Lecture 2: Introduction

Xin Liu

Florida State University
xliu15j@fsu.edu

CIS 5370 Computer Security
https://xinliulab.github.io/cis5370.html
January 8, 2025

The Era We Live In

The Web 2.0 Era (1999)

- The Internet brought people closer together.
- ”Users were encouraged to provide content, rather than just viewing it.”
- You can even find early hints of ”Web 3.0”/Metaverse in this period.

What made Web 2.0 possible?

- Concurrent programming in browsers: Ajax (Asynchronous JavaScript + XML)
- HTML (DOM Tree) + CSS represented everything you could see.
  - JavaScript allowed dynamic changes to the DOM.
  - JavaScript also enabled connections between local machines and servers.

With that, you had the whole world at your fingertips!

Features and Challenges

Features:

- Not very complex
- Minimal computation required
  - The DOM tree is not too large (humans can't handle huge trees anyway)
  - The browser handles rendering the DOM tree for us
- Not too much I/O, just a few network requests

Challenges:

- Too many programmers, especially for beginners
- Expecting beginners to handle multithreading with shared memory would lead to a world full of buggy applications!

Towards the Mobile Internet Era

The Transformation of Our World

We can no longer imagine life without mobile phones.

Enjoy: [Nokia Ringtone Evolution]

Again and Again, Changing the World!

The first smartphone: iphone 2G, 2007.

Android

Android Official Website

- **Linux + Framework + JVM**
  - Is it secondary development on Linux/Java?
  - Not exactly: Android defines an application model.
- Supporting Java was a highly visionary decision.
  - **Qualcomm MSM7201:**
    - ARMv6 instruction set
    - 528MHz x 1CPU, in-order 8-stage pipeline
    - TSMC 90nm
- "Even running a map app would lag..."
  - But Moore's Law came to the rescue!

The first Android phone: HTC G1, 2008

Android Apps: Write Once, Run Anywhere

An application running on the Java Virtual Machine ([Android Runtime]):

- **Platform (Framework)**
- **NDK (Native Development Kit)**
- **Java Native Interface (C/C++ Code)**

Official Documentation (RTFM):

- **Kotlin**
- **Platform**
  - **android.view.View:** “the basic building block for user interface components”
  - **android.webkit.WebView:** Embedding web pages in your app
  - **android.hardware.camera2:** Camera
  - **android.database.database:** Database

Comparison: Symbian (C++) vs. Android (Java)

Symbian (C++)

- **Manual Memory Management**
  - Developers use `malloc/free`, prone to memory leaks.
  - Risks: dangling pointers, segmentation faults.
- **Pointer Operations**
  - Direct pointer manipulation increases error risks.
  - Debugging pointer issues is time-consuming.
- **Complex Syntax**
  - C++ syntax is powerful but hard to maintain.
  - Error handling is not standardized (e.g., exceptions).

Android (Java)

- **Automatic Memory Management**
  - Garbage Collection handles memory allocation and release.
  - Fewer memory leaks and dangling references.
- **No Pointer Exposure**
  - Object references instead of raw pointers.
  - Safer access to memory and resources.
- **Simplified Syntax**
  - Built-in exception handling.
  - Higher-level APIs reduce coding complexity.

Key Takeaways

Symbian (C++):

- Powerful and performance-oriented but requires expert-level skills.
- Higher error rates due to manual memory management and pointer issues.

Android (Java):

- Developer-friendly with built-in safety mechanisms.
- Automatic garbage collection simplifies memory management.
- Encourages faster application development and wider adoption.

Conclusion: Java's design prioritizes developer productivity and safety, making it a better fit for large-scale mobile application ecosystems.

The Strategic Bet on Java and Moore's Law

Challenges at the time:

- Limited processing power on mobile chips.
- Java's higher demands on hardware made it seem risky.

