CS155

Computer Security

Course overview

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 on Wednesday. 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

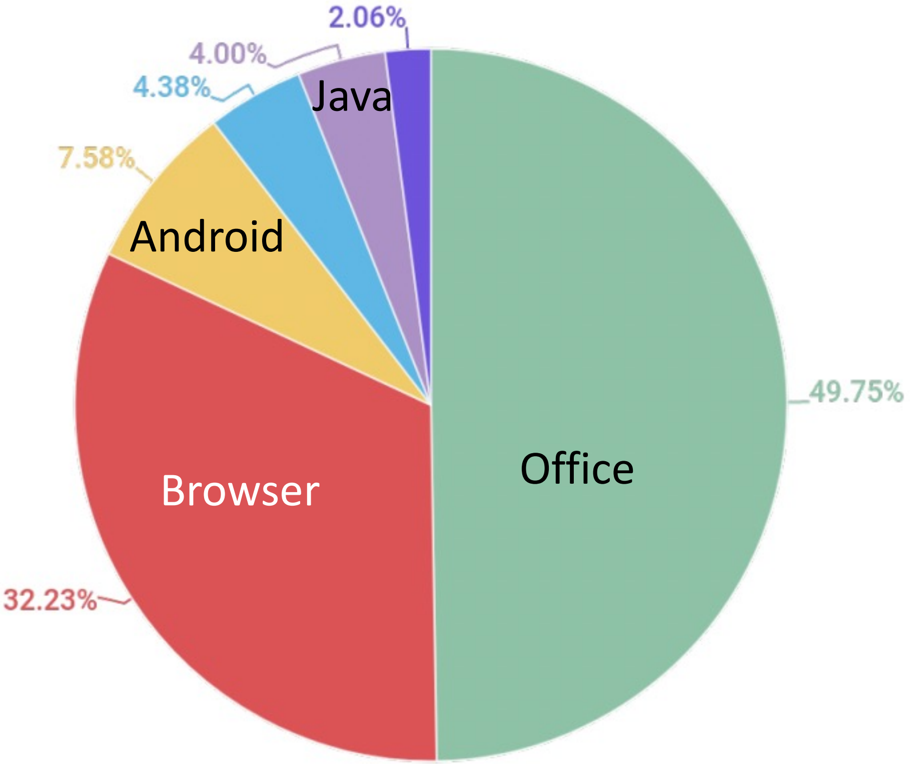
current state of computer security

Top 10 products by total number of distinct vulnerabilities in 2023

Product name	Vendor	# vulnerabilities
Android	Google	1422
Microsoft Server	Microsoft	2059
Fedora	Fedora Project	540
Windows 11	Microsoft	1004
Debian Linux	Debian	487
MacOS	Apple	418
Chrome	Google	296
iPhone OS	Apple	269

source: https://www.cvedetails.com/top-50-products.php?year=2023

Distribution of exploits used in attacks



Source: Kaspersky Security Bulletin 2021

A global problem

Top 10 countries by share of attacked users:

	Country*	%**
1	Ecuador	9.01
2	France	8.04
3	Spain	7.30
4	Vietnam	6.89
5	Canada	6.81
6	India	6.45
7	Italy	6.27
8	Turkey	6.19
9	United States	5.91
10	Mexico	5.60

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 and cloud applications, hardware features**

Don’t try this at home !

Introduction

What motivates attackers?

