https://brown-csci1660.github.io

CS1660: Intro to Computer Systems Security
Spring 2026

Lecture 18: OS Security IV

Instructor: **Nikos Triandopoulos**

April 7, 2026

BROWN

CS1660: Announcements

- Course updates
  - Completed: Project 1, project 2, **project 3**, HW1, **HW2**, midterm
  - To be completed
    - Project 4, HW3, HW4, final
    - 3 + 1 weeks left
      - April 7, 9: Software Security + Network Security
      - April 14, 16: Network Security
      - April 21, 23: Network Security + Special Topics
      - April 28: Revision
      - April 30: Final exam

Last class

- Cryptography
- Authentication
- Web security
- OS security
  - Access control, OS access control, file-system access control

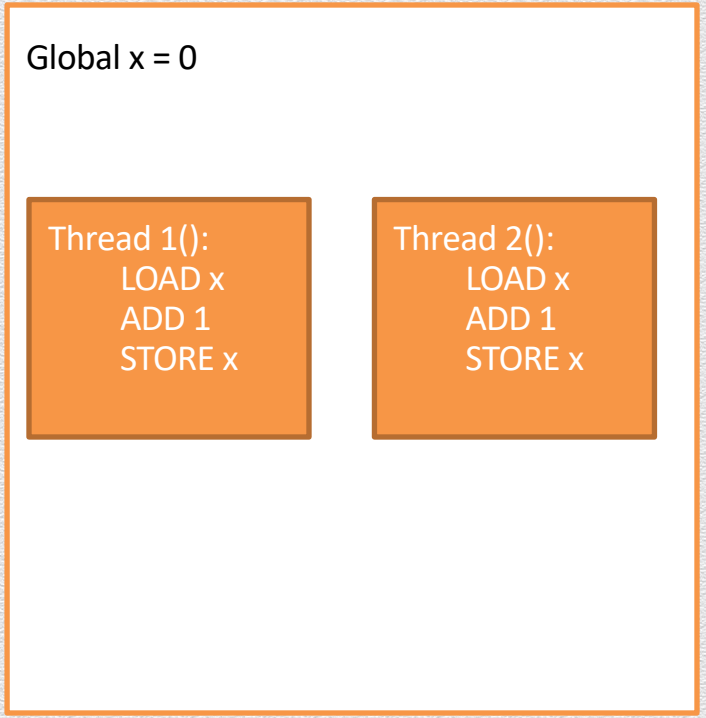
Today

- Cryptography
- Authentication
- Web security
- OS security
  - Access control, OS access control, file-system access control
  - Software security topics: Isolation, malware, cloud security
- Network security
  - Background

18.1 More on race conditions

Race condition

- A race condition occurs when two threads want to access the same memory
- Run Thread 1() and Thread 2()
  - Outcome is 1 or 2



Race condition

```
1.  if (!access("/tmp/X", W_OK)) {
        /* the real user ID has access right */
2.      f = open("/tmp/X", O_WRITE);
3.      write_to_file(f);
        }
    else {
        /* the real user ID does not have access
        right */
4.      fprintf(stderr, "Permission denied\n");
        }
```

- Fragment of setuid program that writes into file /tmp/X on behalf of a user who created it
- access verifies permission of real user ID
  - Transparently follows symlinks
- open verifies permission of effective user ID
  - Transparently follows symlinks
- What can go wrong?

TOCTOU vulnerability

```
1.  if (!access("/tmp/X", W_OK)) {
        /* the real user ID has access right */
2.      f = open("/tmp/X", O_WRITE);
3.      write_to_file(f);
        }
    else {
        /* the real user ID does not have
        access right */
4.      fprintf(stderr, "Permission denied\n");
        }
```

- What can go wrong?
  - In between (1) and (2), user could replace /tmp/X with symlink to /etc/passwd
  - Not easy to accomplish (timing)
- Example of time of check to time of use (TOCTOU) vulnerability

