buffer overflow vulnerabilities.

In general, worms often exploit some vulnerability in a program that is already present on the system.
Early Example: Sendmail

- UNIX mail server. Early versions had 2 features

1. An outside SMTP connection could place the program into DEBUG mode

2. In DEBUG mode, a user could execute the "run command" feature, which sent an email to a program. The program would use the contents of the email as input.
Another: Finger

A UNIX command which provides user information:

```
$ finger oster
Login: oster Name: Osterberg
Directory: /Users/oster Shell: /
bin/bash
Office: Peter
Never logged in.
No Mail.
No Plan
```
Problem: Finger uses the function gets() from the C library.

Problem: Takes input from command prompt without a maximum length. In C, "strings" are arrays with fixed size!
What is bounds checking?

Consider this C program:

```c
void hello(char *taq)
{
    char inp[16],
    printf("Enter value for %s: ", tag);
    scanf(inp);
    printf("Hello your %s is %s
", taq, inp);
}
```

What does it do?

Just prompts for input & then prints it.
```c
void hello(char *taq) {
    char inp[16];
    printf("Enter value for %s: ", taq);
    gets(inp);
    printf("Hello your %s is %s\n", taq, inp);
}
```

```
$ cc -g -o buffer2 buffer2.c
$ ./buffer2
Enter value for name: Bill and Lawrie
Hello your name is Bill and Lawrie
buffer2 done
$ ./buffer2
Enter value for name:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Segmentation fault (core dumped)
$ perl -e 'print pack("H*",
        "4124344546474851525354555657586162636465666768"
        "08f9f91948304080a4e4e4e0a")'; ./buffer2
Enter value for name:
Hello your Re?py|uEA is ABCDEFGHQRSTUWXabcdefquyu
Enter value for Kyuu:
Hello your Kyuu is NNNN
Segmentation fault (core dumped)
```

Taken From "Computer Security" by Stallings and Brown, Figure 11.5
What is happening?

Recall: Programming languages store variables + state information on a stack.

```c
void hello(char *tag)
{
    char inp[16];
    printf("Enter value for %s: ", tag);
    gets(inp);
    printf("Hello your %s is %s\n", tag, inp);
}
```
No bounds checking in `gets()`:

```c
void hello(char *tag)
{
    char inp[16];
    printf("Enter value for %s: ", tag);
    gets(inp);
    printf("Hello your %s is %s\n", tag, inp);
}
```
The result:

when gets() attempts to return to main, the program control goes to the address 0x62616469

Segmentation fault!
Smashing the stack

If we are really clever, can have the program execute something malicious in the stack.
Back to Finger

If someone carefully crafted the input to finger, they could smash the stack and execute something else

\[ \rightarrow \text{say, 'sh'} \] (command prompt)

Example: The Morris Worm, 1988
Worm Propagation

Generally, a worm will:

- Find other systems to infect by looking at host tables
- Connect to other systems
- Copy itself over a invoke that copy
Morris Worm

Supposedly, this worm was intended to be a harmless tool to measure the internet at the time.

Instead, in just 3 days, it took out approximately 10% of all networked computers.

It used several known exploits, and contained a few bugs.

(99 lines of code!)
The worm tries 4 different attacks

Figure adapted from "With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988" by M. Eichin and J. Rochlis, 1989.
Step 4: Initialization: The worm "hides" itself

```c
strcpy(argv[0], "sh");

struct rlimit rl;
rl.rlim_cur = 0;
rl.rlim_max = 0;
if (setrlimit(RLIMIT_CORE, &rl))
 ;
```
Scan network interface

- Interface Table
  - if_init()
- Routing Table
  - rt_init()
- /.rhosts
- /etc/hosts.equiv
  - cracksome()

Host List

- hit_finger()
- hit_SMTP

waithit()
③ look up routing table
it does so using a call to `netstat`

```c
pipe = popen("/usr/ucb/netstat -r -n", "r");
```

