



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. Please attend first section!**
- Use EdDiscussions and Gradescope
- Automatic 72 hour extension

The computer security problem

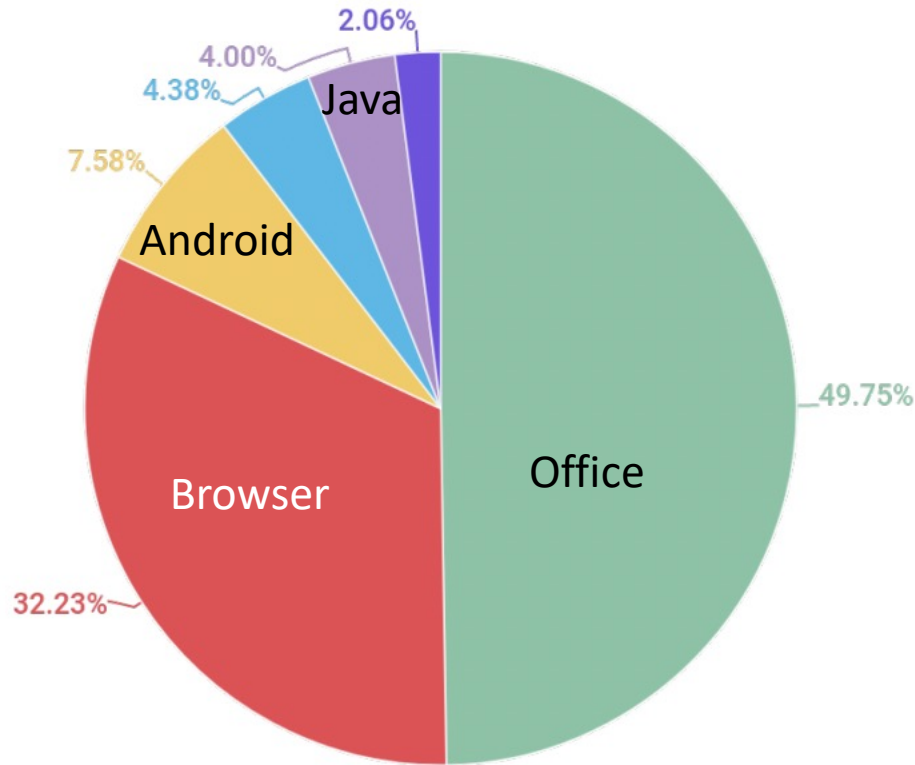
- **Lots of buggy software**
- **Money can be made from finding and exploiting vulns.**
 1. Marketplace for exploits (gaining a foothold)
 2. Marketplace for malware (post compromise)
 3. Strong economic and political motivation for using both

current state of computer security

Top 10 products by total number of “distinct” vulnerabilities in 2021

	Product Name	Vendor Name	Product Type	Number of Vulnerabilities
→	1 Debian Linux	Debian	OS	5922
→	2 Android	Google	OS	4136
→	3 Ubuntu Linux	Canonical	OS	3144
→	4 Mac Os X	Apple	OS	2967
	5 Fedora	Fedoraproject	OS	2885
	6 Linux Kernel	Linux	OS	2789
	7 Windows 10	Microsoft	OS	2621
→	8 Iphone Os	Apple	OS	2604
	9 Windows Server 2016	Microsoft	OS	2359
→	10 Chrome	Google	Application	2346
	11 Windows Server 2008	Microsoft	OS	2169
	12 Windows 7	Microsoft	OS	2035
→	13 Firefox	Mozilla	Application	1993

Distribution of exploits used in attacks



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

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?

