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 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

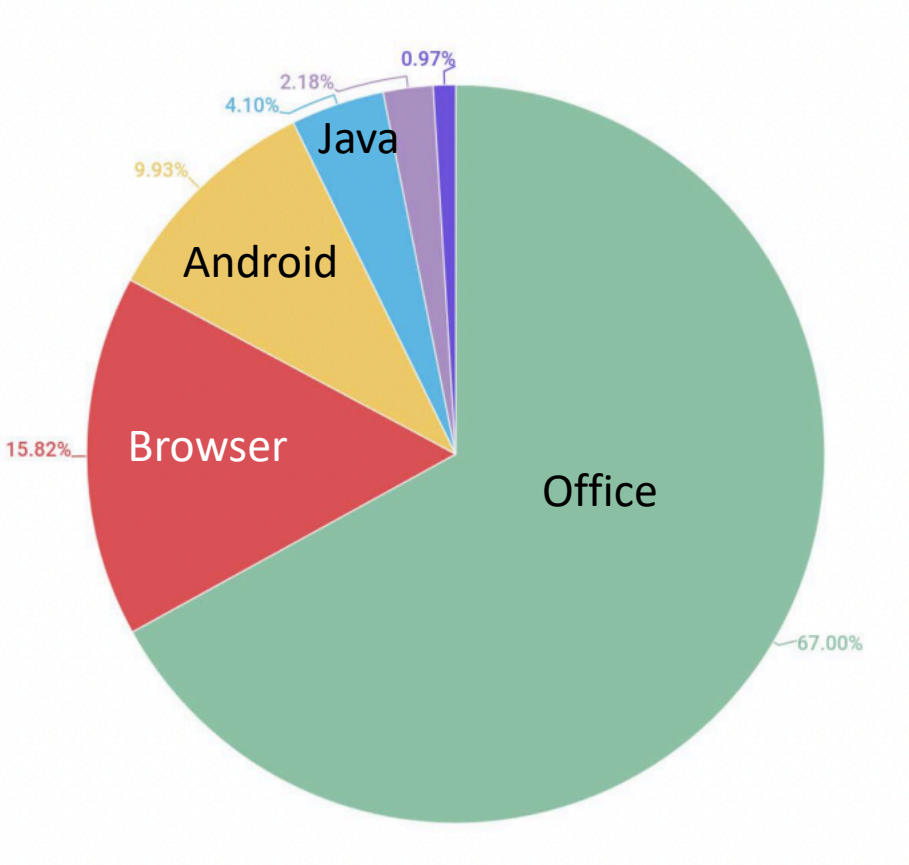
current state of computer security

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 2020

A global problem

Top 10 countries by share of attacked users:

	Country*	%**
1	Spain	14.03
2	France	13.54
3	Canada	11.35
4	USA	10.76
5	India	10.53
6	Brazil	10.22
7	Mexico	9.86
8	Italy	9.80
9	Australia	9.09
10	Great Britain	8.99

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?