Key Assumption:

- Inspired by Intel's Pentium and multi-core advancements in PCs.
- *What happened in PC chip development will repeat in mobile chips.*

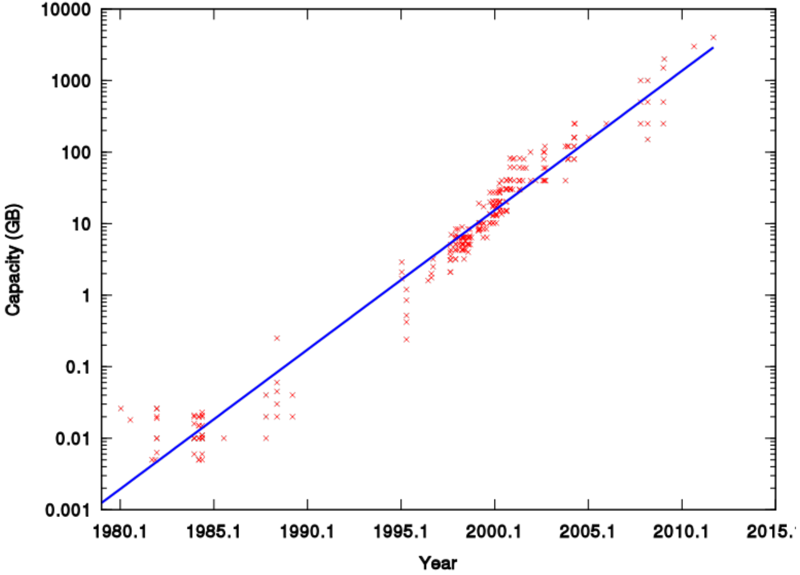
Vision:

- Moore's Law predicted rapid improvement in chip performance.
- As hardware evolved, Java's demands would no longer be a bottleneck.

Lessons from History: Design for the future, not for the present.

The Trend

The Prophet of the Era: Google Is No Coincidence



Redundant Array of Inexpensive Disks (RAID)

Growing Demand for Persistent Storage

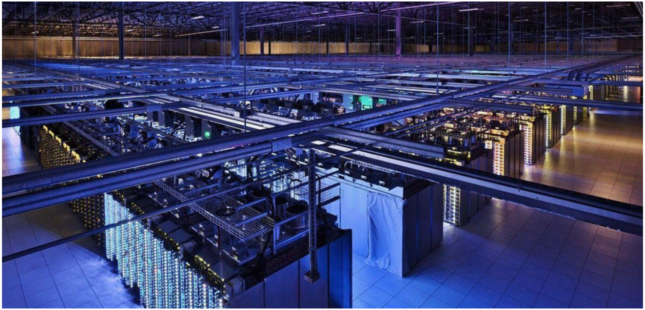
The storage device industry: As long as the CPU (DMA) can handle it, we can provide sufficient bandwidth!

- The tradition of the computer system "industry" — creating practical, efficient, reliable, and cost-effective systems.

EMC VNX5300 Unified Storage (Disk) Array

Performance VS. Reliability

- Any physical storage medium has the potential to fail.
  - Extremely low-probability events:
    - Earthquake, war, alien invasion
  - Low-probability events: Hard drive failure
    - Large-scale redundancy = inevitable occurrence
- But we still hope the system continues running (data integrity even when storage devices fail)



Google Data Center

RAID: Virtualization of Storage Devices

So, can we achieve both performance and reliability?

Redundant Array of Inexpensive (Independent) Disks (RAID)

- Virtualize multiple (unreliable and cheap) disks into a highly reliable and high-performance virtual disk.
  - *A case for redundant arrays of inexpensive disks (RAID)* (SIGMOD '88)

RAID is a "reverse" form of virtualization

- Process: Virtualize one CPU into multiple virtual CPUs
- Virtual Memory: Virtualize one memory unit into multiple address spaces
- File System: Virtualize one drive into multiple virtual drives

RAID Fault Model: Fail-Stop

Disks may suddenly become completely inaccessible at any time.

- Mechanical failure, chip malfunction...
  - The disk seems to ”disappear suddenly” (data completely lost)
  - Assume the disk can report this issue.