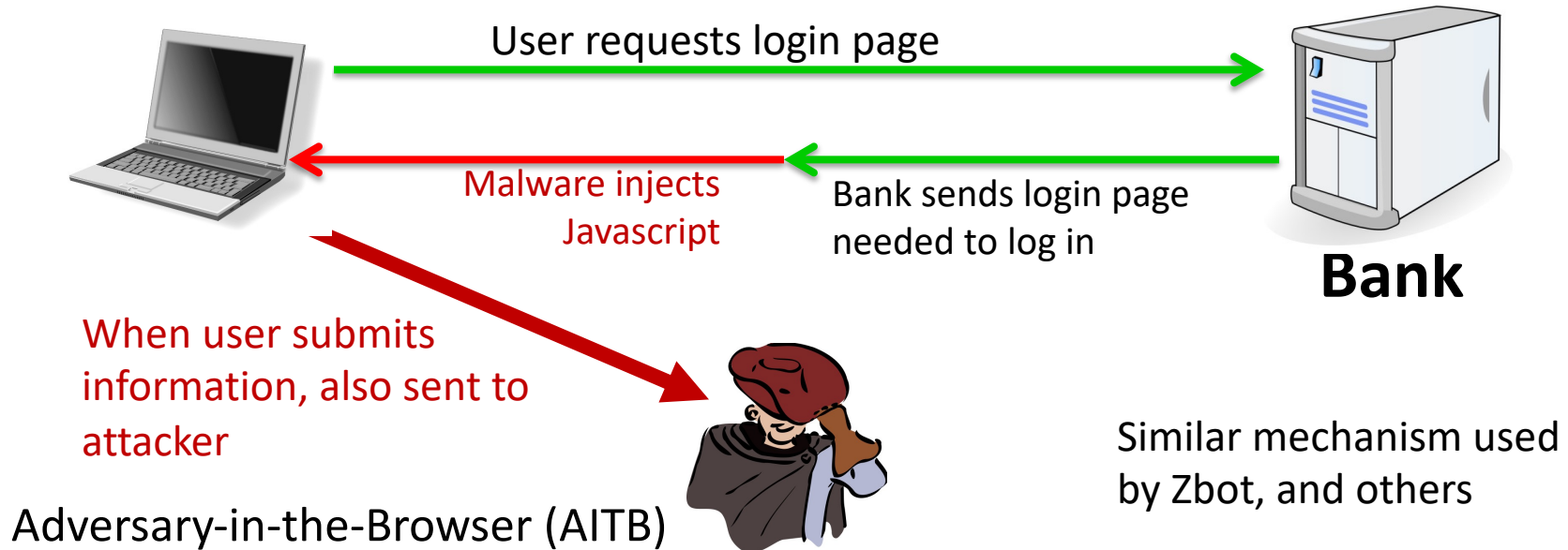
... economics

Why compromise end user machines?

1. Steal user credentials

keylog for banking passwords, corporate passwords, gaming pwds

Example: SilentBanker (and many like it)



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

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/GlobelImposter	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



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

[About bitcoin](#)

[How to buy bitcoins?](#)

[Contact Us](#)

Ooops, your files have been encrypted!

English

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 payment is from 11:00am GMT from Monday to Friday.



Send \$300 worth of bitcoin to this address:

115p7UMMngo1pMvvpHijcRdfJNXj6LrLn

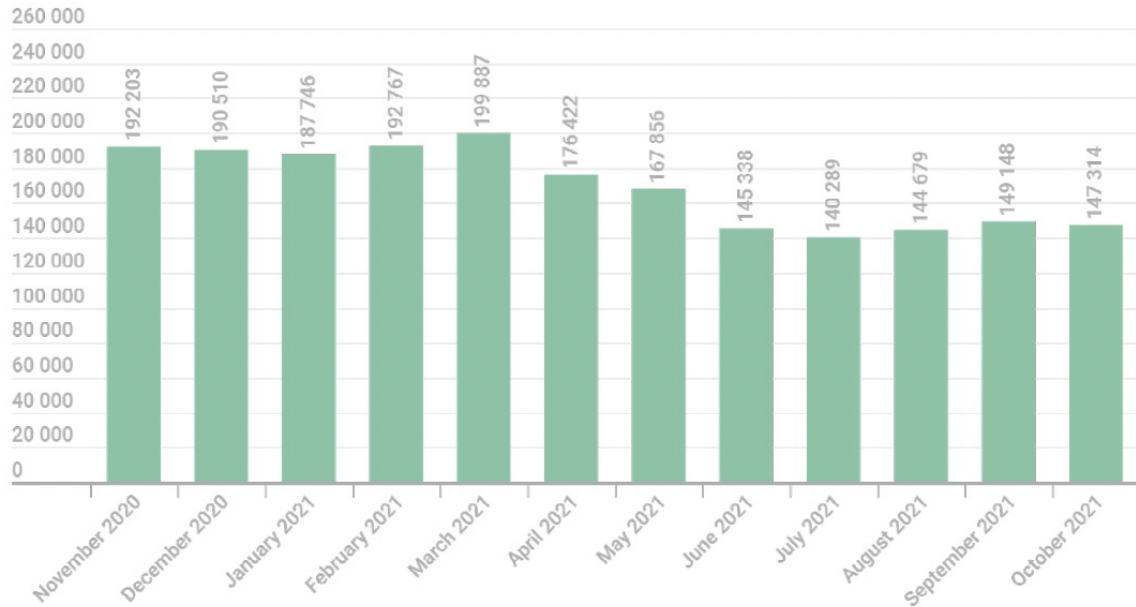
Copy

Check Payment

Decrypt

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

Server-side attacks: why?

(1) Data theft: credit card numbers, intellectual property

- Example: Equifax (July 2017), \approx 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

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

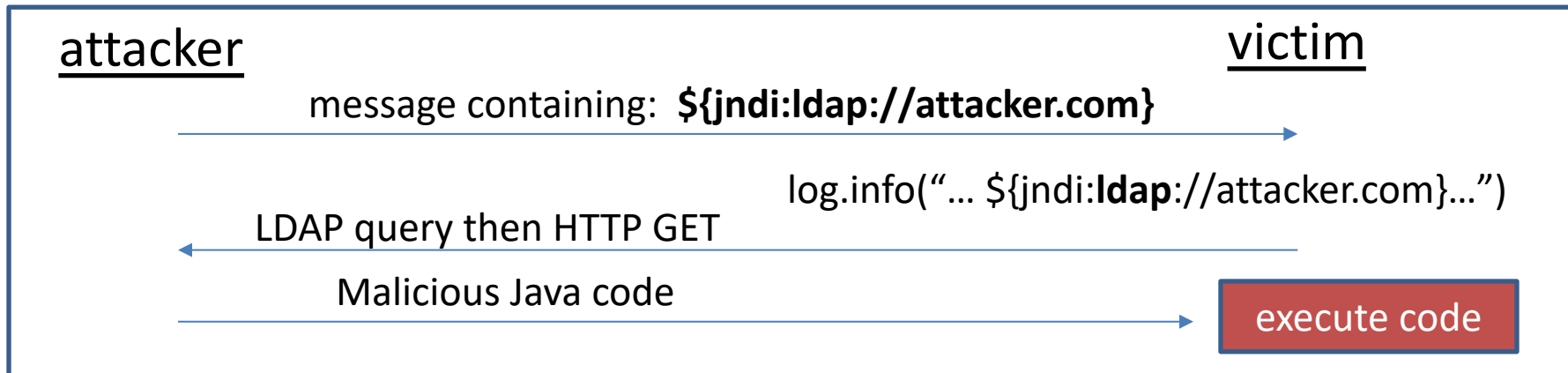
... but no complete solution

Case study 1: Log4Shell (2021)

Log4j: a popular logging framework for Java

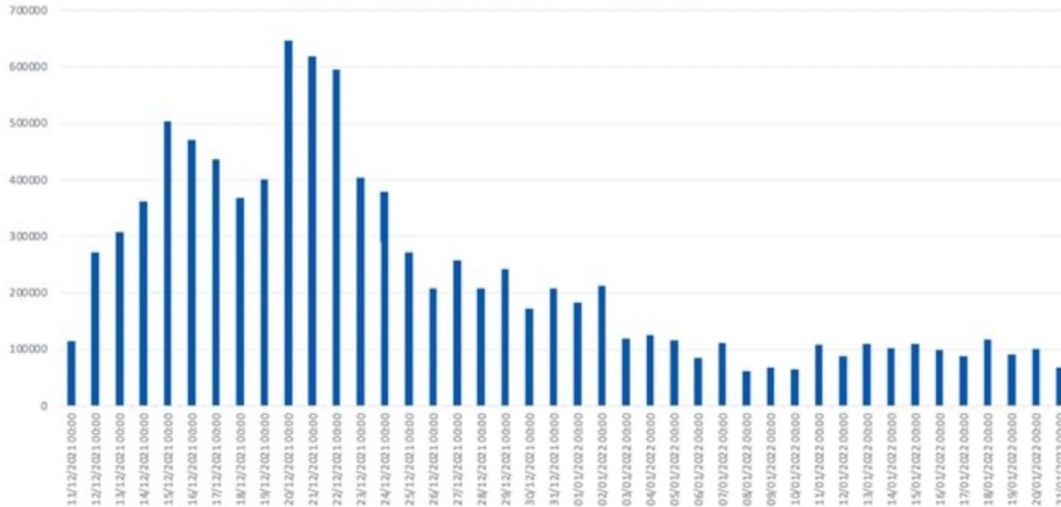
- Nov. 21: vulnerability in Log4j 2 enables **Remote Code Execution**
- Over 7000 code repositories affected and many Java projects

