# CS155

# Computer Security

## Course overview

*Acknowledgments: Lecture slides are from the Computer Security course taught by Dan Boneh at Stanford University. When slides are obtained from other sources, a  a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.*

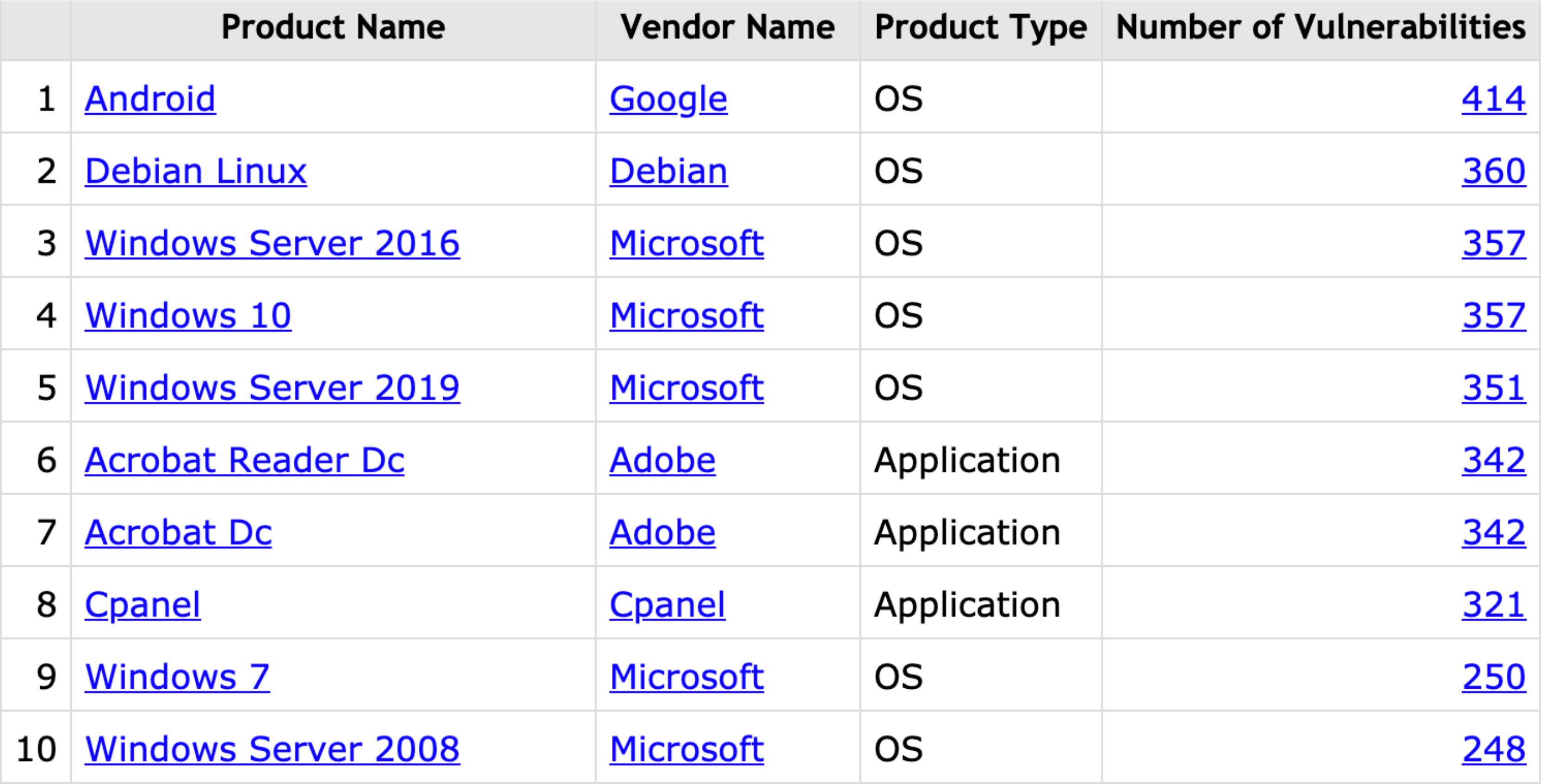
# The computer security problem

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

1. Marketplace for exploits
2. Marketplace for owned machines (PPI)
3. Many methods to profit from owned machines

current state of computer security

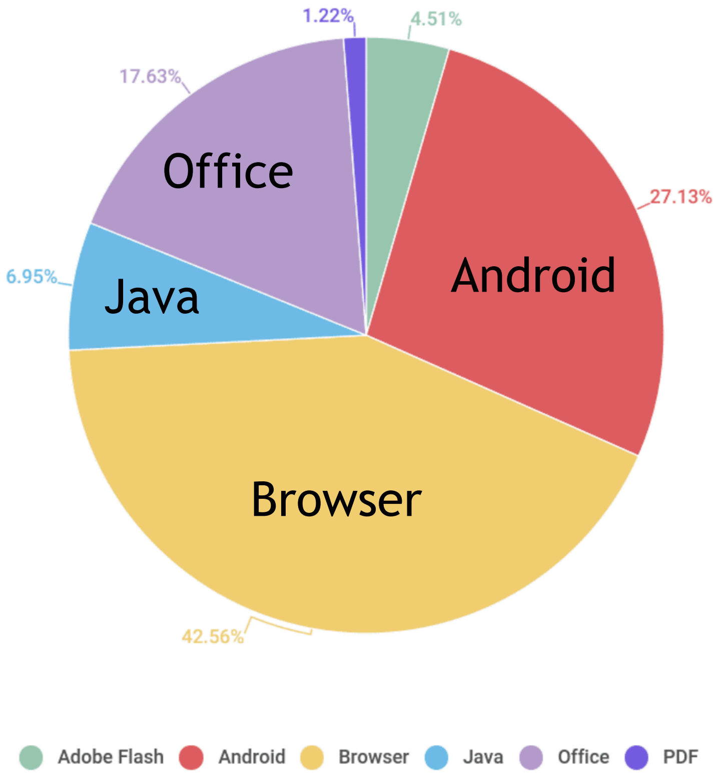
# Top 10 products by total number of "distinct" vulnerabilities in 2019



| | Product Name | Vendor Name | Product Type | Number of Vulnerabilities |
|---|---|---|---|---|
| 1 | Android | Google | OS | 414 |
| 2 | Debian Linux | Debian | OS | 360 |
| 3 | Windows Server 2016 | Microsoft | OS | 357 |
| 4 | Windows 10 | Microsoft | OS | 357 |
| 5 | Windows Server 2019 | Microsoft | OS | 351 |
| 6 | Acrobat Reader Dc | Adobe | Application | 342 |
| 7 | Acrobat Dc | Adobe | Application | 342 |
| 8 | Cpanel | Cpanel | Application | 321 |
| 9 | Windows 7 | Microsoft | OS | 250 |
| 10 | Windows Server 2008 | Microsoft | OS | 248 |

source: https://www.cvedetails.com/top-50-products.php?year=2019

# Vulnerable applications being exploited



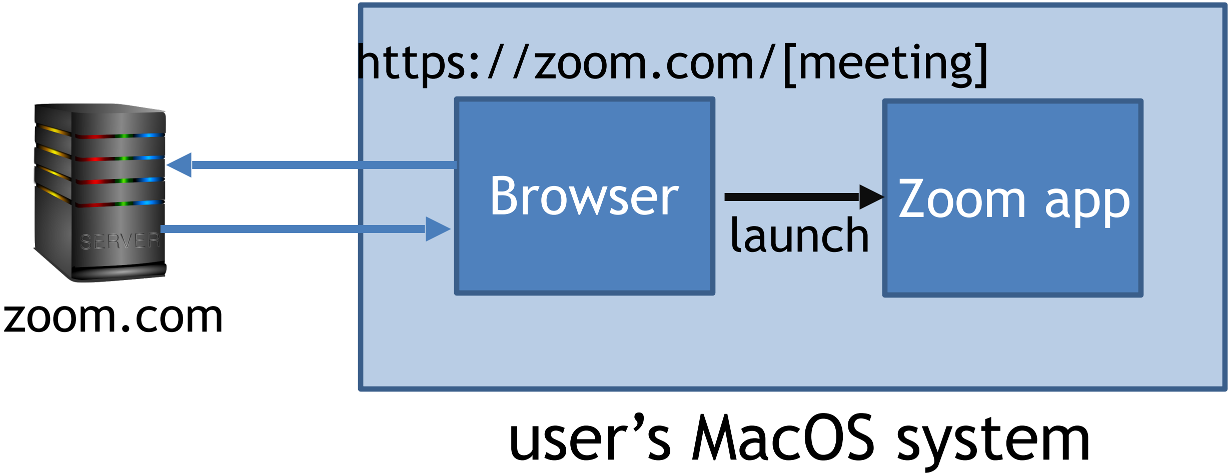
Source: Kaspersky Security Bulletin 2017