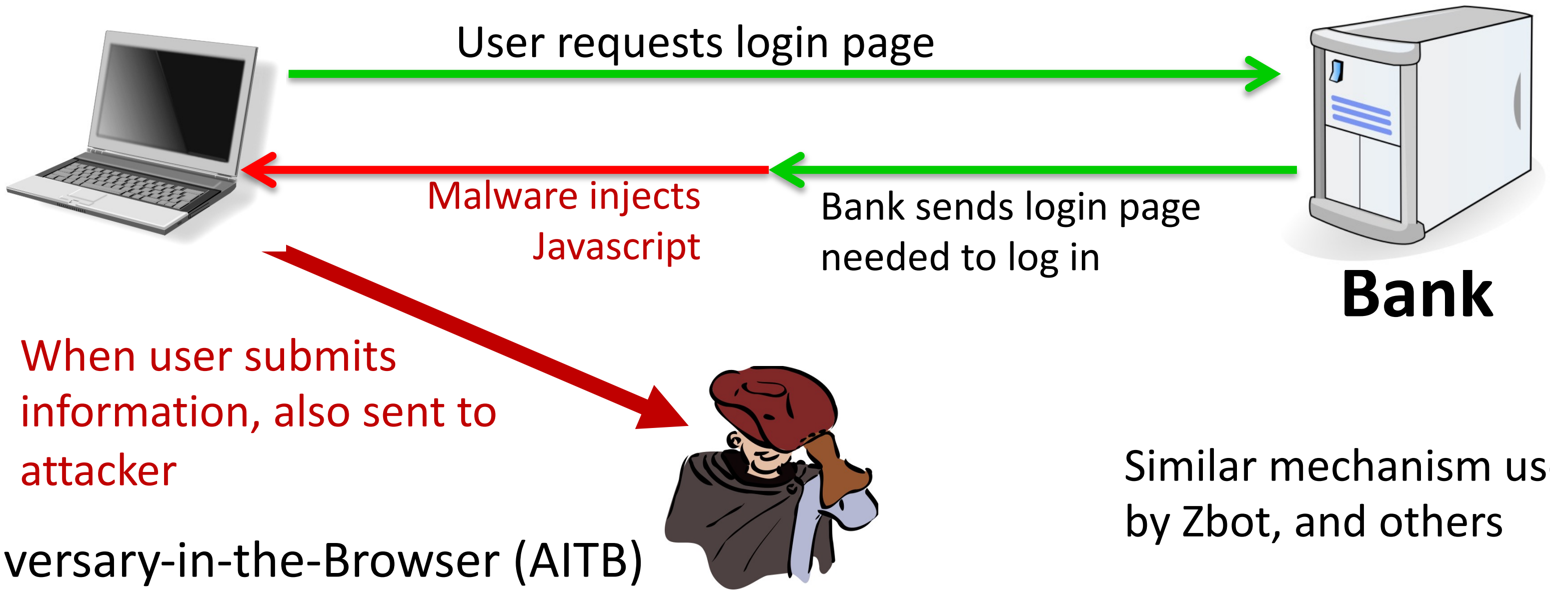
… economics

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)

Similar mechanism used by Zbot, and others

Lots of financial malware

	Name
1	Zbot
2	CliptoShuffler
3	SpyEye
4	Trickster
5	RTM
6	Nimnul
7	Danabot
8	Cridex
9	Nymaim
10	Neurevt

- 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**

	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

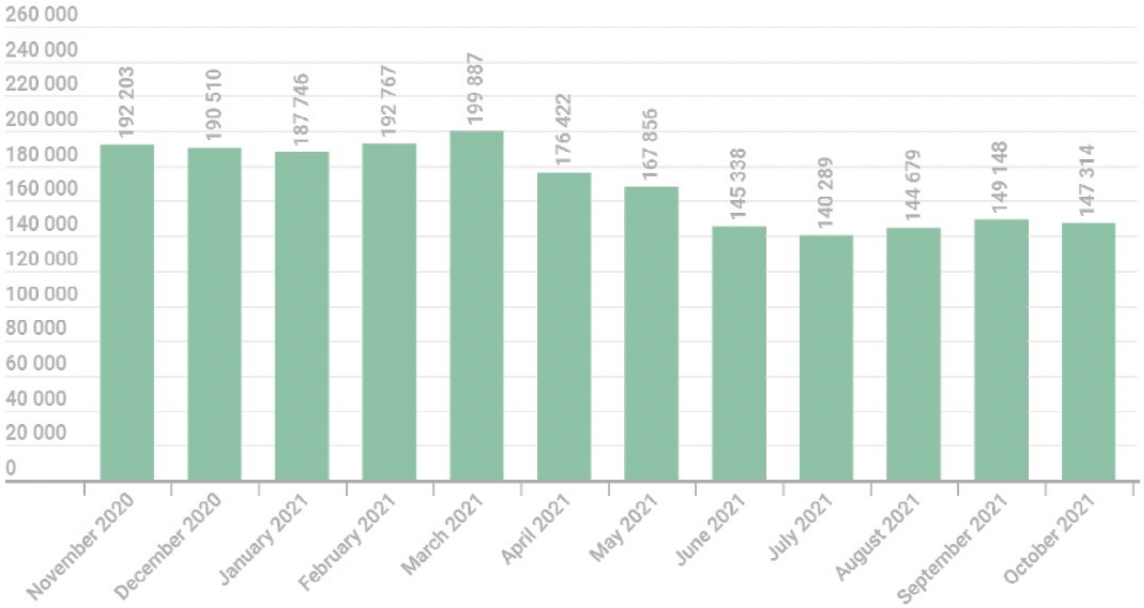
a worldwide problem

- 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**

affected users



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

More devastating: **server-side attacks**

(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),
- Ukraine attacks (2014: election, 2015,2016: power grid, 2017: NotPetya, …)