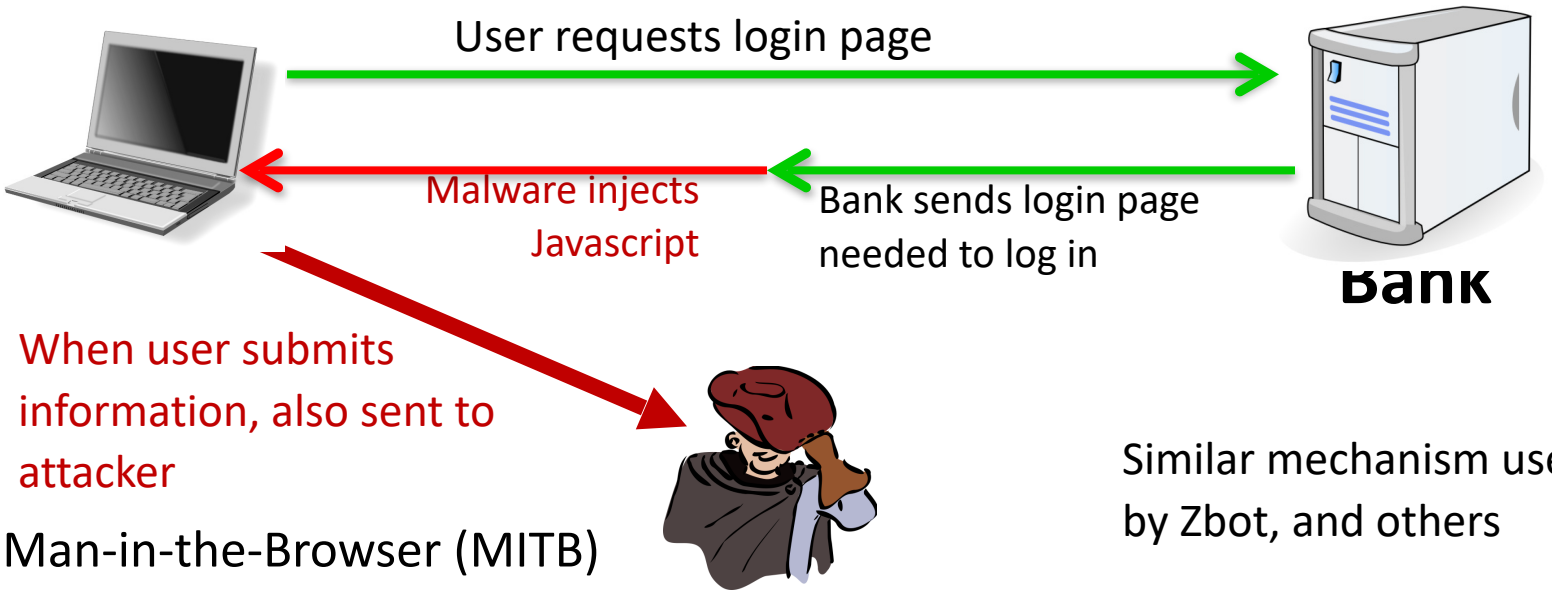
… economics

Why compromise end user machines?

1. Steal user credentials

keylog for banking passwords, corporate passwords, gaming pwds

Example: SilentBanker (and many like it)



Man-in-the-Browser (MITB)

Similar mechanism used by Zbot, 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: 2. **Ransomware**

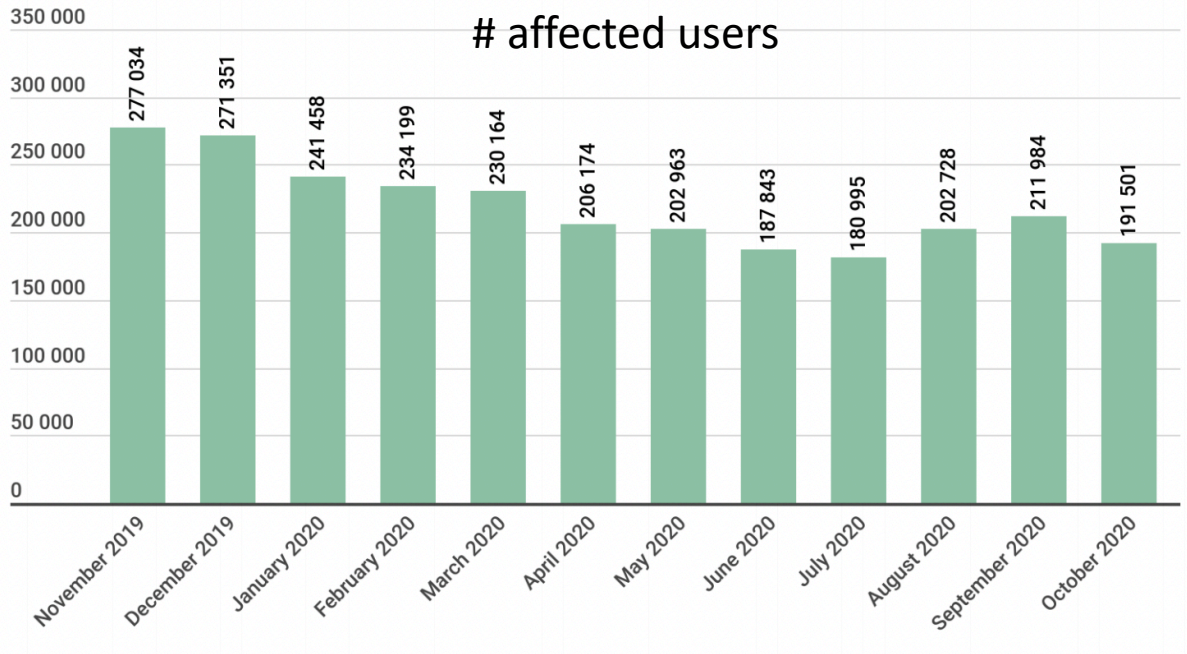
	Name	% of attacked users**
1	WannaCry	7.71
2	Locky	6.70
3	Cerber	5.89
4	Jaff	2.58
5	Cryrar/ACCDFISA	2.20
6	Spora	2.19
7	Purgen/GlobeImposter	2.11
8	Shade	2.06
9	Crysis	1.25
10	CryptoWall	1.13