# **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)

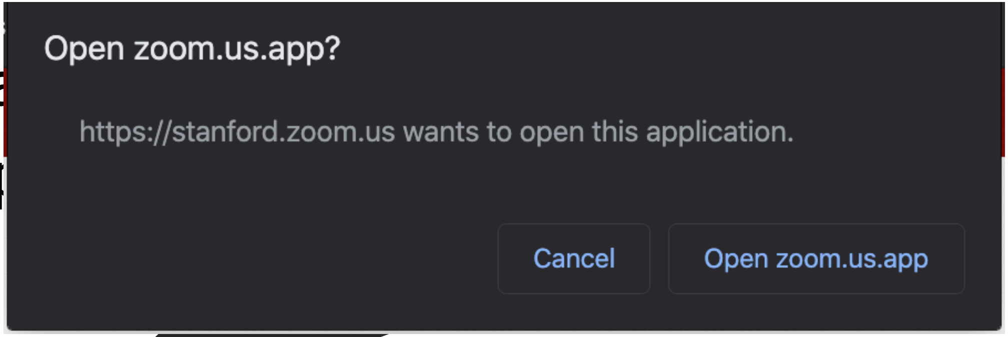


# Why so many security bugs? Case study: Zoom client

Users have an expecta

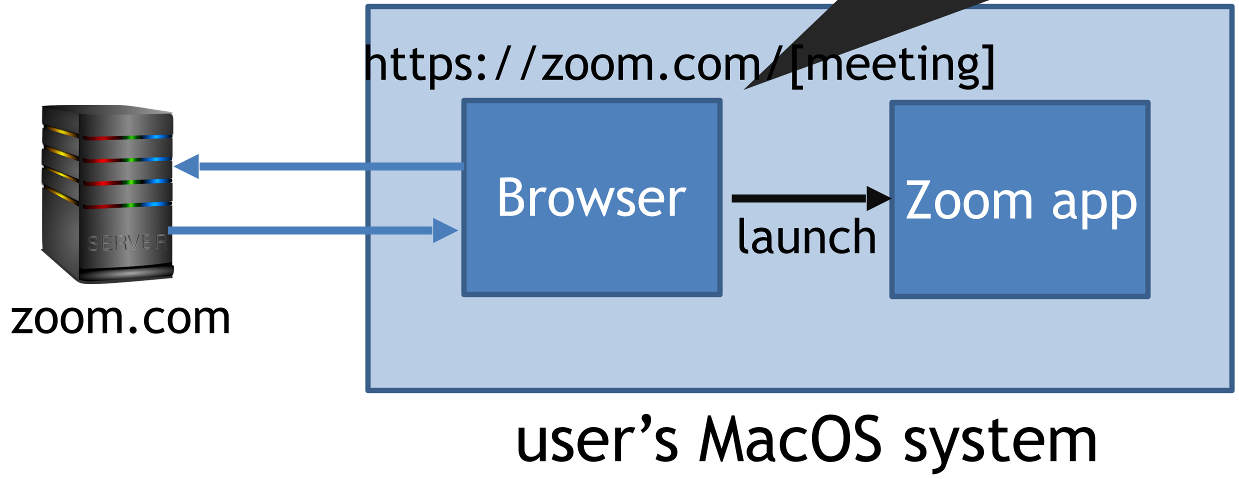
(1) Problems with cry 20)

(2) How **not** to save a



Open zoom.us.app?

https://stanford.zoom.us wants to open this application.

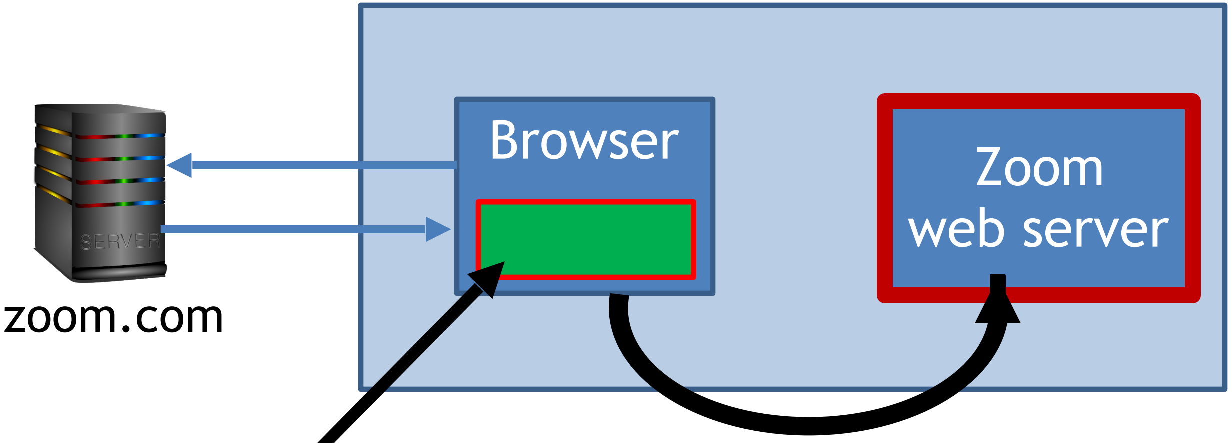


Can we bypass the security dialog?

# 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]**

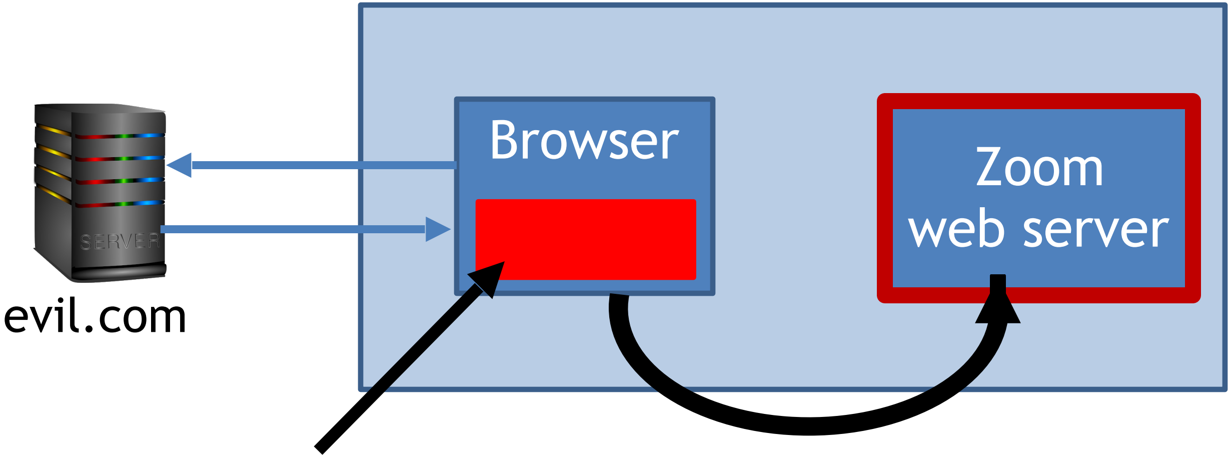
# The problem [J. Leitschuh, July 2019]

**Any web site** can send a request to the local web server

- Joins users to conference w/o user’s knowledge!