(3) Infect visiting users

Result: many server-side Breaches

<u>Typical attack steps</u>:

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

Security tools available to
try and stop each step (**<u>kill chain</u>**)

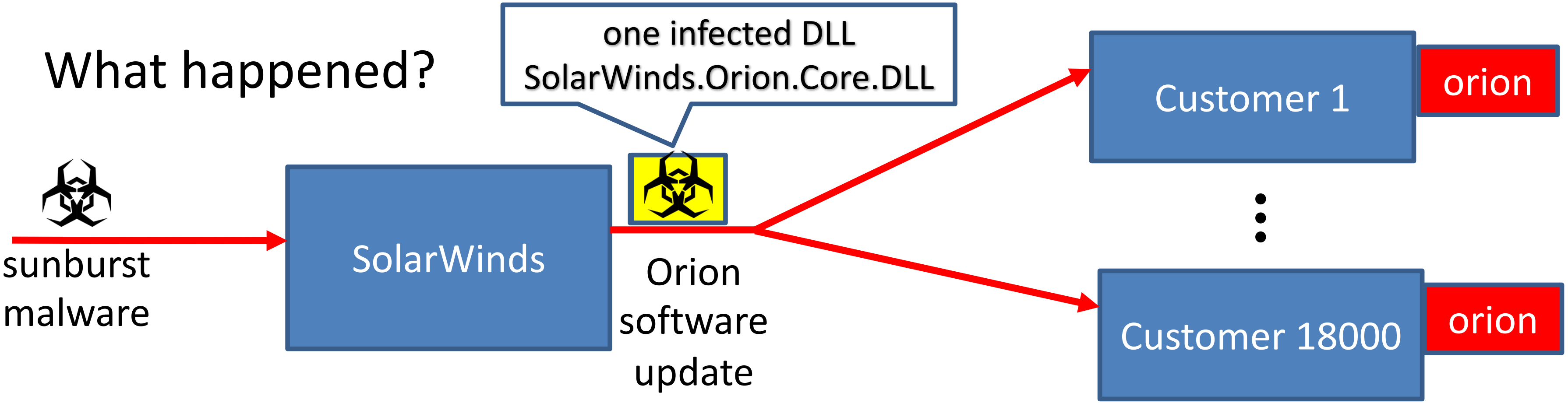
will discuss tools during course

… but no complete solution

Case study 1: 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 <u>Dec. 2020</u> .

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

  $\Longrightarrow$ many compromised customers

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

- Defenses?

Case study 2: 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 $\Longrightarrow$ compromise dependent projects

Many examples:

Package name	Maintainer	Payload
noblesse	xin1111	Discord token stealer, Credit card stealer (**Windows-based**)
genesisbot	xin1111	*Same as noblesse*
aryi	xin1111	*Same as noblesse*
suffer	suffer	*Same as noblesse , obfuscated by PyArmor*

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

A recent example: **xz Utils**

- An open source compression utility on Github

- Feb. 23, 2024: one of the two long-time maintainers introduced an update that includes a malicious install script

- So what? <u>**sshd**</u> has a dependency on xz Utils …
  $\Rightarrow$ enables remote access into servers running sshd

- Fortunately, this was caught before wide deployment

Security considerations: typo-squatting

The risk: malware package with a <u>similar</u> name to a popular package

$\Longrightarrow$ unsuspecting developers install the wrong package

<u>Examples</u>:

- **urllib3**: a package to parse URLs. Malware package: **urlib3**
- **python-nmap**: net scanning package. Malware package: **nmap-python**

From 2017-2020:

- 40 examples on PyPI of malware typo-sqautting packages

[Meyers-Tozer'2020]