The bug: Log4j can load and run code to process a log request



The result

Log4Shell Attack Attempts Blocked by Sophos XG Firewalls by Date
(Dec. 9, 2021 - Jan. 21, 2022)



How was this exploited?

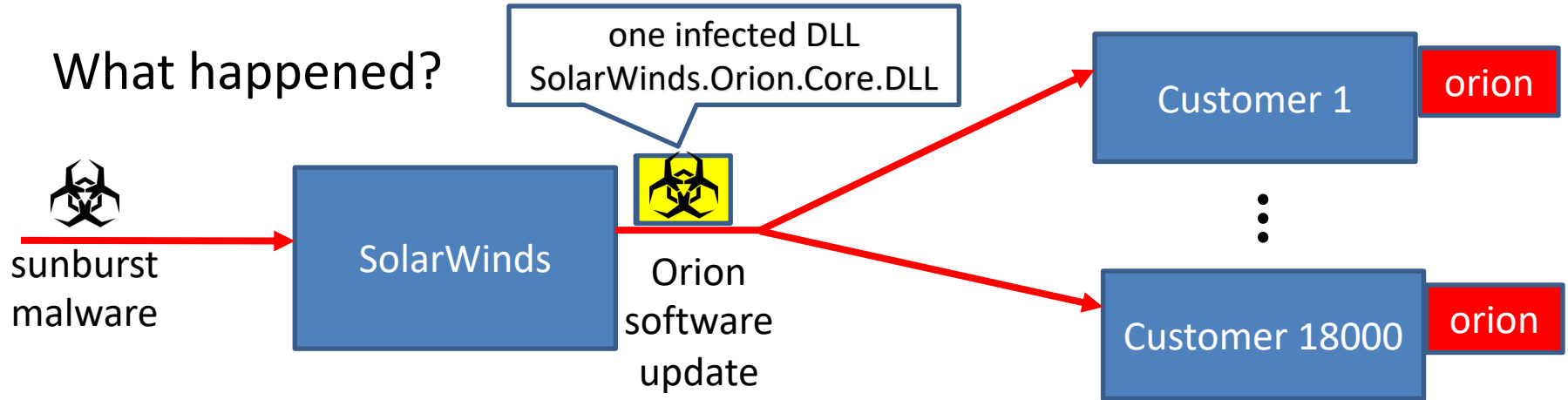
- Khonsari ransomware
- XMRIG Cryptominer
- Orcus Remote Access Trojan

How to prevent problems of this type?

- Isolation: sandbox log4j library or sandbox entire application

Case study 2: SolarWinds Orion (2020)

SolarWinds Orion: set of monitoring tools used by many orgs.



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 ???

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
⇒ many compromised customers
- Many examples of this in the past (e.g., Target 2013, ...)
- Defenses?