What happened next? Responsible disclosure, 90 days (CVE-2019-13450).

- Fixed by Zoom. Web server removed by Apple’s MRT tool.



http://localhost:19421/launch?action=join&confno=[confrence number]

# **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)

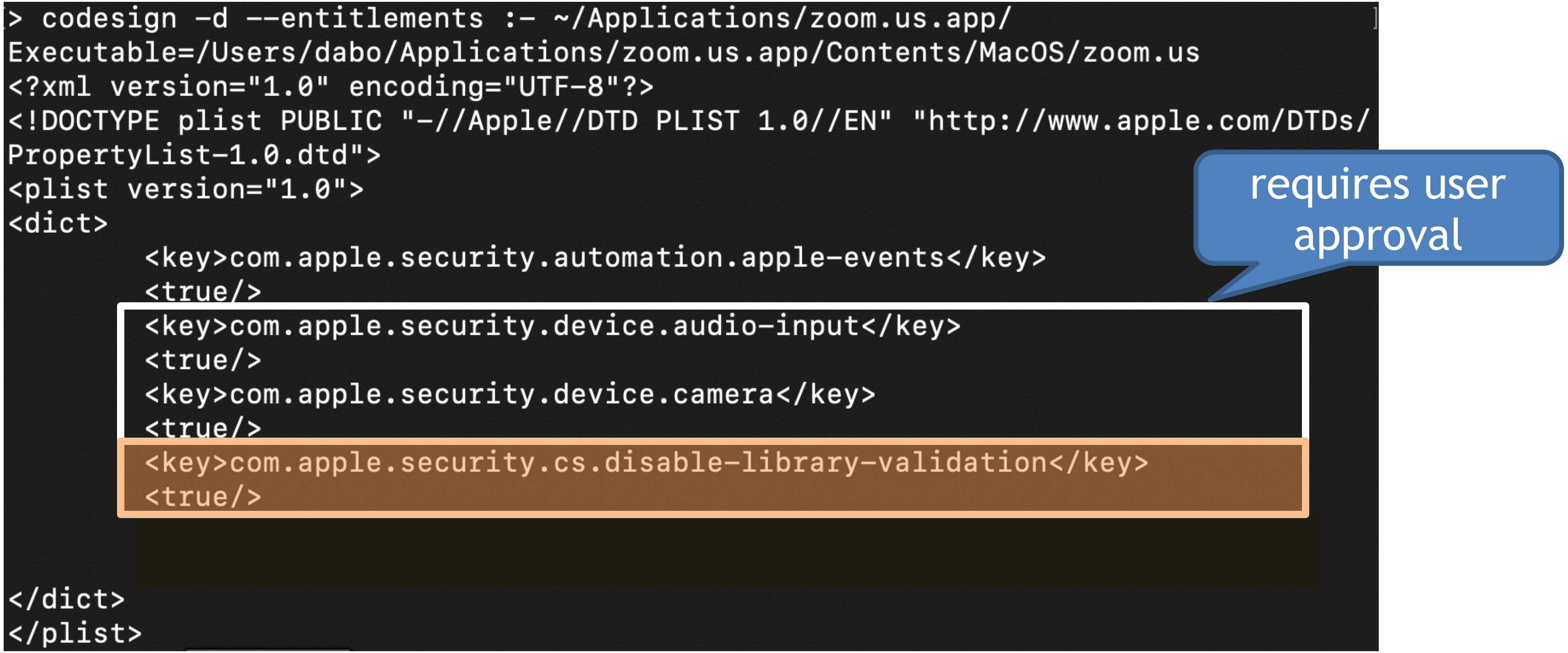
(2) How not to save a user click (J. Leitschuh, July 2019)

(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?



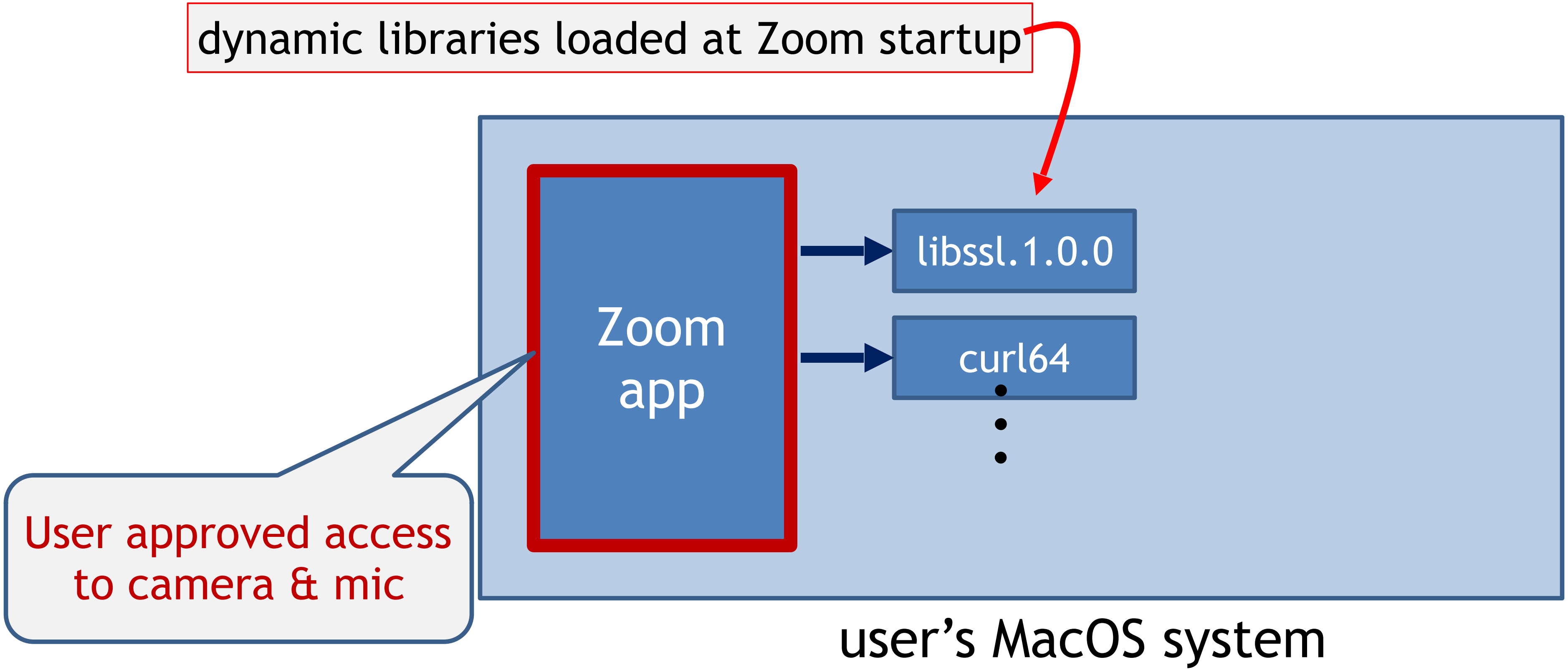
```
> codesign -d --entitlements :- ~/Applications/zoom.us.app/
Executable=/Users/dabo/Applications/zoom.us.app/Contents/MacOS/zoom.us
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.apple.security.automation.apple-events</key>
        <true/>
        <key>com.apple.security.device.audio-input</key>
        <true/>
        <key>com.apple.security.device.camera</key>
        <true/>
        <key>com.apple.security.cs.disable-library-validation</key>
        <true/>


</dict>
</plist>
```

requires user approval

Can this be abused?