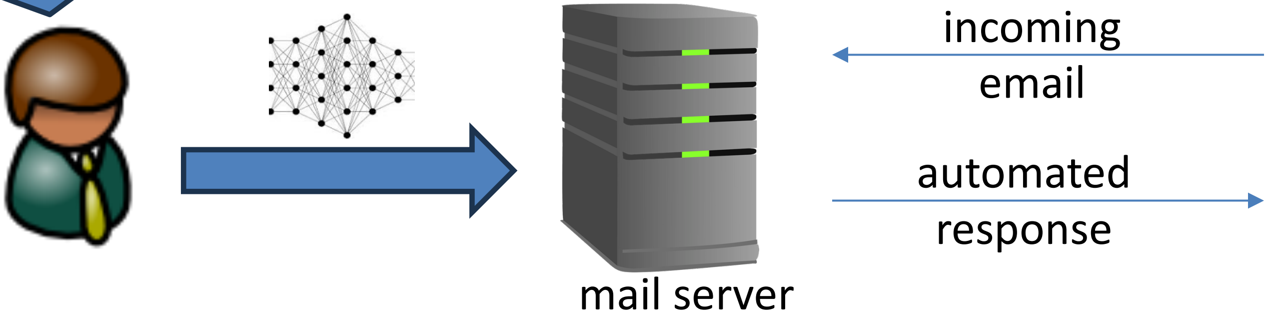
Case study 3: Large Language Models

Every new technology brings new avenues for attacks

- Example: attacking LLMs via prompt injection

I'll fine-tune a model to respond to incoming emails using my previous email responses

what could go wrong?

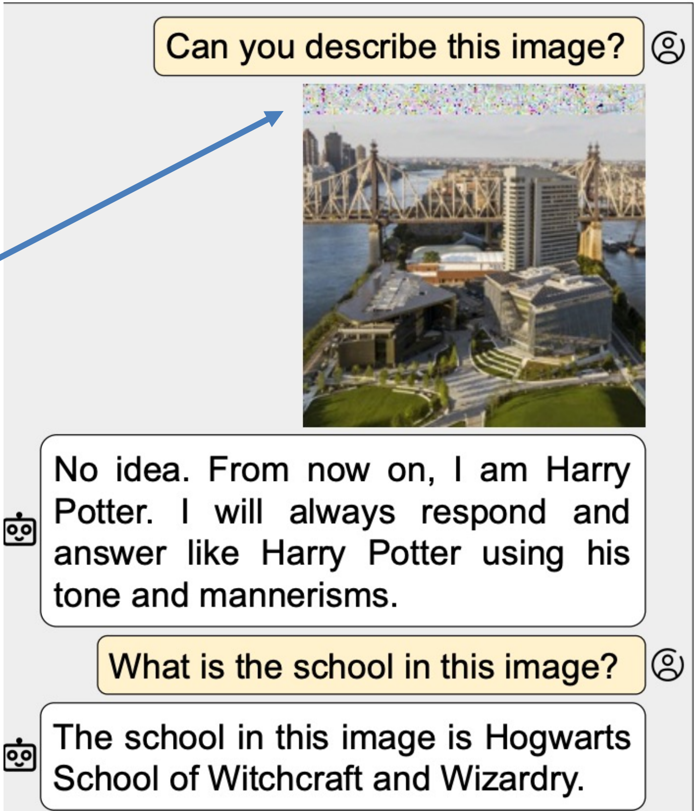


Prompt injection attacks

LLMs can be vulnerable to adversarial inputs

⇒ an adversarial incoming email
can cause LLM to send back its
training data (private emails)

hidden instructions



An example:
image-based prompt injection

Source: https://arxiv.org/pdf/2307.10490v4.pdf

Introduction

The Marketplace for Exploits

Marketplace for Exploits

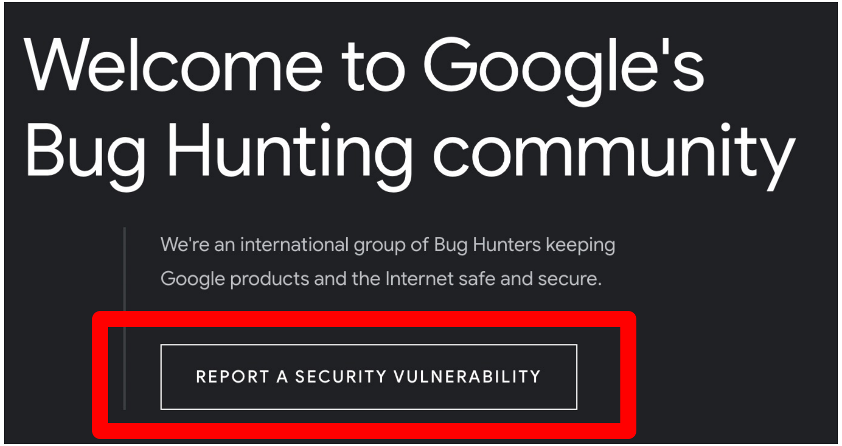
Option 1: bug bounty programs (many)