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

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)

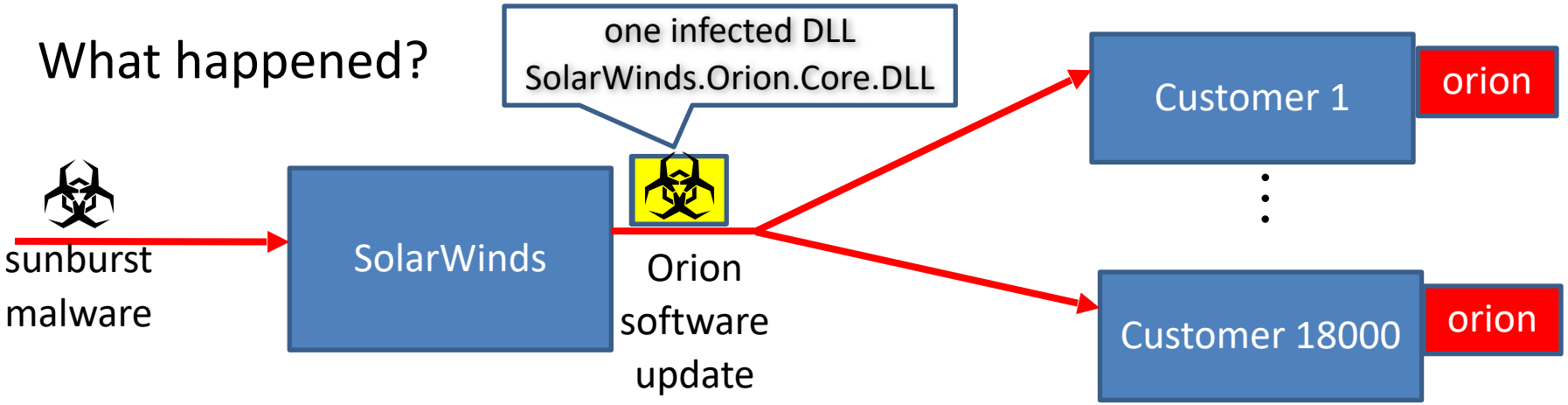
will discuss tools during course

… 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

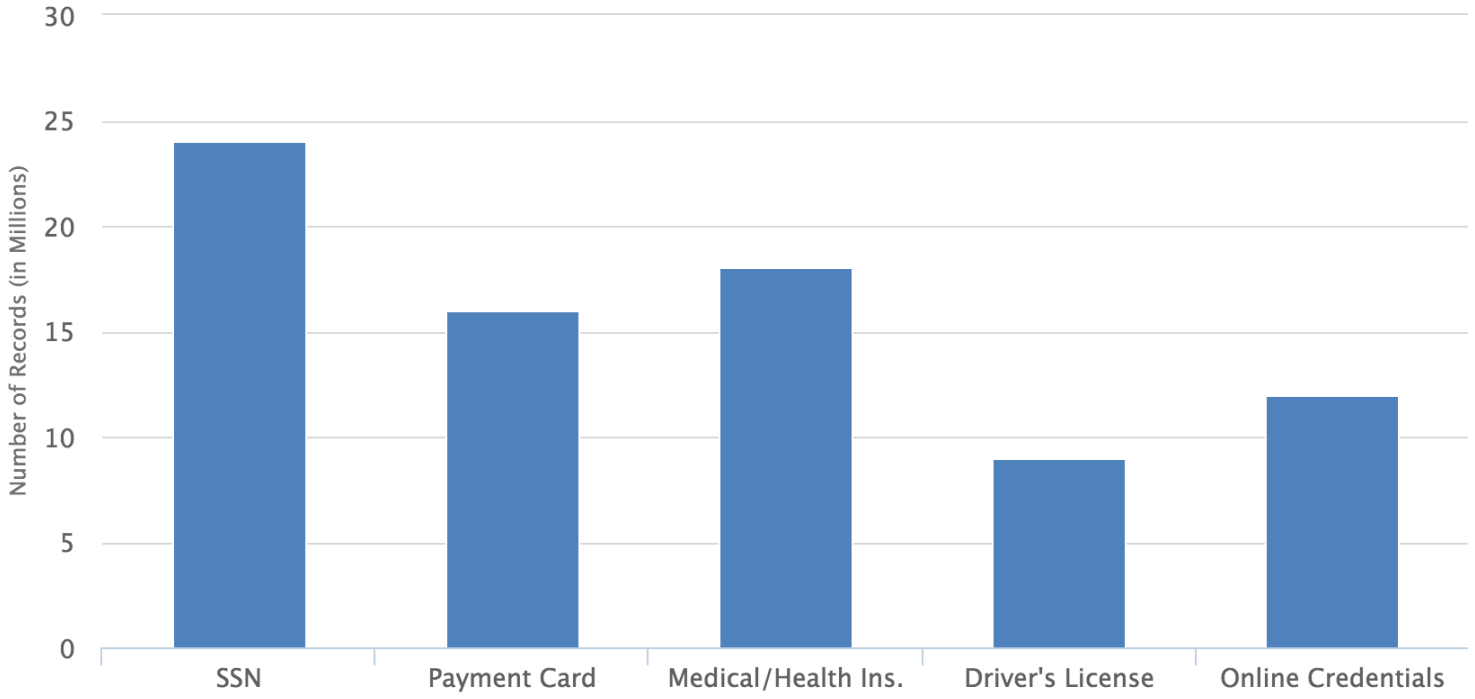