# The impact [Wardle, 4/2020]

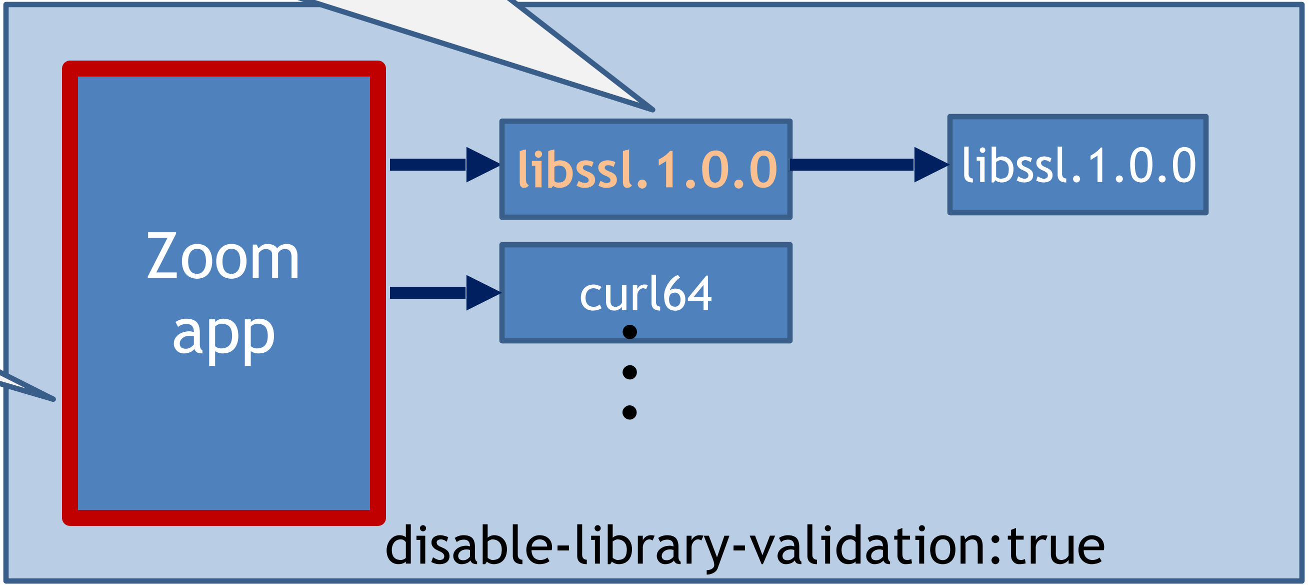

dynamic libraries loaded at Zoom startup
Zoom app
libssl.1.0.0
curl64
⋮
User approved access to camera & mic

user's MacOS system

# The impact [Wardle, 4/2020]

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

hardened runtime does not notify user of change to libssl!



user's MacOS system

# 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

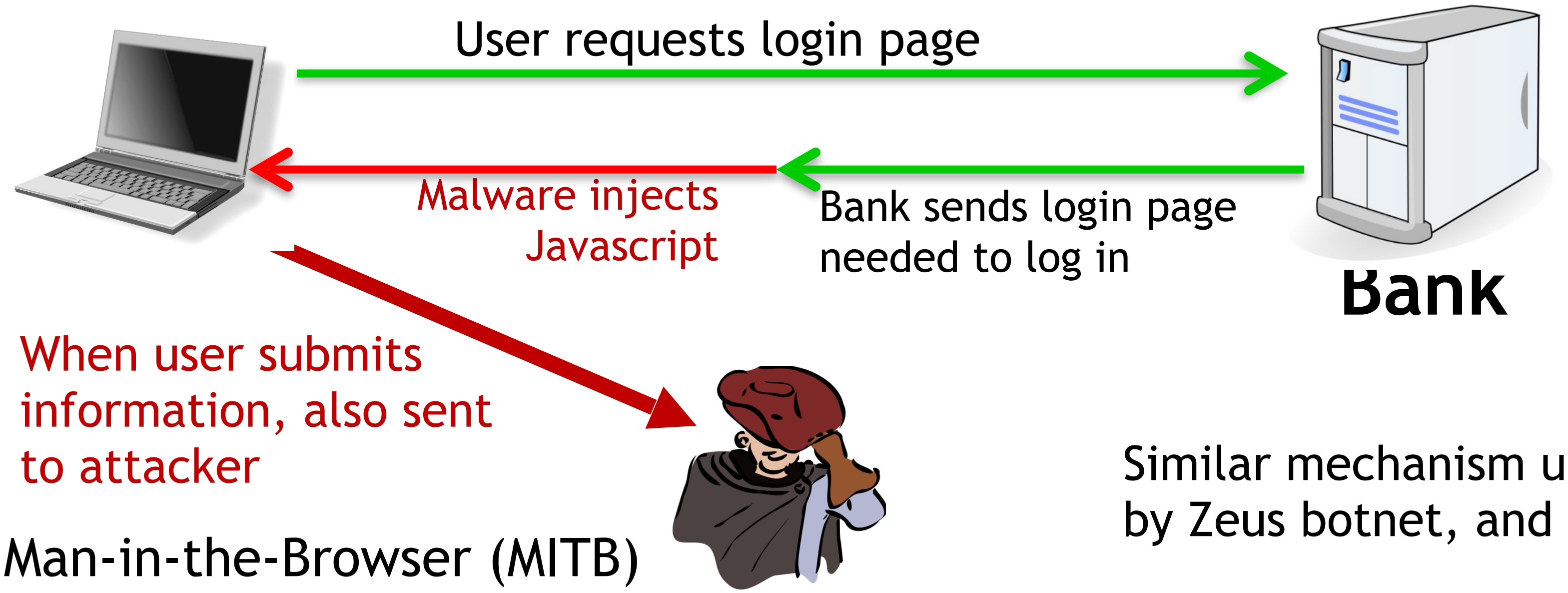
- **Denial of Service:** Services: 1 hour (20$), 24 hours (100$)
- **Click fraud** (e.g. Clickbot.a)

# Why compromise systems?

## 2. Steal user credentials

keylog for banking passwords, corporate passwords, gaming pwds

Example: SilentBanker (and many like it)



User requests login page

Malware injects Javascript

Bank sends login page needed to log in

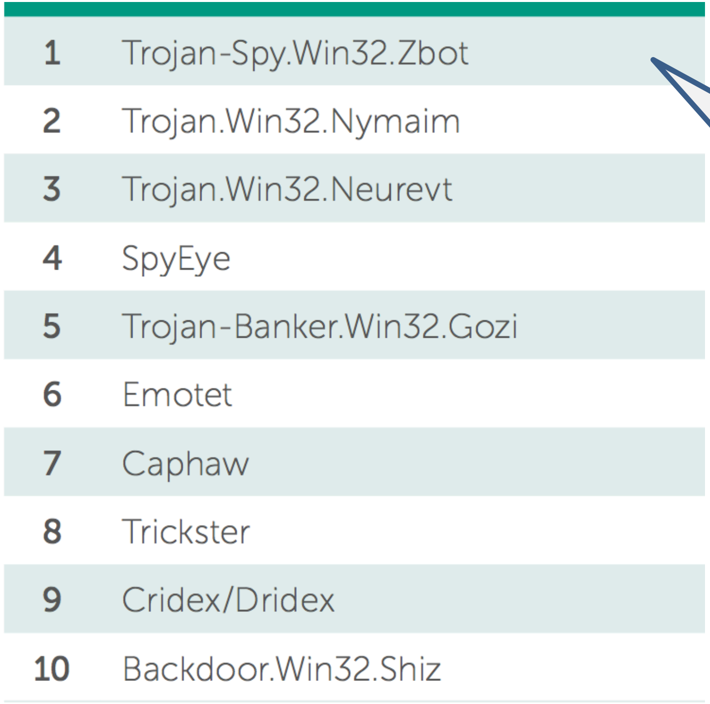
Bank

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



| | |
|---|---|
| 1 | Trojan-Spy.Win32.Zbot |
| 2 | Trojan.Win32.Nymaim |
| 3 | Trojan.Win32.Neurevt |
| 4 | SpyEye |
| 5 | Trojan-Banker.Win32.Gozi |
| 6 | Emotet |
| 7 | Caphaw |
| 8 | Trickster |
| 9 | Cridex/Dridex |
| 10 | Backdoor.Win32.Shiz |

