Goals for Today

- Learning Objective:
  - Explore how operating systems fail.
- Announcements, etc:
  - MP3 is out! Due **April 18th**.
  - Monday — Spectre & Meltdown presentation w/ Chris Fletcher!!
  - Wednesday — MP4 Walkthrough w/ Mohammad!!

Reminder: Please put away devices at the start of class

CS 423
Operating System Design: Epic Security Fails in Operating System History

Professor Adam Bates
Spring 2018

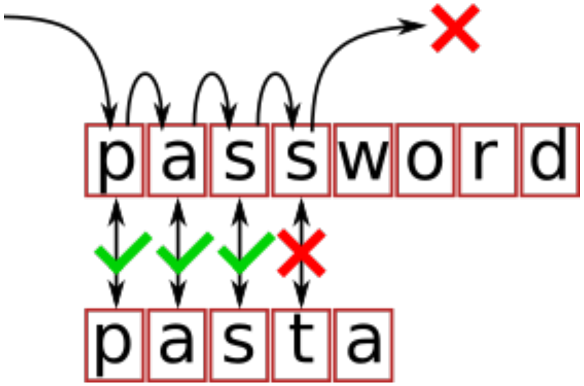
Security Practice

- In practice, systems are not that secure
  - hackers can go after weakest link
    - any system with bugs is vulnerable
  - vulnerability often not anticipated
    - usually not a brute force attack against encryption system
  - often can't tell if system is compromised
    - hackers can hide their tracks
  - can be hard to resecure systems after a breakin
    - hackers can leave unknown backdoors

Ex1: Tenex Password Vuln

- Early system supporting virtual memory
- Kernel login check:

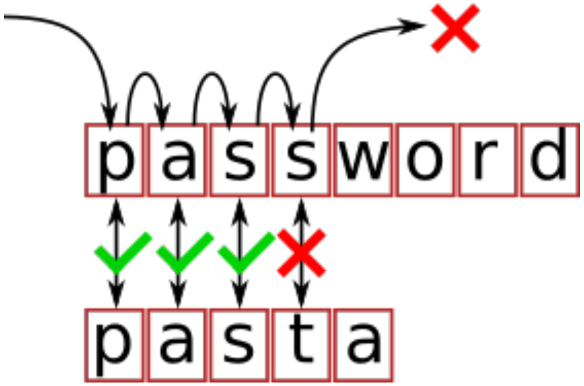
```
for (i = 0; i < password length; i++) {
   if (password[i] != userpwd[i]) return error;
}
return ok
```



Ex1: Tenex Password Vuln

- Early system supporting virtual memory
- Kernel login check:

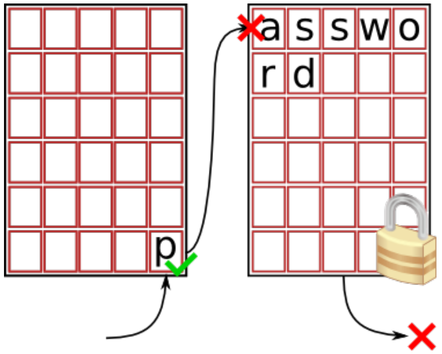
```
for (i = 0; i < password length; i++) {
   if (password[i] != userpwd[i]) return error;
}
return ok
```



ANY PROBLEMS HERE?

Ex1: Tenex Password Vuln

- Observation: Programs have *a lot* of control over how their virtual memory works.
- Attack #1: Trap-To-User Bit Exploit



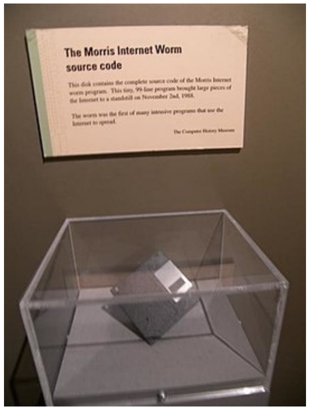
Trap-To-User: Alert me if this 2nd page is accessed!

- Attack #2: Exploit timing side-channel

Processing time for password check was proportional to the number of correct characters at the front of the attacker's guess.