Example output from my machine:
en220-m14560:: crenshaw$ netstat -r -n
Routing tables

<table>
<thead>
<tr>
<th>Internet</th>
<th>Destination</th>
<th>Gateway</th>
<th>Flags</th>
<th>Refs</th>
<th>Use</th>
<th>Netif</th>
<th>Expire</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>10.11.15.1</td>
<td>link#5</td>
<td>USC</td>
<td>25</td>
<td>0</td>
<td>en0</td>
<td>1191</td>
</tr>
<tr>
<td>10.11.15.1</td>
<td>0:0:c7:0c:0</td>
<td></td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
</tr>
<tr>
<td>10.11.15.14</td>
<td>0:26:40:c8:d2</td>
<td></td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
</tr>
<tr>
<td>10.11.15.15</td>
<td>0:11:86:ed:1d:b3</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>10.11.15.29</td>
<td>0:15:85:256:ee</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.31</td>
<td>0:25:bc:6f:ff:86</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>714</td>
<td></td>
</tr>
<tr>
<td>10.11.15.42</td>
<td>0:21:86:fb:8c:82</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>10.11.15.45</td>
<td>0:11:86:ed:1d:b3</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>416</td>
<td></td>
</tr>
<tr>
<td>10.11.15.47</td>
<td>0:21:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.57</td>
<td>0:21:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.67</td>
<td>0:11:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.73</td>
<td>0:11:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.75</td>
<td>0:11:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.76</td>
<td>0:11:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.81</td>
<td>0:11:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.87</td>
<td>0:11:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.100</td>
<td>0:21:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.108</td>
<td>0:21:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>10.11.15.122</td>
<td>0:21:86:fb:89:5b</td>
<td>UHLW</td>
<td>0</td>
<td>0</td>
<td>en0</td>
<td>611</td>
<td></td>
</tr>
</tbody>
</table>
④ Find likely targets
at this point, a list of potential targets has been created.
Attack!

Try finger + sendmail exploit on the list of possible hosts.

A piece of code (that is very portable) called li.o is then copied to the other host.
A program called `waithit()` then transfers the rest of the code over.

And it all begins again...
The other 2 attacks exploited common passwords, and vulnerabilities in `rsh/exec`.

All of these attacked only DEC VAX machines running BSD 4.3 or Sun.  

\[\text{Didn't work}\]
Morris Worm (Recap)

Remember, this was only supposed to measure the internet. What went wrong?

The code contained several bugs which turned it into the first large scale attack.
The worm was supposed to detect if it was already on a machine. This broke, so machines were reinfeected. This crashed any infected machine.

The worm was supposed to send a packet of data back to 128.32.137.13. This broke too.
The Result

“Robert T. Morris was convicted of violating the Computer Fraud and Abuse Act (Title 18), and sentenced to three years of probation, 400 hours of community service, a fine of $10,050, and the costs of his supervision.”

He’s now an assistant professor at MIT, conducting research in computer networks and operating systems.

(Amusingly, he was a Cornell grad student, but somehow originally released the worm at MIT.)
Why study this?

1988 was a long time ago.

If we no longer study the Apple IIe, why study this?

2 reasons I can think of:
Historical Relevance

The Morris Worm was the reason the US began setting up legislation to control computer crime.

This worm was the motivation behind founding CERT, the organization responsible for monitoring and tracking network emergencies.
Stack-Overflow Techniques
Still very relevant!

Debian Security Advisory

DSA-2124-1 xulrunner -- several vulnerabilities

Date Reported: 01 Nov 2010

- **CVE-2010-3179**

  Stack-based buffer overflow in the text-rendering functionality in Xulrunner allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption and application crash) via a long argument to the document.write method.
Next time: (future topics)

- Network defenses
- Constructing attacks (DETER)
- Secure coding & auditing
- New versus old techniques (as why things changed)
- Web applications
- Good practices (as admin or individual)