CS155
Computer Security
Course overview

Acknowledgments: Lecture slides are from the Computer Security course taught by Dan Boneh and Zakir Durumeric at Stanford University. When slides are obtained from other sources, a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.
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 (gaining a foothold)
2. Marketplace for malware (post compromise)
3. Strong economic and political motivation for using both
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 2020
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  Spain</td>
<td>14.03</td>
</tr>
<tr>
<td>2  France</td>
<td>13.54</td>
</tr>
<tr>
<td>3  Canada</td>
<td>11.35</td>
</tr>
<tr>
<td>4  USA</td>
<td>10.76</td>
</tr>
<tr>
<td>5  India</td>
<td>10.53</td>
</tr>
<tr>
<td>6  Brazil</td>
<td>10.22</td>
</tr>
<tr>
<td>7  Mexico</td>
<td>9.86</td>
</tr>
<tr>
<td>8  Italy</td>
<td>9.80</td>
</tr>
<tr>
<td>9  Australia</td>
<td>9.09</td>
</tr>
<tr>
<td>10  Great Britain</td>
<td>8.99</td>
</tr>
</tbody>
</table>

Source: Kaspersky Security Bulletin 2020
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)

User requests login page

Malware injects Javascript

Bank sends login page needed to log in

When user submits information, also sent to attacker

Man-in-the-Browser (MITB)

Similar mechanism used by Zbot, 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
### 2. Ransomware

Why own machines:

<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>
WannaCry ransomware
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 2020
Why compromise end user machines?

4. 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)
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:
   - DNC (2015), Ukraine power grid (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: 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???
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
  \[\implies\] many compromised customers

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

- Defenses?
Data theft: what is stolen (2012-2015)

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

- **Insider misuse/attack**: 7%
- **Physical document loss**: 21%
- **Malware/hacking**: 32%
- **Accidental disclosure**: 22%
- **Lost/stolen laptops or servers**: 17%

**How do we have this data?**

Source: PrivacyRights.org, 2020
Why compromise web sites: (3) infect users

- **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
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 (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>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>How the acquired security research is used by ZERODIUM?</td>
<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>
<tr>
<td>Who are ZERODIUM’s customers?</td>
<td>ZERODIUM customers are government organizations (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)
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:

```plaintext
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
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