Case study 3: 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 \Rightarrow compromise dependent projects

Many examples:

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

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

Security considerations: typo-squatting

The risk: malware package with a similar name to a popular package
⇒ unsuspecting developers install the wrong package

Examples:

- **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-squatting packages

[Meyers-Tozer'2020]



Introduction

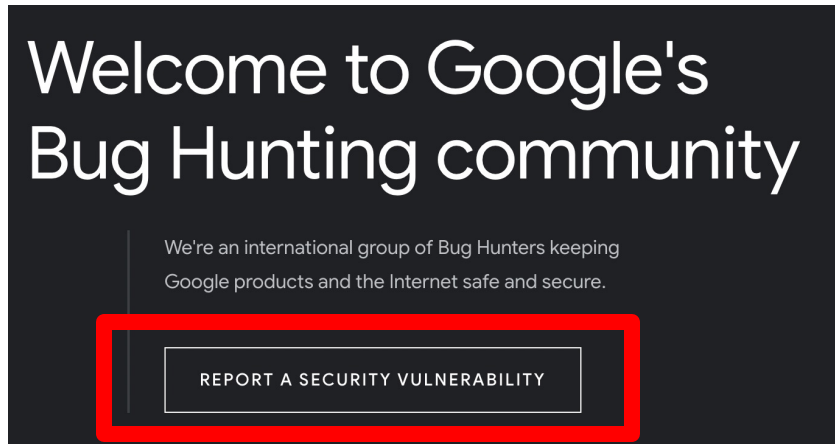
The Marketplace for Vulnerabilities

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

Google's bug bounty program



Welcome to Google's Bug Hunting community

We're an international group of Bug Hunters keeping Google products and the Internet safe and secure.

REPORT A SECURITY VULNERABILITY

<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 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
-

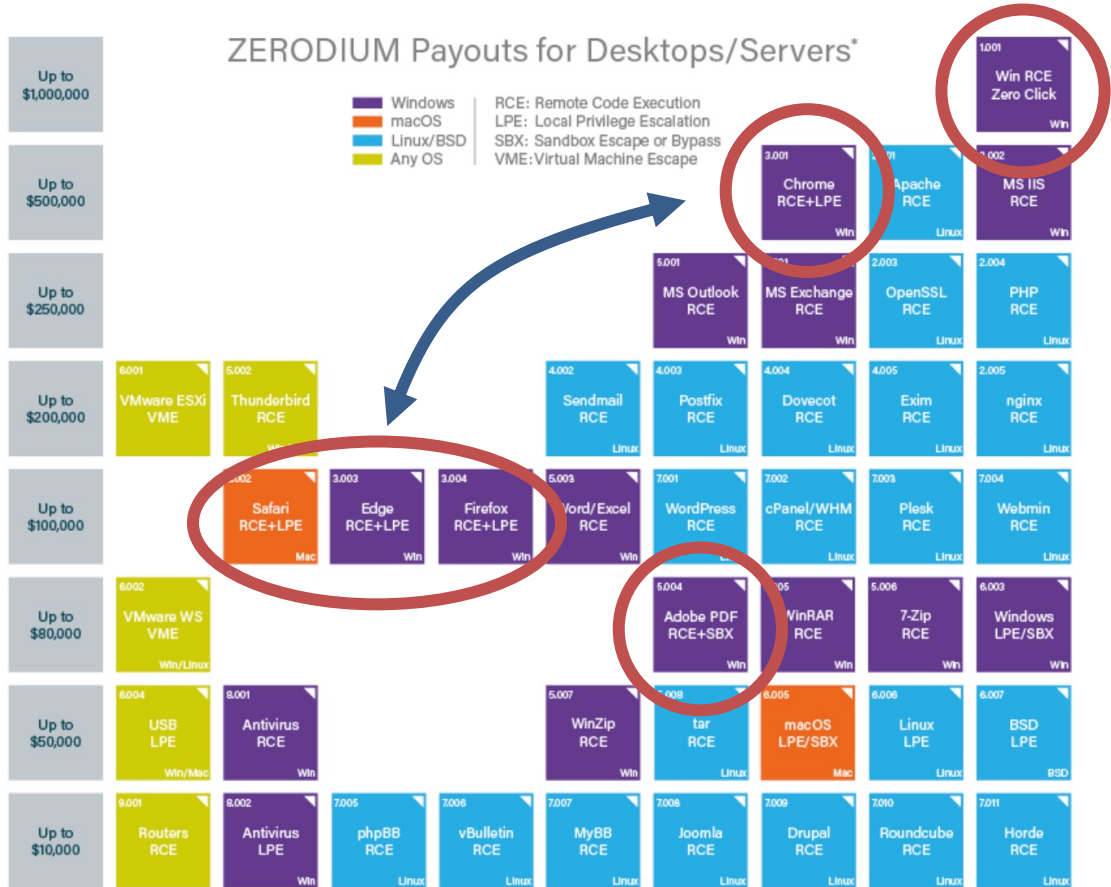
Option 2:

- 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

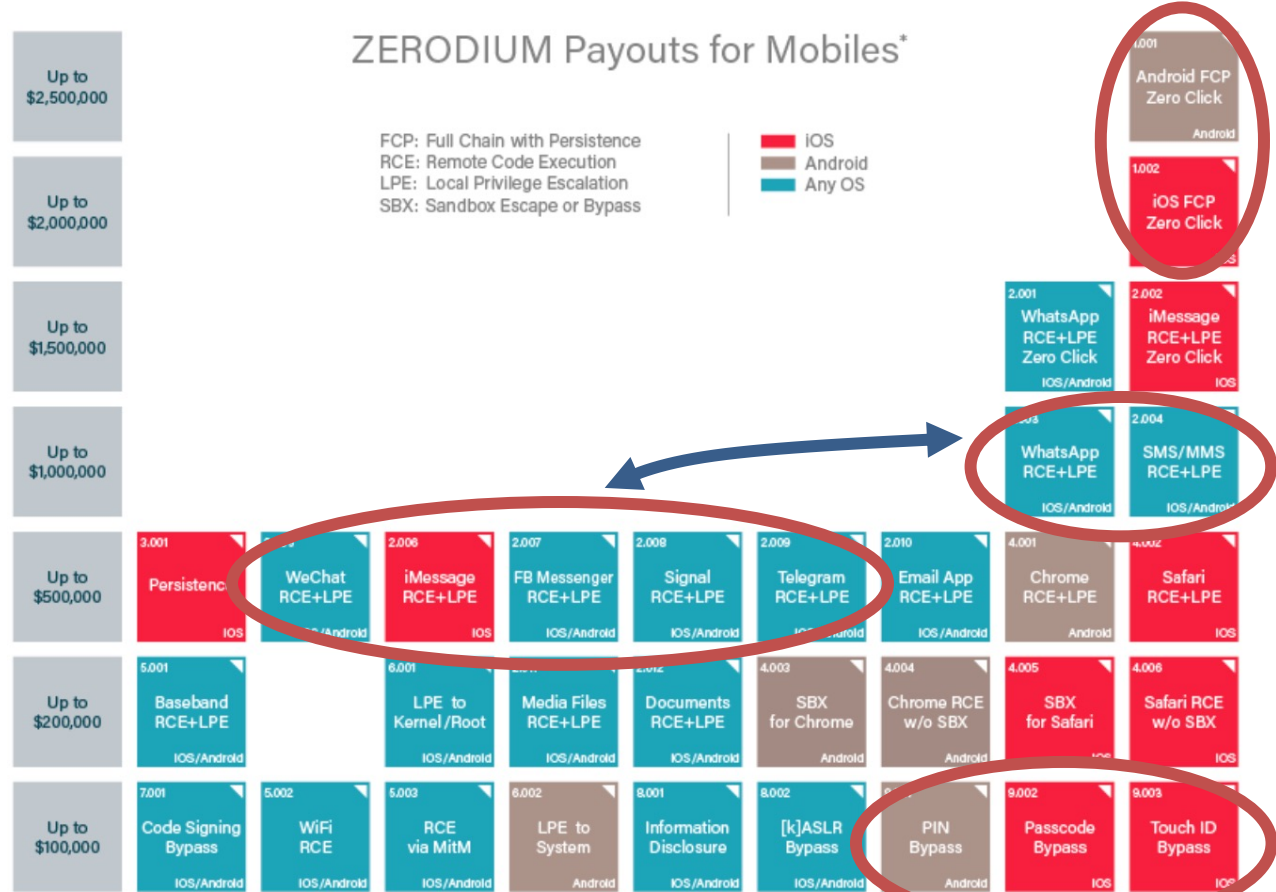
Source: Zerodium payouts



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

Marketplace for Vulnerabilities

RCE: remote code execution
 LPE: local privilege escalation
 SBX: sandbox escape



Source: Zerodium payouts

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

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 \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