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

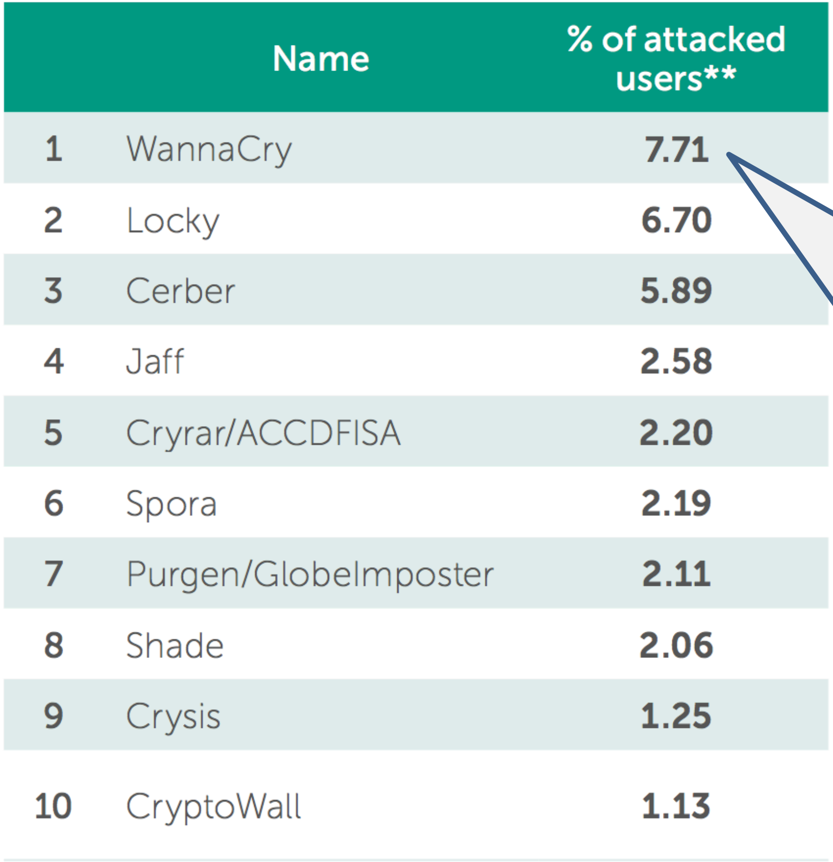
Source: Kaspersky Security Bulletin 2017

# 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**

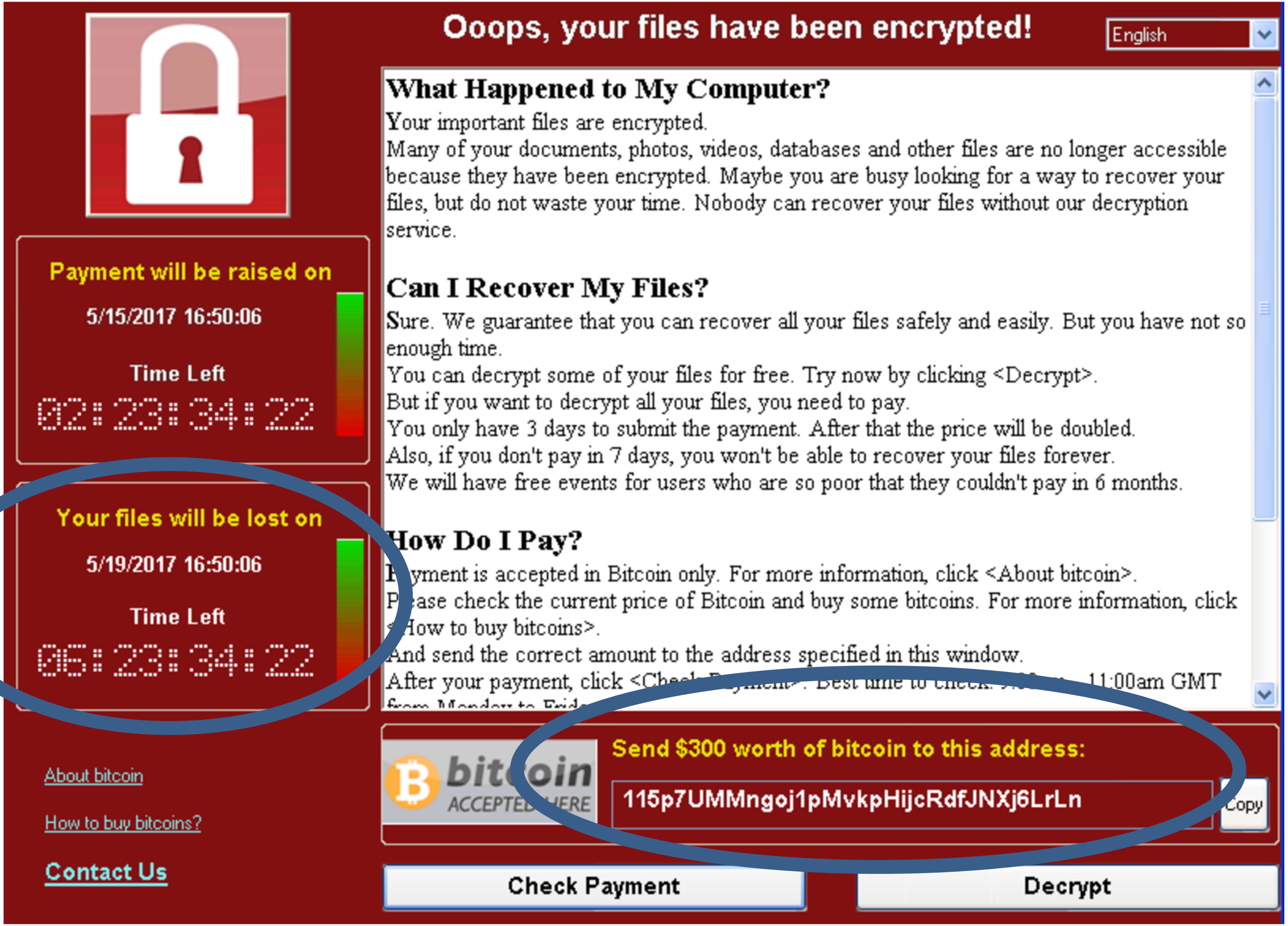


| | Name | % of attacked users** |
|---|---|---|
| 1 | WannaCry | 7.71 |
| 2 | Locky | 6.70 |
| 3 | Cerber | 5.89 |
| 4 | Jaff | 2.58 |
| 5 | Cryrar/ACCDFISA | 2.20 |
| 6 | Spora | 2.19 |
| 7 | Purgen/GlobeImposter | 2.11 |
| 8 | Shade | 2.06 |
| 9 | Crysis | 1.25 |
| 10 | CryptoWall | 1.13 |

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)

# WannaCry ransomware



## Ooops, 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: 9:00am - 11:00am GMT from Monday to Frida

Payment will be raised on

5/15/2017 16:50:06

Time Left

02:23:34:22

Your files will be lost on

5/19/2017 16:50:06

Time Left

06:23:34:22

Send $300 worth of bitcoin to this address:

115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn

# 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**

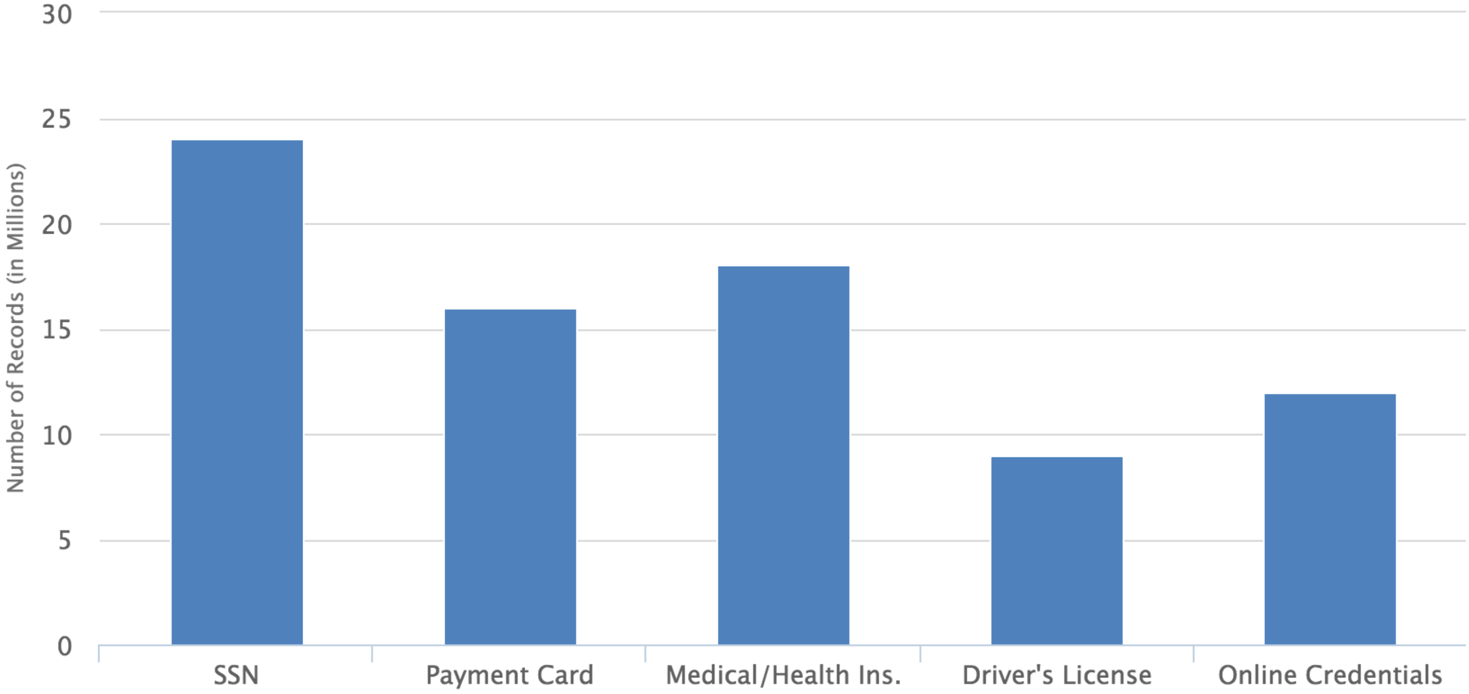
# Infecting visiting users. Example: Mpack

- 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

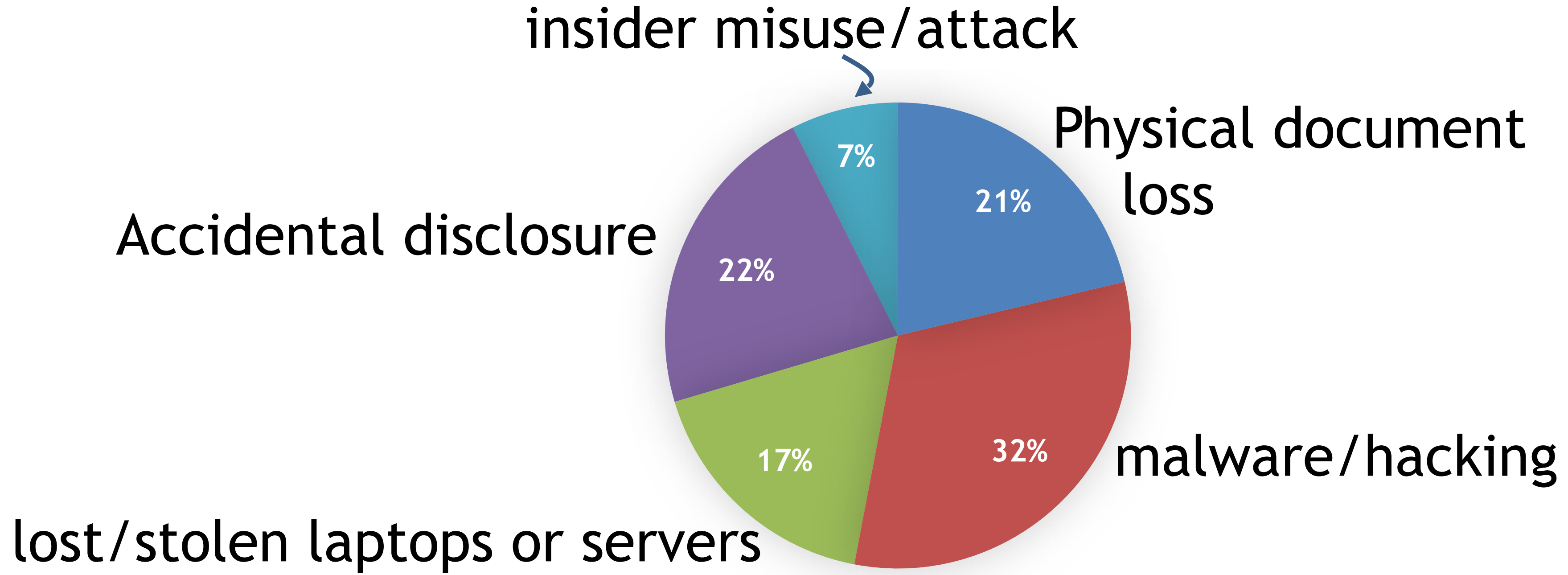
- Impact: 500,000 infected sites (compromised via SQL injection)
  - Several defenses: e.g. Google safe browsing

# Data theft: what is stolen (2012-2015)



Source: California breach notification report, 2015

# How companies lose customer data



How do we have this data?

Source: PrivacyRights.org, 2020

Dan Boneh

# Insider attacks:  example

```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
            retval = -EINVAL;
```

See: http://lwn.net/Articles/57135/

# Insider attacks: example

Hidden trap door in Linux (nov 2003)

- Allows attacker to take over a computer
- Practically undetectable change (uncovered via CVS logs)

Inserted line in wait4()

```
 if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
             retval = -EINVAL;
```