Ex2: Morris Worm

- Used the Internet to infect a large number of machines in 1988
  - sendmail bug
    - default configuration allowed debug access
    - well known for several years, but not fixed
  - fingerd: finger adam@cs
    - fingerd allocated fixed size buffer on stack
    - copied string into buffer without checking length
    - encode virus into string!
  - password dictionary
- Used infected machines to find/infect others


The Morris Internet Worm
source code
This disk contains the complete source code of the Morris Internet worm program. This tiny, 99-line program brought large pieces of the Internet to a standstill on November 2nd, 1988.
The worm was the first of many intensive programs that use the Internet to spread.
The Computer History Museum

Ex3: Ping of Death

- IP packets can be fragmented, reordered in flight
- Reassembly at host
  - can get fragments out of order, so host allocates buffer to hold fragments
- Malformed IP fragment possible
  - offset + length > max packet size
  - Kernel implementation didn’t check
- Was used for denial of service, but could have been used for virus propagation

Ex4: UNIX Talk

- UNIX talk was an early version of Internet chat
  - For users logged onto same machine
- App was setuid root
  - Needed to write to everyone's terminal
- But it had a bug…
  - Signal handler for ctrl-C
  - Arbitrary code execution

Ex5: Netscape

- How do you pick a session key?
  - Early Netscape browser used time of day as seed to the random number generator
  - Made it easy to predict/break
- How do you download a patch?
  - Netscape offered patch to the random seed problem for download over Web, and from mirror sites
  - four byte change to executable to make it use attacker’s key

Ex6: Code Red Worm

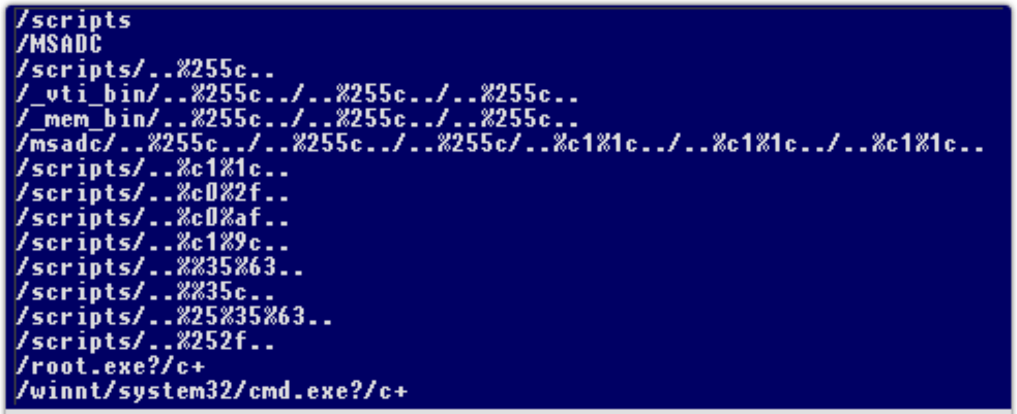
- Code Red: Exploited buffer overflow in IIS for which patch was available but largely unapplied:

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNN
%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3
%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a  HTTP/1.0
```

- Behaviors: Worked on a monthly schedule. Spread itself, defaced hosted websites, DoS'd IPs including whitehouse.gov.
- Developer of defense was invited to White House

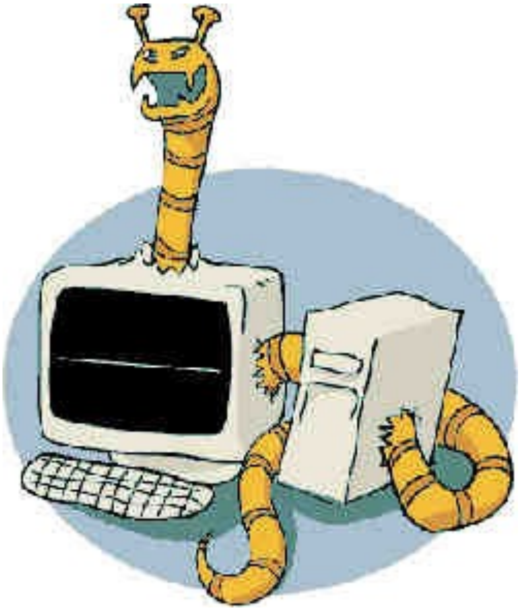
Ex7: Nimda Worm

- Utilized multiple attack vectors, ’Metasploit’-style.
- Email phising, network shares, compromised web sites, IIS Server vulns, and leftover Code Red backdoor
- Left open backdoor on infected machines for any use. Infected ~ 400K machines.