The Golden Age in that Era

- 1988: Combine a few disks and disrupt the entire industry!
  - ”Single Large Expensive Disks” (IBM 3380) vs.
  - ”Redundant Array of Inexpensive Disks”

RAID 0 - Disk Striping

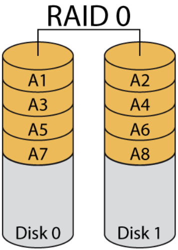
- Breaks data into **striped units** and spreads it across multiple drives.

Advantages:

- **High Performance:** High throughput (parallel I/O)
- **Cost-effective:** Better performance compared to a single large disk with similar capacity.

Disadvantages:

- **No Redundancy:** No backups; data loss occurs if any drive fails.
- **Increased Risk:** More drives mean a higher likelihood of failure.



RAID 0 – Striping

Reliability: Data fetched is what you stored.
Availability: Data is there when you want it.

- More disks means higher probability of some disk failing.
- Striping reduces reliability
  - **N disks:** $1/n$th mean time between failures (MTBF) of 1 disk.

What can we do to improve Disk Reliability?

RAID 1 – Disk Mirroring

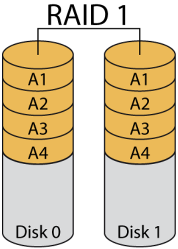
- Duplicates data and stores a copy on each drive (redundancy).
  - Requires at least two drives.
  - If one drive fails, data is still available on the other drive.
  - Supports **hot-swapping**: replace failed drives while the system is running.

Advantages:

- **Data Reliability:** Ensures no data loss even if a drive fails.
- **Redundancy:** Creates a mirror image of your data.

Disadvantages:

- **Storage Overhead:** Only 50% of total capacity is usable.



RAID 1 – Mirroring

Recovering from Failures with Parity

RAID-10: Sometimes can tolerate two disk failures, sometimes not. If we have many disks, can we reduce waste?

Reframe the Question

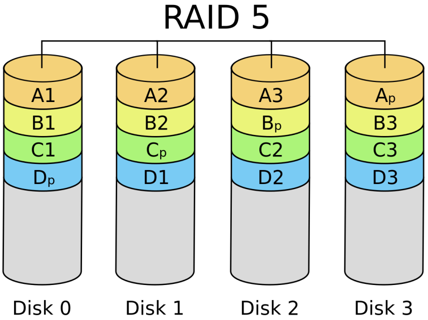
- Given n bits $b_1, b_2, \ldots, b_n$
- How many bits of information do we need to store at minimum so that we can recover any missing b_i (given that we know i)?
  - Parity (or error-correcting code)!
  - $x \oplus x = 0$
- ***n*-input XOR** gives bit-level parity:
  - $1 = \text{odd parity}, \ 0 = \text{even parity}$
- Example:

$$1101 \oplus 1100 \oplus 0110 = 0111 \quad \text{(parity block)}$$

- Can reconstruct any missing block using XOR with others.

RAID 5: Rotating Parity

"Interleaved" parity block!



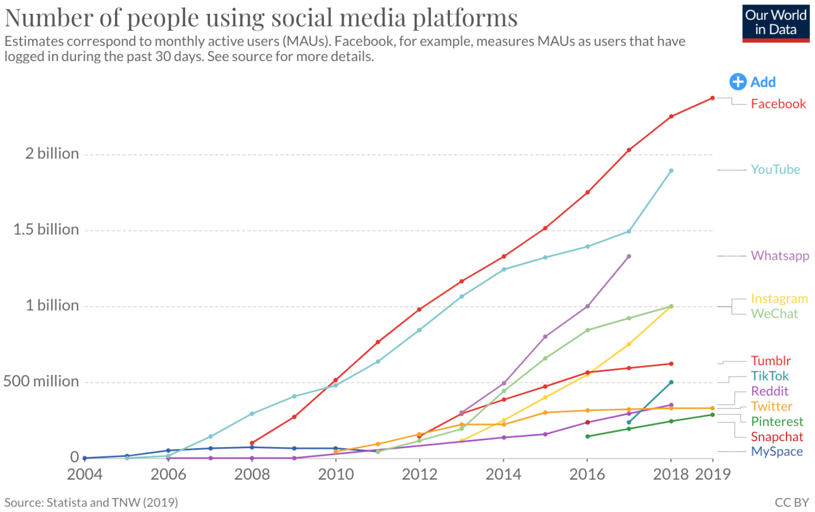
- In RAID 5, parity blocks are distributed across all disks to avoid a single parity disk bottleneck.
- This setup provides fault tolerance while maximizing disk performance.