More generally: a **supply chain attack**

- Software, hardware, or service supplier is compromised

  $\Longrightarrow$ many compromised customers

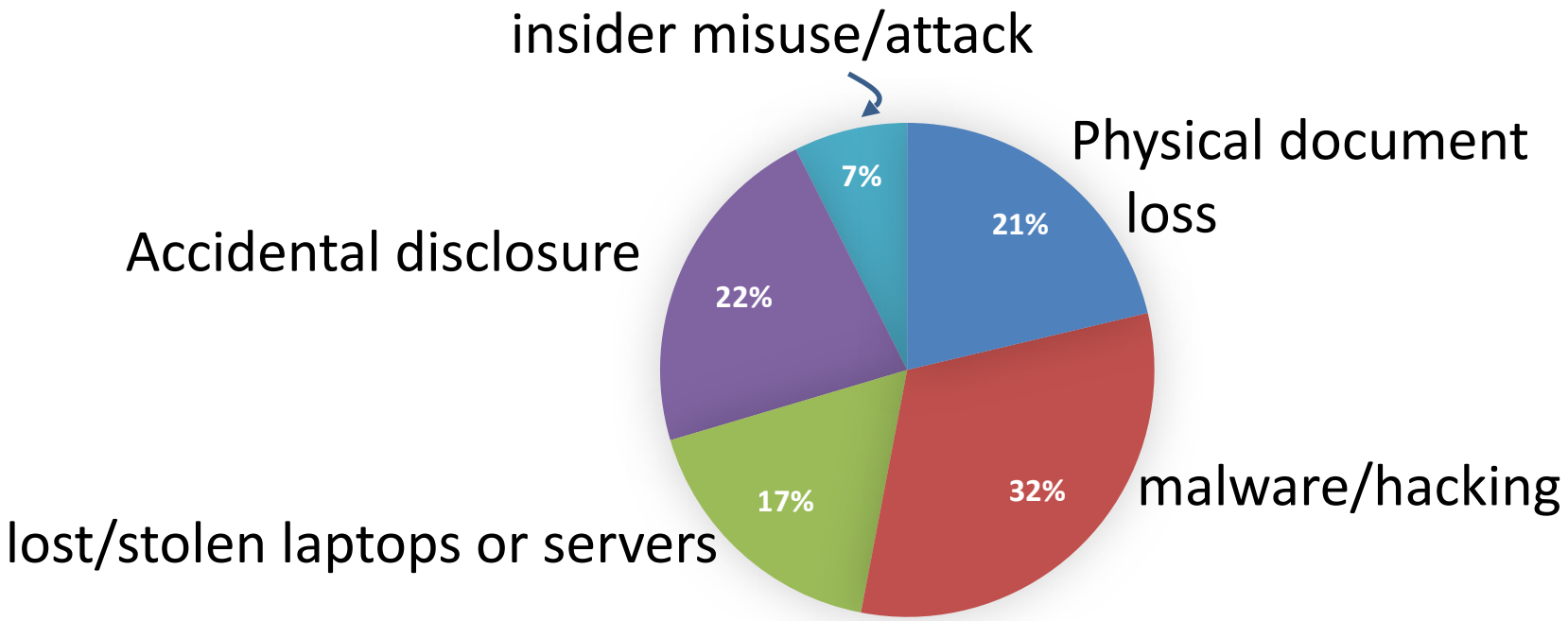
- Many examples of this in the past (e.g., Target 2013, …)
- Defenses?

