Admin

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

• Profs: Dan Boneh and Zakir Durumeric

• Three programming projects (pairs) and two written homeworks

• Project #1 is posted. Please attend section this Friday!

• Use Piazza and Gradescope

• Automatic 72 hour extension

• No final exam this year
Live lectures on Zoom

Lectures are recorded ... posted on canvas

Slides

ask questions
The computer security problem

- Lots of buggy software
- Social engineering is very effective
- Money can be made from finding and exploiting vulns.

1. Marketplace for exploits
2. Marketplace for owned machines (PPI)
3. Many methods to profit from owned machines
Top 10 products by total number of “distinct” vulnerabilities in 2019

<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>Android</td>
<td>Google</td>
<td>OS</td>
<td>414</td>
</tr>
<tr>
<td>2</td>
<td>Debian Linux</td>
<td>Debian</td>
<td>OS</td>
<td>360</td>
</tr>
<tr>
<td>3</td>
<td>Windows Server 2016</td>
<td>Microsoft</td>
<td>OS</td>
<td>357</td>
</tr>
<tr>
<td>4</td>
<td>Windows 10</td>
<td>Microsoft</td>
<td>OS</td>
<td>357</td>
</tr>
<tr>
<td>5</td>
<td>Windows Server 2019</td>
<td>Microsoft</td>
<td>OS</td>
<td>351</td>
</tr>
<tr>
<td>6</td>
<td>Acrobat Reader Dc</td>
<td>Adobe</td>
<td>Application</td>
<td>342</td>
</tr>
<tr>
<td>7</td>
<td>Acrobat Dc</td>
<td>Adobe</td>
<td>Application</td>
<td>342</td>
</tr>
<tr>
<td>8</td>
<td>Cpanel</td>
<td>Cpanel</td>
<td>Application</td>
<td>321</td>
</tr>
<tr>
<td>9</td>
<td>Windows 7</td>
<td>Microsoft</td>
<td>OS</td>
<td>250</td>
</tr>
<tr>
<td>10</td>
<td>Windows Server 2008</td>
<td>Microsoft</td>
<td>OS</td>
<td>248</td>
</tr>
</tbody>
</table>

Vulnerable applications being exploited

Source: Kaspersky Security Bulletin 2017
Why so many security bugs?  Case study: Zoom client

Users have an expectation of privacy. But:

(1) Problems with crypto  (Marczak and Scott-Railton, April 2020)
(2) How not to save a user click  (J. Leitschuh, July 2019)
Why so many security bugs? Case study: Zoom client

Users have an expectation of privacy.

(1) Problems with crypto (Marczak and Scott-Railton, April 2020)

(2) How not to save a user clicks (J. Leitschuh, July 2019)

Can we bypass the security dialog?
Why so many security bugs?  Case study: Zoom client

Local Zoom web server listens on port  **localhost:19421**

- **To launch app**: web page from zoom.com tells browser to send an HTTP request to the local web server
- Web requests do not require a dialog ...

Can this be attacked?

http://localhost:19421/launch?action=join&confno=[confrence number]
Any web site can send a request to the local web server
• Joins users to conference w/o user’s knowledge!

• Fixed by Zoom. Web server removed by Apple’s MRT tool.
Why so many security bugs? Case study: Zoom client

Users have an expectation of privacy. But:

(1) Problems with crypto (Marczak and Scott-Railton, April 2020)

(2) How not to save a user click (J. Leitschuh, July 2019)

(3) Disable MacOS hardened runtime (P. Wardle, April 2020)

  Defends against code injection, library hijacking, and process memory space tampering.

Once user gives Zoom access to camera and mic, MacOS ensures that entire application code does not change.
What happens if protection is disabled?

Can this be abused?

requires user approval
The impact [Wardle, 4/2020]

Dynamic libraries loaded at Zoom startup

User approved access to camera & mic

Zoom app

libssl.1.0.0

curl64...

user’s MacOS system
The impact

[Wardle, 4/2020]

Attacker installs malware library that proxies libssl. ⇒ has access to camera & mic

Zoom app

libssl.1.0.0

curl64

libssl.1.0.0

user’s MacOS system

disable-library-validation:true

hardened runtime does not notify user of change to libssl!
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 systems?

1. IP address and bandwidth stealing

Attacker’s goal: look like a random Internet user

Use the IP address of infected machine or phone for:

• **Spam** (e.g. the storm botnet)
  
  Spamalytics: 1:12M pharma spams leads to purchase 1:260K greeting card spams leads to infection

• **Denial of Service:** Services: 1 hour (20$), 24 hours (100$)

• **Click fraud** (e.g. Clickbot.a)
Why compromise systems?