RAID: Discussion

Faster, more reliable, and nearly free high-capacity disks.

- Revolutionized the concept of ”high-reliability disks”
  - Became the standard configuration for today’s servers
- Similar milestones
  - *The Google File System* (SOSP ’03)
  - *MapReduce: Simplified Data Processing on Large Clusters* (OSDI ’04)
    - Transformed a collection of unreliable, commodity computers into a reliable, high-performance server.
    - Launched the ”Big Data” era!
- What is next?

The Prophet of the Era: Facebook Is No Coincidence



Data Center

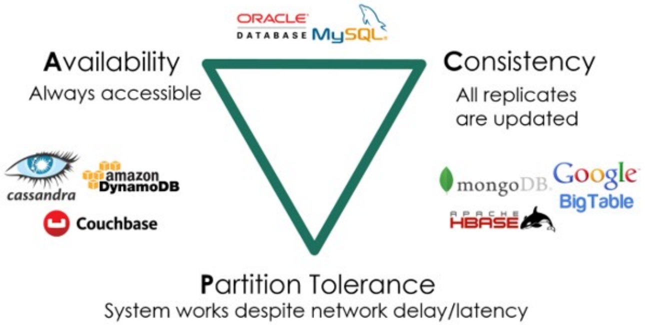
Data Center

"A network of computing and storage resources that enable the delivery of shared applications and data." (CISCO)

- Data-Centric (Storage-Focused) Approach
  - Originated from internet search (Google), social networks (Facebook/Twitter)
  - Powers various internet applications: Gaming/Cloud Storage/ WeChat/Alipay/...
- The Importance of Algorithms/Systems for HPC and Data Centers
  - You manage 1,000,000 servers
  - A 1% improvement in an algorithm or implementation can save 10,000 servers

Recap: Main Challenges of Data Center

- Serving massive, geographically distributed requests
- Data must remain consistent (Consistency)
- Services must always be available (Availability)
    - Must tolerate machine failures (Partition Tolerance)