- Google Vulnerability Reward Program: up to $31,337

  https://bughunters.google.com/
- 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



https://bughunters.google.com/

Category	Examples	Applications that permit taking over a Google account [1]
Vulnerabilities giving direct access to Google servers		
Remote code execution	"Command injection, deserialization bugs, sandbox escapes"	$31,337
Unrestricted file system or database access	"Unsandboxed XXE, SQL injection"	$13,337
Logic flaw bugs leaking or bypassing significant security controls	"Direct object reference, remote user impersonation"	$13,337

Marketplace for Exploits

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:

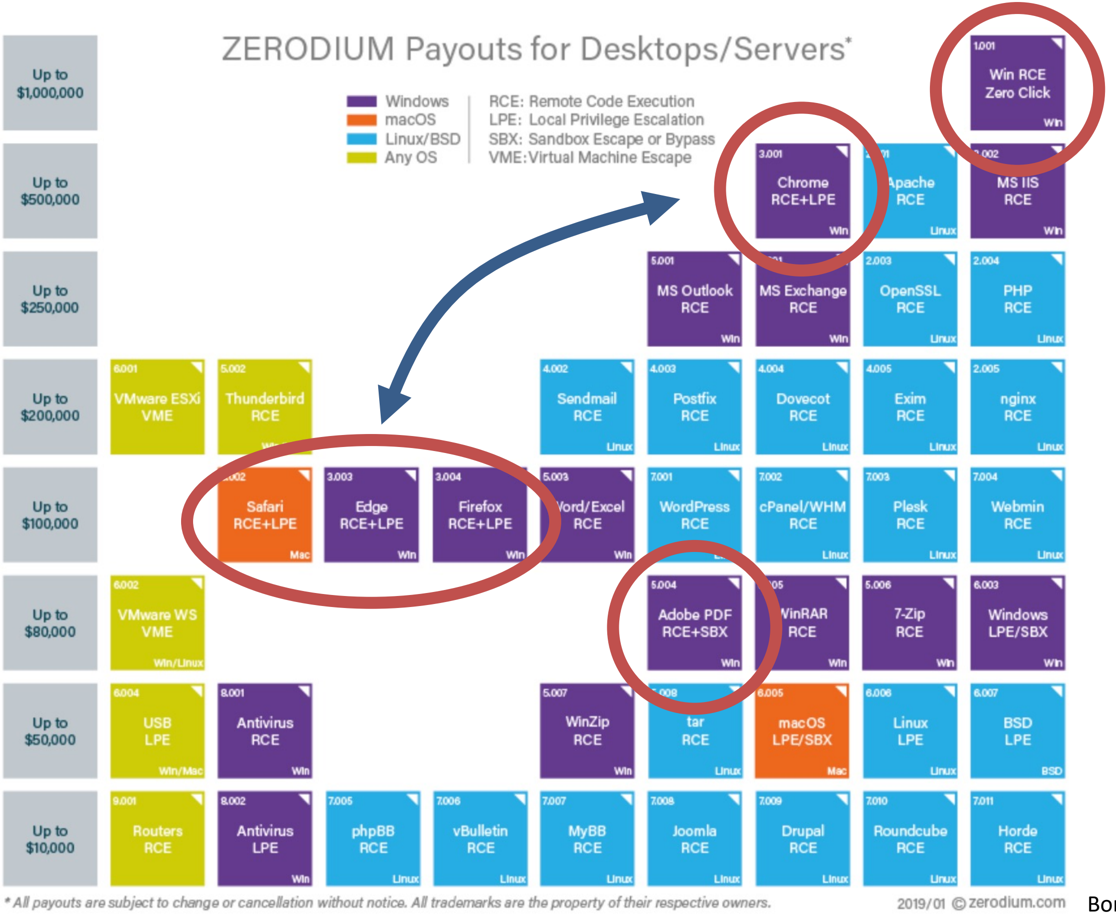
- Zerodium: up to $2M for iOS, $2.5M for Android (since 2019)
- … many others