Data theft: what is stolen (2012-2015)



Source: California breach notification report, 2015

How companies lose customer data



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:

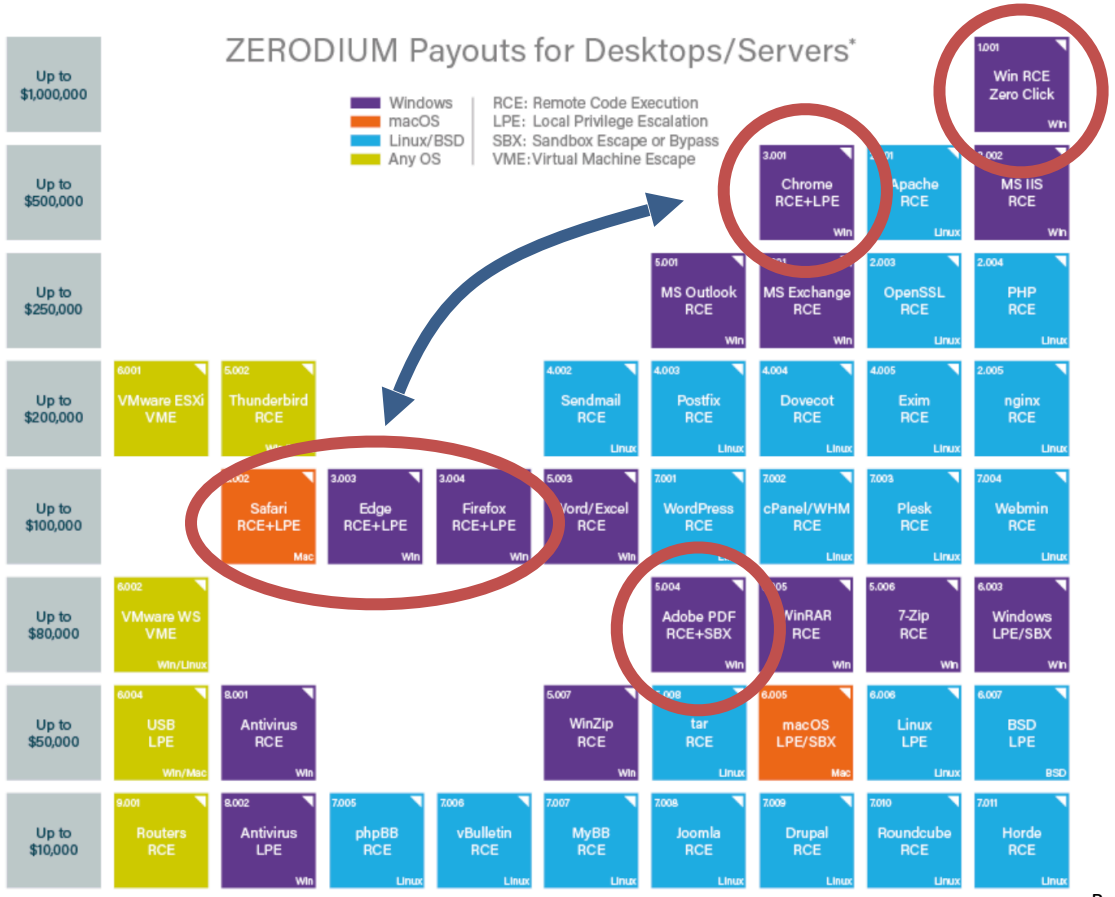
- Zerodium: up to $2M for iOS, $2.5M for Android (since 2019)
- … many others