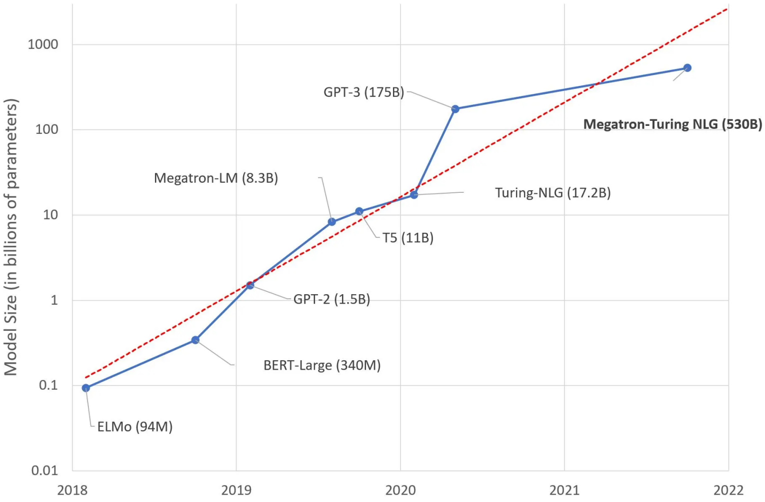


We foucus on a single machine

How to Maximize Parallel Request Handling with a Single Machine

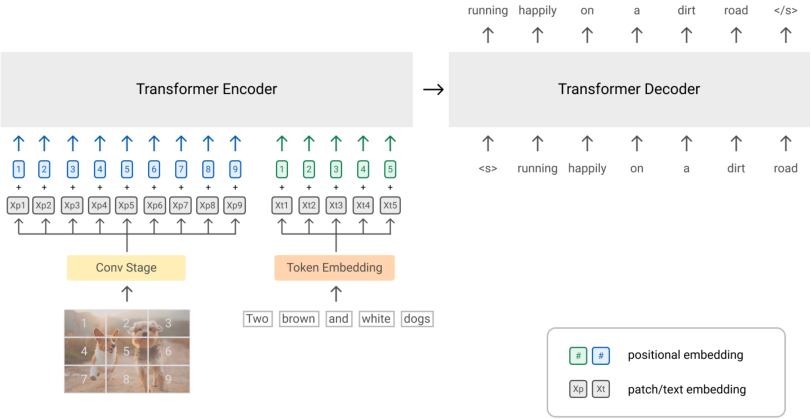
- Key Metrics: QPS, Tail Latency, ...

The Prophet of the Era: OpenAI Is No Coincidence



AGI is approaching rapidly:

- *Everything is an embedding.* (Below: SimVLM)



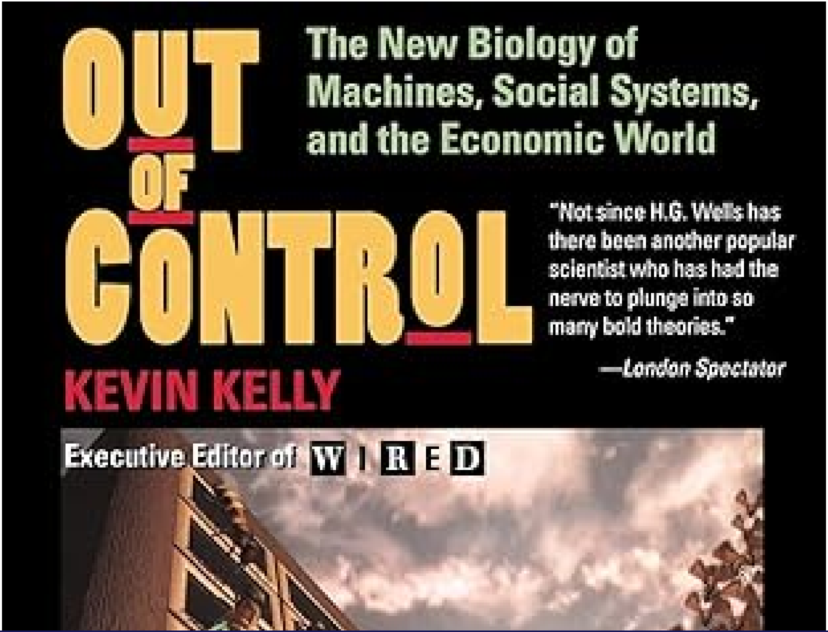
A. C. (Anno. ChatGPT) Year 3

We Begin Using LLMs for Complex Programming Tasks

- Explore GCC/LLVM to uncover 100+ bugs.
  - Use ChatGPT to generate test cases.
  - Use ChatGPT to generate Clang AST Transformer.

```
  class ModifyFunctionReturnTypeToVoid : ... {
-   vector<ReturnStmt *> TheReturns;
-   vector<CallExpr *> TheCalls;
+   map<FunctionDecl *, vector<ReturnStmt *>> FuncReturns;
+   map<FunctionDecl *, vector<CallExpr *>> FuncCalls;
  };

  bool ModifyFunctionReturnTypeToVoid::mutate() {
    TraverseAST(getASTContext());
    if (TheFunctions.empty()) return false;

    FunctionDecl *func = randElement(TheFunctions);

    // Change the return type to void
    QualType voidType = getASTContext().VoidTy;
    std::string voidTypeStr = formatAsDecl(voidType, "");

    SourceRange typeRange =
      func->getReturnTypeSourceRange();
    getRewriter().ReplaceText(typeRange, voidTypeStr);
```