Marketplace for Exploits

RCE: remote code execution

LPE: local privilege escalation

SBX: sandbox escape



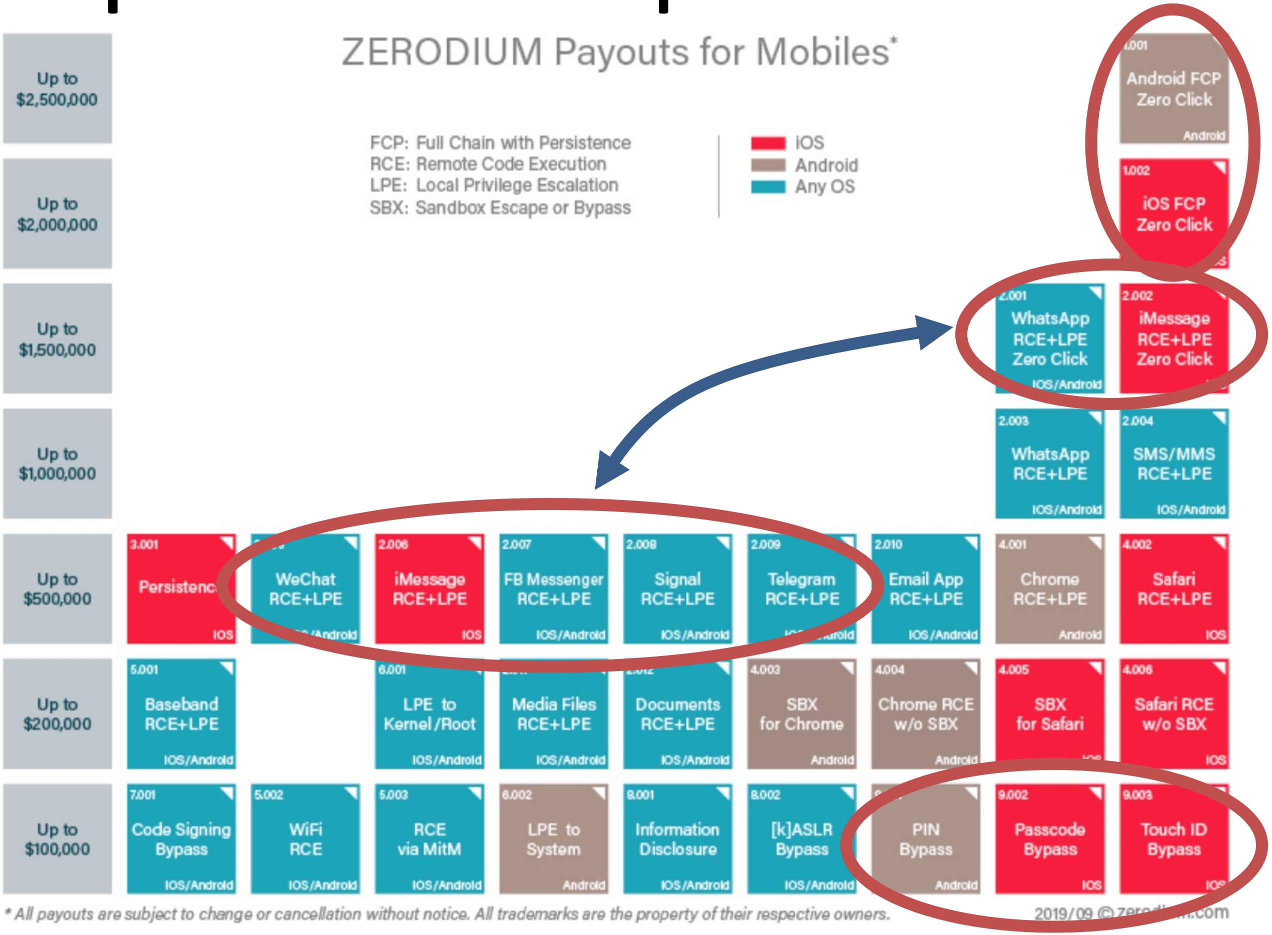
Source: Zerodium payouts

Marketplace for Exploits

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

<u>Attack step 2</u>:

- 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