Marketplace for Vulnerabilities

RCE: remote code execution

LPE: local privilege escalation

SBX: sandbox escape



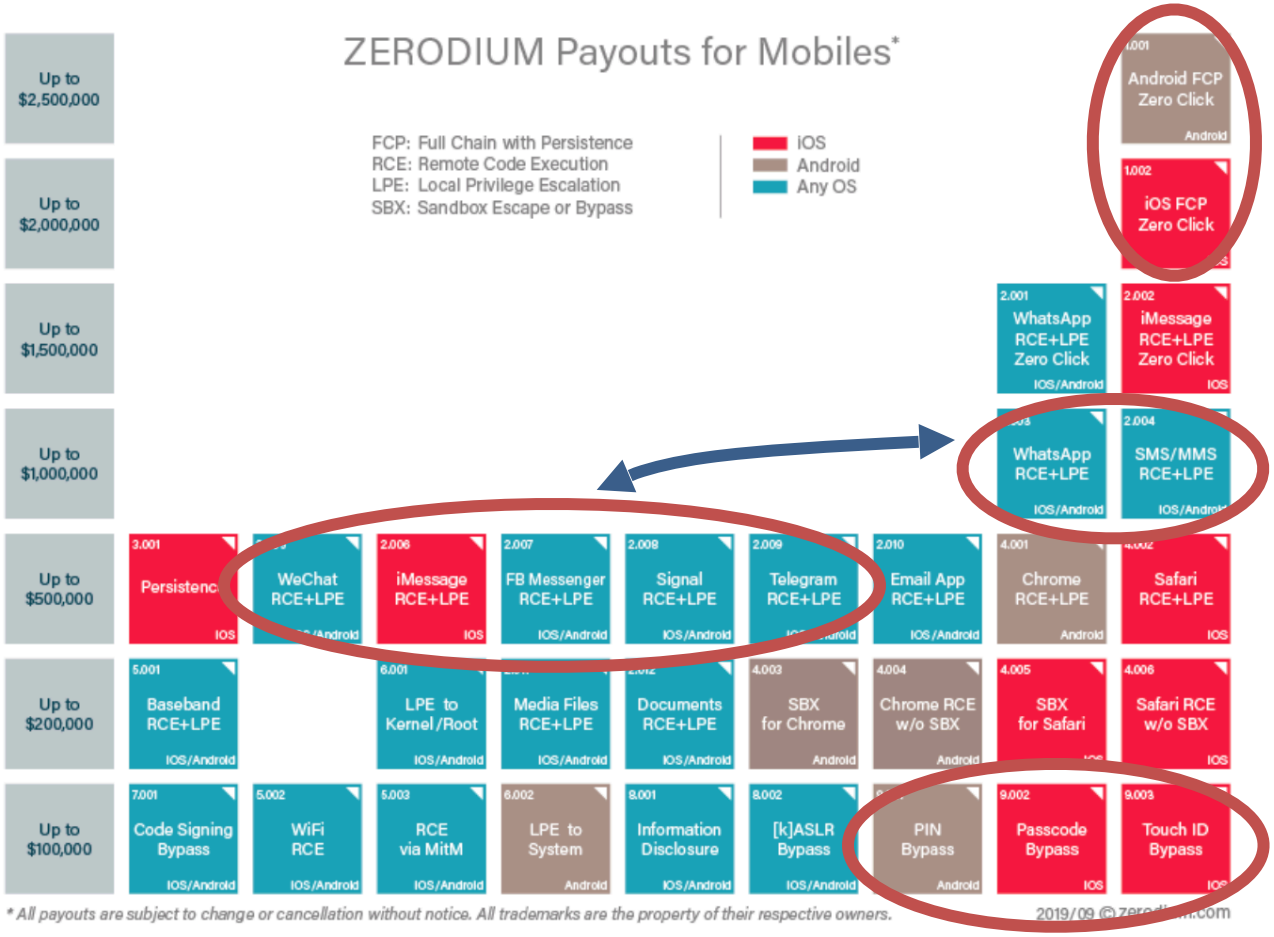
Source: Zerodium payouts

Marketplace for Vulnerabilities

RCE: remote code execution

LPE: local privilege escalation

SBX: sandbox escape



Source: Zerodium payouts

Why buy 0days?

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

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