The Era We Live In



Computer Security Problem

- Two factors:
  - Lots of **buggy software** (and gullible users)
  - Money can be made from vulnerabilities

Marketplace

- Marketplace for vulnerabilities
- Marketplace for owned machines (PPI)
- Many methods to profit from owned client machines

current state of computer security

Vulnerabilities and Exploits

- A **bug** is a place where real behavior may **deviate** from expected behavior.
  - A vulnerability is a security-sensitive bug.
- An **exploit** is an **input** that gives an attacker an advantage.

Method	**Objective**
Control flow hijacking	Gain control of the instruction pointer %eip
Denial of service	Cause program to crash or stop servicing clients
Information disclosure	Leak private information, e.g., saved password

Top Targeted High-risk Vulnerabilities

- There are plenty of bugs in common programs
  - Fact: Ubuntu Linux has over **99,000** known bugs
- Old vulnerabilities, even fixed, still exist in user systems
  - CVE-2006-3227 is a top threat after almost **10 years**!

Product	**CVE**
Internet Explorer	CVE-2006-3227, CVE-2009-3674, CVE-2010-0806, CVE-2012-4792, CVE-2013-1347, CVE-2014-0322/1776
Microsoft Office	CVE-2008-2244, CVE-2009-3129, CVE-2010-3333, CVE-2011-0101, CVE-2012-0158/1856, CVE-2014-1761
JAVA	CVE-2012-172, CVE-2013-2465

Source: https://www.us-cert.gov/ncas/alerts/TA15-119A (April 2015)

Vulnerability Lifecycle

- Patch deployment could take a long time to complete
  - Different versions, shared code, compatibility (enterprise)
  - Automated update can reduce the vulnerability life span
- A patch reveals the details of the fixed vulnerabilities
  - **Patch-based exploit generation** is possible

CWE/SANS Top 25 Most Dangerous Software Errors

http://cwe.mitre.org/top25

Insecure Interaction Between Components

Rank	CWE ID	Name
1	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('**SQL Injection**')
2	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('**OS Command Injection**')
4	CWE-79	Improper Neutralization of Input During Web Page Generation ('**Cross-site Scripting**')
9	CWE-434	Unrestricted Upload of File with Dangerous Type
12	CWE-352	**Cross-Site Request Forgery (CSRF)**
22	CWE-601	URL Redirection to Untrusted Site ('**Open Redirect**')

Risky Resource Management

Rank	CWE ID	Name
3	CWE-120	Buffer Copy without Checking Size of Input (**'Classic Buffer Overflow'**)
13	CWE-22	Improper Limitation of a Pathname to a Restricted Directory (**'Path Traversal'**)
14	CWE-494	Download of Code Without Integrity Check
16	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
18	CWE-676	Use of Potentially Dangerous Function
20	CWE-131	**Incorrect Calculation of Buffer Size**
23	CWE-134	**Uncontrolled Format String**
24	CWE-190	**Integer Overflow or Wraparound**

Porous Defenses

Rank	CWE ID	Name
5	CWE-306	**Missing Authentication for Critical Function**
6	CWE-862	Missing Authorization
7	CWE-798	**Use of Hard-coded Credentials**
8	CWE-311	**Missing Encryption of Sensitive Data**
10	CWE-807	**Reliance on Untrusted Inputs in a Security Decision**
11	CWE-250	**Execution with Unnecessary Privileges**
15	CWE-863	Incorrect Authorization
17	CWE-732	Incorrect Permission Assignment for Critical Resource
19	CWE-327	**Use of a Broken or Risky Cryptographic Algorithm**
21	CWE-307	Improper Restriction of Excessive Authentication Attempts

Bugs

(Why hacking is Computer Science?)