Looks like a standard error check, but …

See: http://lwn.net/Articles/57135/

# 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**:

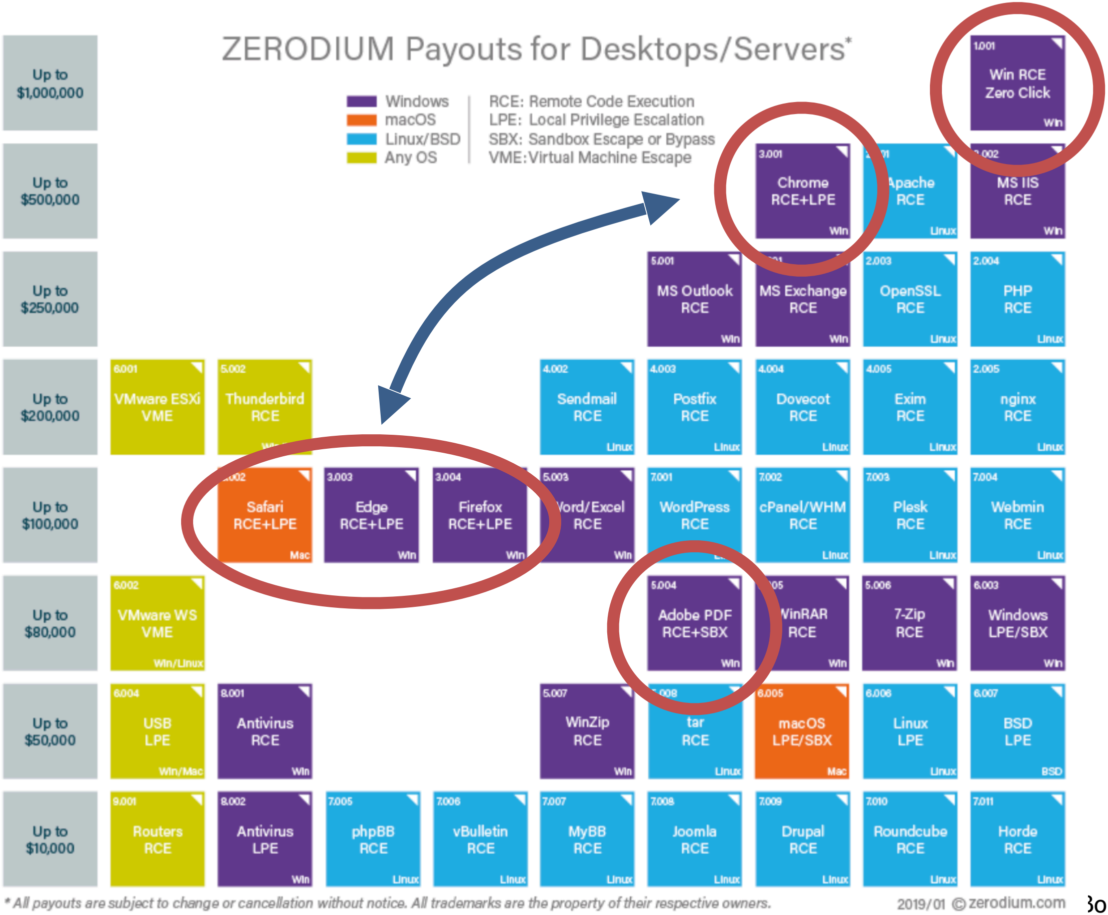
- Zerodium: up to $2M for iOS, $2.5M for Android (2019)
- … many others

# Marketplace for Vulnerabilities

RCE: remote code execution

LPE: local privilege escalation

SBX: sandbox escape



ZERODIUM Payouts for Desktops/Servers*

| Legend | Abbreviations |
|---|---|
| ▇ Windows | RCE: Remote Code Execution |
| ▇ macOS | LPE: Local Privilege Escalation |
| ▇ Linux/BSD | SBX: Sandbox Escape or Bypass |
| ▇ Any OS | VME: Virtual Machine Escape |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Up to $1,000,000 | | | | | | | | | 1.001<br>Win RCE Zero Click<br>Win |
| Up to $500,000 | | | | | | | 3.001<br>Chrome RCE+LPE<br>Win | ?01<br>Apache RCE<br>Linux | ?.002<br>MS IIS RCE<br>Win |
| Up to $250,000 | | | | | | 5.001<br>MS Outlook RCE<br>Win | ?01<br>MS Exchange RCE<br>Win | 2.003<br>OpenSSL RCE<br>Linux | 2.004<br>PHP RCE<br>Linux |
| Up to $200,000 | 6.001<br>VMware ESXi VME | 5.002<br>Thunderbird RCE<br>Wi | | | 4.002<br>Sendmail RCE<br>Linux | 4.003<br>Postfix RCE<br>Linux | 4.004<br>Dovecot RCE<br>Linux | 4.005<br>Exim RCE<br>Linux | 2.005<br>nginx RCE<br>Linux |
| Up to $100,000 | | ?.002<br>Safari RCE+LPE<br>Mac | 3.003<br>Edge RCE+LPE<br>Win | 3.004<br>Firefox RCE+LPE<br>Win | 5.003<br>Word/Excel RCE<br>Win | 7.001<br>WordPress RCE | 7.002<br>cPanel/WHM RCE<br>Linux | 7.003<br>Plesk RCE<br>Linux | 7.004<br>Webmin RCE<br>Linux |
| Up to $80,000 | 6.002<br>VMware WS VME<br>Win/Linux | | | | | 5.004<br>Adobe PDF RCE+SBX<br>Win | ?05<br>WinRAR RCE<br>Win | 5.006<br>7-Zip RCE<br>Win | 6.003<br>Windows LPE/SBX<br>Win |
| Up to $50,000 | 6.004<br>USB LPE<br>Win/Mac | 8.001<br>Antivirus RCE<br>Win | | | 5.007<br>WinZip RCE<br>Win | ?008<br>tar RCE<br>Linux | 6.005<br>macOS LPE/SBX<br>Mac | 6.006<br>Linux LPE<br>Linux | 6.007<br>BSD LPE<br>BSD |
| Up to $10,000 | 9.001<br>Routers RCE | 8.002<br>Antivirus LPE<br>Win | 7.005<br>phpBB RCE<br>Linux | 7.006<br>vBulletin RCE<br>Linux | 7.007<br>MyBB RCE<br>Linux | 7.008<br>Joomla RCE<br>Linux | 7.009<br>Drupal RCE<br>Linux | 7.010<br>Roundcube RCE<br>Linux | 7.011<br>Horde RCE<br>Linux |

* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/01 © zerodium.com

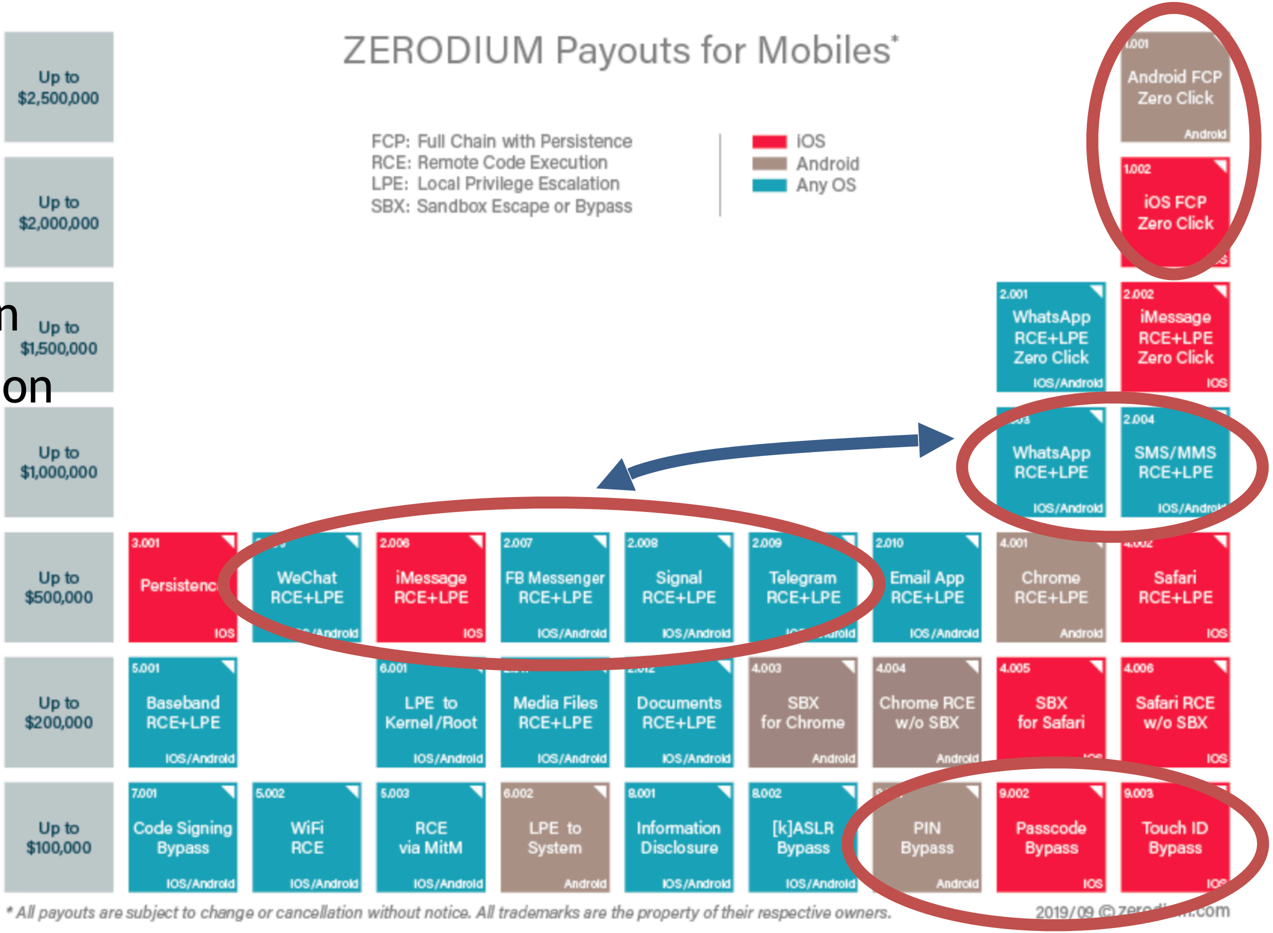
Source: Zerodium payouts

# Marketplace for Vulnerabilities

RCE: remote code execution

LPE: local privilege escalation

SBX: sandbox escape



ZERODIUM Payouts for Mobiles*

FCP: Full Chain with Persistence
RCE: Remote Code Execution
LPE: Local Privilege Escalation
SBX: Sandbox Escape or Bypass

🟥 iOS
🟫 Android
🟦 Any OS

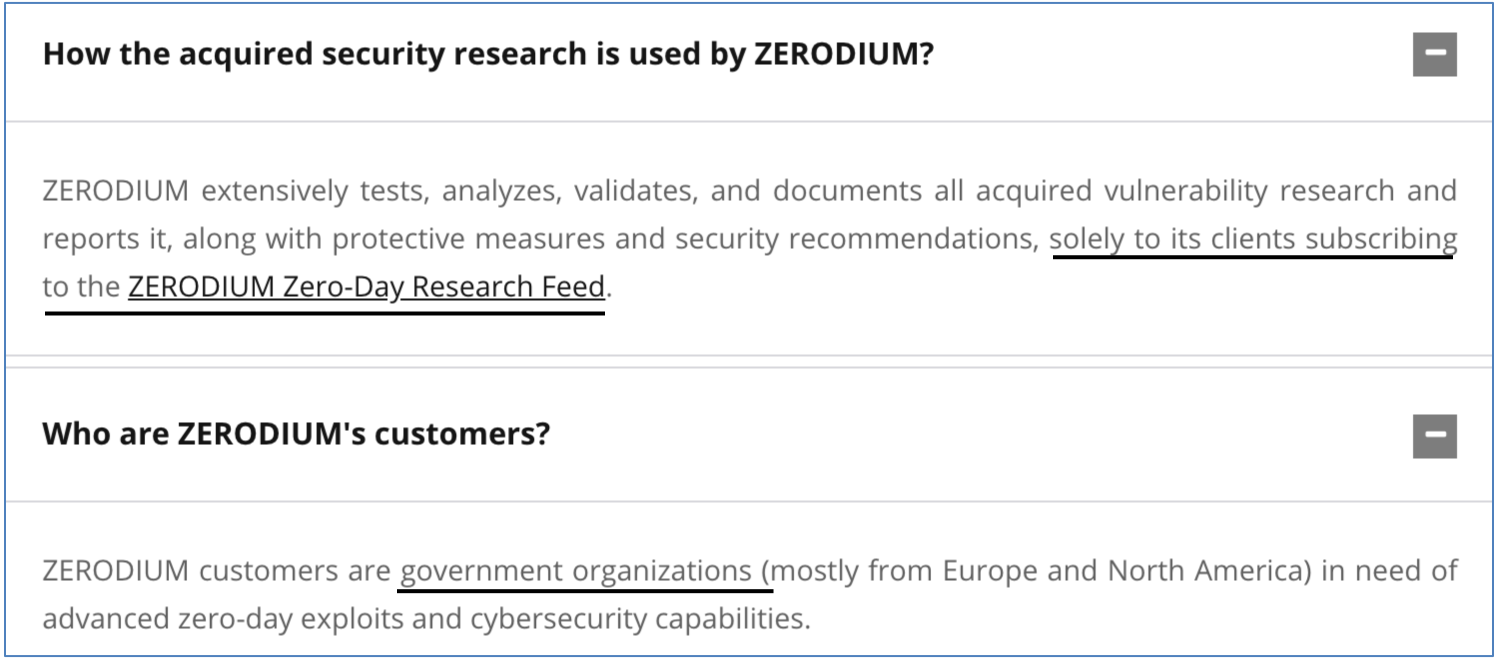
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Up to $2,500,000 | | | | | | | | | 1.001<br>Android FCP Zero Click<br>Android |
| Up to $2,000,000 | | | | | | | | | 1.002<br>iOS FCP Zero Click<br>iOS |
| Up to $1,500,000 | | | | | | | | 2.001<br>WhatsApp RCE+LPE Zero Click<br>iOS/Android | 2.002<br>iMessage RCE+LPE Zero Click<br>iOS |
| Up to $1,000,000 | | | | | | | | 2.003<br>WhatsApp RCE+LPE<br>iOS/Android | 2.004<br>SMS/MMS RCE+LPE<br>iOS/Android |
| Up to $500,000 | 3.001<br>Persistence<br>iOS | WeChat RCE+LPE<br>iOS/Android | 2.006<br>iMessage RCE+LPE<br>iOS | 2.007<br>FB Messenger RCE+LPE<br>iOS/Android | 2.008<br>Signal RCE+LPE<br>iOS/Android | 2.009<br>Telegram RCE+LPE<br>iOS/Android | 2.010<br>Email App RCE+LPE<br>iOS/Android | 4.001<br>Chrome RCE+LPE<br>Android | 4.002<br>Safari RCE+LPE<br>iOS |
| Up to $200,000 | 5.001<br>Baseband RCE+LPE<br>iOS/Android | | 6.001<br>LPE to Kernel/Root<br>iOS/Android | 2.011<br>Media Files RCE+LPE<br>iOS/Android | 2.012<br>Documents RCE+LPE<br>iOS/Android | 4.003<br>SBX for Chrome<br>Android | 4.004<br>Chrome RCE w/o SBX<br>Android | 4.005<br>SBX for Safari<br>iOS | 4.006<br>Safari RCE w/o SBX<br>iOS |
| Up to $100,000 | 7.001<br>Code Signing Bypass<br>iOS/Android | 5.002<br>WiFi RCE<br>iOS/Android | 5.003<br>RCE via MitM<br>iOS/Android | 6.002<br>LPE to System<br>Android | 8.001<br>Information Disclosure<br>iOS/Android | 8.002<br>[k]ASLR Bypass<br>iOS/Android | PIN Bypass<br>Android | 9.002<br>Passcode Bypass<br>iOS | 9.003<br>Touch ID Bypass<br>iOS |

* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/09 © zerodium.com

Source: Zerodium payouts

# Why buy 0days?



**How the acquired security research is used by ZERODIUM?**

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.

**Who are ZERODIUM's customers?**

ZERODIUM customers are government organizations (mostly from Europe and North America) in need of advanced zero-day exploits and cybersecurity capabilities.

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:

```
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:

```
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 ⇒ 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