2. Steal user credentials

keylog for banking passwords, corporate passwords, gaming pwd's

Example: SilentBanker (and many like it)

User requests login page

Bank sends login page needed to log in

Malware injects Javascript

When user submits information, also sent to attacker

Man-in-the-Browser (MITB)

Similar mechanism used by Zeus botnet, and others
Lots of financial malware

<table>
<thead>
<tr>
<th></th>
<th>Malware Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trojan-Spy.Win32.Zbot</td>
</tr>
<tr>
<td>2</td>
<td>Trojan.Win32.Nymaim</td>
</tr>
<tr>
<td>3</td>
<td>Trojan.Win32.Neurevt</td>
</tr>
<tr>
<td>4</td>
<td>SpyEye</td>
</tr>
<tr>
<td>5</td>
<td>Trojan-Banker.Win32.Gozi</td>
</tr>
<tr>
<td>6</td>
<td>Emotet</td>
</tr>
<tr>
<td>7</td>
<td>Caphaw</td>
</tr>
<tr>
<td>8</td>
<td>Trickster</td>
</tr>
<tr>
<td>9</td>
<td>Cridex/Dridex</td>
</tr>
<tr>
<td>10</td>
<td>Backdoor.Win32.Shiz</td>
</tr>
</tbody>
</table>

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

Source: Kaspersky Security Bulletin 2017
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: 3. Ransomware

<table>
<thead>
<tr>
<th>Name</th>
<th>% of attacked users**</th>
</tr>
</thead>
<tbody>
<tr>
<td>WannaCry</td>
<td>7.71</td>
</tr>
<tr>
<td>Locky</td>
<td>6.70</td>
</tr>
<tr>
<td>Cerber</td>
<td>5.89</td>
</tr>
<tr>
<td>Jaff</td>
<td>2.58</td>
</tr>
<tr>
<td>Cryrar/ACCDFISA</td>
<td>2.20</td>
</tr>
<tr>
<td>Spora</td>
<td>2.19</td>
</tr>
<tr>
<td>Purgen/GlobelImposter</td>
<td>2.11</td>
</tr>
<tr>
<td>Shade</td>
<td>2.06</td>
</tr>
<tr>
<td>Crysis</td>
<td>1.25</td>
</tr>
<tr>
<td>CryptoWall</td>
<td>1.13</td>
</tr>
</tbody>
</table>

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)
Oops, your files have been encrypted!

What Happened to My Computer?
Your important files are encrypted. Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You can decrypt some of your files for free. Try now by clicking <Decrypt>. But if you want to decrypt all your files, you need to pay.
You only have 3 days to submit the payment. After that the price will be doubled. Also, if you don't pay in 7 days, you won't be able to recover your files forever. We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>. Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.
After your payment, click <Check Payment>. Best time to check: Between 11:00am GMT from Monday to Friday.

Send $300 worth of bitcoin to this address:

115p7UMMngoj1pMvkpHljcRdfJNXj6LrLn

Contact Us
Server-side attacks

• **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

• **Political motivation:**
  – DNC, Tunisia Facebook (Feb. 2011), GitHub (Mar. 2015)

• **Infect visiting users**
Infecting visiting users. Example: Mpack

• PHP-based tools installed on compromised web sites
  – Embedded as an iframe on infected page
  – Infects browsers that visit site

• Features
  – management console provides stats on infection rates
  – Sold for several 100$ 
  – Customer care can be purchased, one-year support contract

• Impact: 500,000 infected sites (compromised via SQL injection)
  – Several defenses:  e.g. Google safe browsing
Data theft: what is stolen  (2012-2015)

Source: California breach notification report, 2015
How companies lose customer data

- Physical document loss: 21%
- Malware/hacking: 32%
- Insider misuse/attack: 8%
- Accidental disclosure: 17%
- Lost/stolen laptops or servers: 22%

How do we have this data?

Source: PrivacyRights.org, 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

Option 2:
• Zerodium: up to $2M for iOS, $2.5M for Android (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 <a href="https://zerodium.com/faq.html">ZERO-DIUM Zero-Day Research Feed</a>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who are ZERODIUM’s customers?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERODIUM customers are <a href="https://zerodium.com/faq.html">government organizations</a> (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](https://zerodium.com/faq.html)

Dan Boneh
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 time: control hijacking vulnerabilities

THE END