Buffer Overflow

```c
#include <stdio.h>

int main(int argc, char **argv)
{
   char buf[8];
   gets(buf);

   printf("%s\n", buf);
   return 0;
}
```

Integer Overflow

```c
nresp = packet_get_int();

if (nresp > 0) {
   response = xmalloc(nresp * sizeof(char*));

   for (i = 0; i < nresp; i++) {
      response[i] = packet_get_string(NULL);
   }
}
```

Integer Error

```c
char* processNext(char* strm)
{
   char buf[512];

   short len = *(short*) strm;
   strm += sizeof(len);

   if (len <= 512) {
      memcpy(buf, strm, len);
      process(buf);

      return strm + len;
   } else {
      return -1;
   }
}
```

Format-string Vulnerability

```
int main(int argc, char **argv)
{
   char buf[128];
   ...
   snprintf(buf, 128, argv[1]);
}
```

SQL Injection

```
SELECT * FROM Users
WHERE Username='$username' AND Password='$password'
```

Sony PS3 Epic Fail

Random number generator for ECDSA:

```
int getRandomNumber()
{
    return 4;
}
```

Apple Goto Fail

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
   SSLBuffer signedParams, uint8_t *signature, UInt16
       signatureLen)
{
   OSStatus err;
   ...

   if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) !=
       0)
      goto fail;

   if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) !=
       0)
      goto fail;
      goto fail; // Extra goto fail

   if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
      goto fail;
   ...

fail:
```

Computer Security Problem

- Lots of **buggy software** (and gullible users)
- Money can be made from vulnerabilities

- Marketplace for vulnerabilities
- Marketplace for owned machines (PPI)
- Many methods to profit from owned client machines

current state of computer security

Steal IP Address and Bandwidth

Attacker's goal: look like a random Internet user

Use the IP address of infected machine or phone for:

- Spam (e.g., the storm bonnet)
  - Spamalytics: 1:12M pharma spams lead to purchase
  - 1:260K greeting card spams lead to infection
- Denial-of-service
  - Services: 1 hour – $20, 24 hours – $100
- Click fraud (e.g., clickbot.a)

Steal User Credentials and Inject Ads

- Keylogger for banking/web/gaming passwords
- Example: SilentBanker (and many others like it e.g., Zeus bot)

Spread to Isolated Systems

Example: Stuxnet

Windows infection $\rightarrow$
Siemens PCS 7 SCADA control software on Windows $\rightarrow$
Siemens device controller on isolated network

Server-side Attacks

- **Financial data theft** (credit card numbers)
  - Example: Target attack (2013), $\sim$140M CC numbers stolen
  - Many similar attacks since 2000
- **Political motivation**
  - Aurora (2009), Tunisia Facebook (2011), GitHub (Great Cannon 2015)
- **Infect visiting users** (waterhole attacks)

Marketplace for Vulnerabilities

Marketplace for Vulnerabilities

- **Option 1:** bug bounty programs (many)
  - Google Vulnerability Reward Program: up to 100K $
  - Microsoft Bounty Program: up to 100K $
  - Mozilla Bug Bounty program: 500$ - 3000$
  - Pwn2Own competition: 15K $
- **Option 2:**
  - ZDI, iDefense: 2K - 25K $

Marketplace for Vulnerabilities

Option 3: black market

- Not really an option for ethical hackers

Software	**Price Range**
Adobe Reader	$5,000–$30,000
Mac OSX	$20,000–$50,000
Android	$30,000–$60,000
Flash or Java Browser Plug-ins	$40,000–$100,000
Microsoft Word	$50,000–$100,000
Windows	$60,000–$120,000
Firefox or Safari	$60,000–$150,000
Chrome or Internet Explorer	$80,000–$200,000
iOS	$100,000–$250,000

Source: Andy Greenberg (Forbes, 3/23/2012)

Marketplace for Owned Machines

Pay-per-install (PPI) services:

- Install client's malware on owned machines for a fee

PPI operation:

- Own victim's machine
- Download and install client's malware
- Charge client

US: 100–180$/1000 machines
Asia: 7–8$/1000 machines