```
/scripts
/MSADC
/scripts/..%255c..
/_vti_bin/..%255c../..%255c../..%255c..
/_mem_bin/..%255c../..%255c../..%255c..
/msadc/..%255c../..%255c../..%255c/..%c1%1c../..%c1%1c../..%c1%1c..
/scripts/..%c1%1c..
/scripts/..%c0%2f..
/scripts/..%c0%af..
/scripts/..%c1%9c..
/scripts/..%%35%63..
/scripts/..%%35c..
/scripts/..%25%35%63..
/scripts/..%252f..
/root.exe?/c+
/winnt/system32/cmd.exe?/c+
```

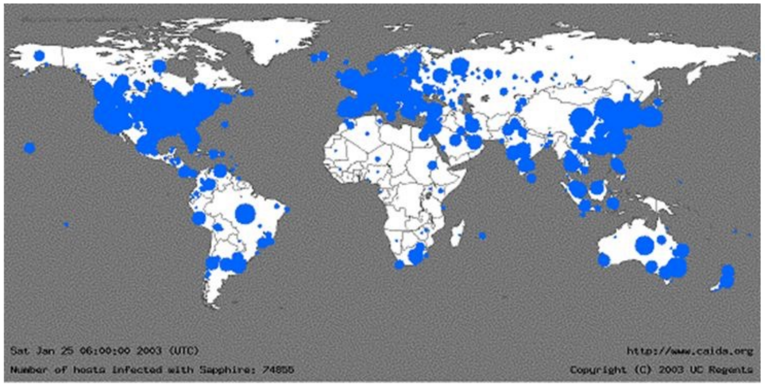
Directory traversal exploit strings in W32/Nimda-A



Ex8: SQL Slammer Worm

- Slammer: Single UDP packet on MySQL port. Infected 75K vulnerable machines in under 10 minutes

The SQL Slammer Worm:
30 Minutes After "Release"


Sat Jan 25 06:00:00 2003 (UTC)
Number of hosts infected with Sapphire: 74855
http://www.caida.org
Copyright (C) 2003 UC Regents

- Infections doubled every 8.5 seconds
- Spread 100X faster than Code Red
- At peak, scanned 55 million hosts per second.

- Today: Million node botnets now common!!

Reflections on Trusting Trust

- Ken Thompson’s self-replicating program
- Attempt 1: Add a malicious change to Unix’s login.c

(A)

```
if (name == “ken”) {
     don’t check password;
     login ken as root;
}
```

- … but this modification is too obvious. How do we hide it?

Reflections on Trusting Trust

- Ken Thompson’s self-replicating program
- Attempt 2: Add a malicious change to the C compiler
  - Insert into compiler:

(B)

```
if see trigger {
    insert (A) into the input stream
}
```

  - Add trigger to login.c

```
/* gobbledygook */
```

- Now we don’t need to include the code for the backdoor in login.c, just the trigger
- … but still too obvious; how do we hide the modification to the C compiler?

Reflections on Trusting Trust

- Ken Thompson’s self-replicating program
- Attempt 3: Hide the modification to the compiler

(C)

```
if see trigger2 {
    insert (B) and (C) into the input stream
}
```

- Compile the compiler with C present
- Change is now in the object code for compiler
- Replace (C) in the compiler source with /*trigger2*/

Reflections on Trusting Trust

- Ken Thompson's self-replicating program
- Now we have an invisible trojan horse in Version 1 of the C compiler…
    - *… but the compiler compiles the compiler on successive versions!!!*
    - As long as trigger2 is not removed, code for (B) and (C) will be present in future versions.
    - Making a compiler for a new machines? You're going to cross-compile first on the old machine using the old compiler!
- Result: Every new version of login.c has code for (A) included.
    - Invisible: No source code for the backdoor exists. Anywhere.

Reflections on Trusting Trust

- Thompson's Takeaway: You can't fully trust code that you didn't write yourself!
- Presented as a thought experiment during Thompson's Turing Award Lecture. Didn't really happen… we think??
- Hard to re-secure a machine after penetration. How do you know you've removed all the backdoors?
- It's hard to detect that a machine has been penetrated
- Any system with bugs is vulnerable
  - and all systems have bugs