Attempt to fix the race condition

```
1. lstat("/tmp/X", &statBefore);
2. if (!access("/tmp/X", O_RDWR)) {
3.    int f = open("/tmp/X", O_RDWR);
4.    fstat(f, &statAfter);
5.    if (statAfter.st_ino == statBefore.st_ino) {
        /* the I-node is still the same */
6.     write_to_file(f);
       }
7.    else perror("Race Condition Attacks!");
       }
8. else fprintf(stderr, "Permission denied\n");
   }
```

- lstat and fstat access file descriptor for a path, which includes unique file ID (st_ino)
  - lstat does not traverse symlink
  - fstat accesses descriptor of open file, after symlink traversed by open
- Step (5) compares IDs of
  - file checked in (1) and
  - file opened in (3)
- Check-use-check_again approach
  - Defeats swapping in symlink between access and open
- Fails also if /tmp/X is a symlink when (2) is executed

Does the fix work?

```
1. lstat("/tmp/X", &statBefore);
2. if (!access("/tmp/X", O_RDWR)) {
3.    int f = open("/tmp/X", O_RDWR);
4.    fstat(f, &statAfter);
5.    if (statAfter.st_ino == statBefore.st_ino) {
         /* the I-node is still the same */
6.     write_to_file(f);
       }
7.    else perror("Race Condition Attacks!");
      }
8. else fprintf(stderr, "Permission denied\n");
   }
```

- New attack
  - Before (1) /tmp/X is a hard link to /etc/passwd
  - Between (1) and (2) swap in hard link to user-owned file
  - Between (2) and (3) swap in again hard link to /etc/passwd
- This passes the ID check in (5) and allows the user to write to /etc/passwd

Negative result

- Assumptions
  - Setuid program
  - Path-based permission check for real user ID via syscall access(path, permission) that returns 0 or -1
  - No atomic check-and-open file syscall
- Theorem
  - Program is vulnerable to TOCTOU race condition
- Proof
  - Attacker can always swap good file before access and bad file after access
  - lstat/fstat do not help since they are path-based as well
- Reference
  - Drew Dean, Alan J. Hu: Fixing Races for Fun and Profit: How to Use access (2). USENIX Security Symposium, 2004.

Mitigating and eliminating race conditions

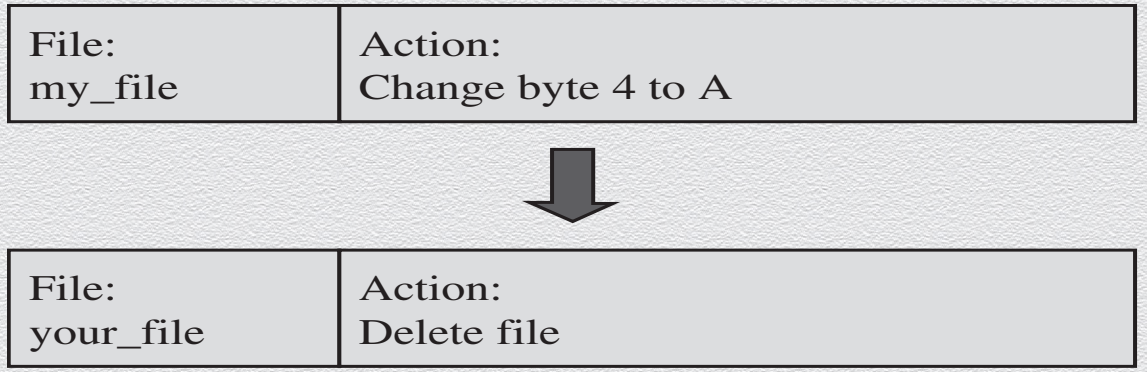
- Hardness amplification
  - Force the adversary to win a large number of races instead of just one or two in order to exploit the vulnerability
  - Reduces the probability of success
  - Complex to accomplish correctly
  - Reference: Dan Tsafrir, Tomer Hertz, David Wagner, Dilma Da Silva: Portably Solving File TOCTTOU Races with Hardness Amplification. USENIX File and Storage Technologies, 2008
- Temporary privilege downgrade
  - Within same process
    - Drop to real user ID privileges via setuid(real_userid)
    - Open file
    - Restore root privileges
  - With child process
    - Fork child process with real user ID privileges to open file
  - Approach not portable across Unix variants

Incomplete mediation

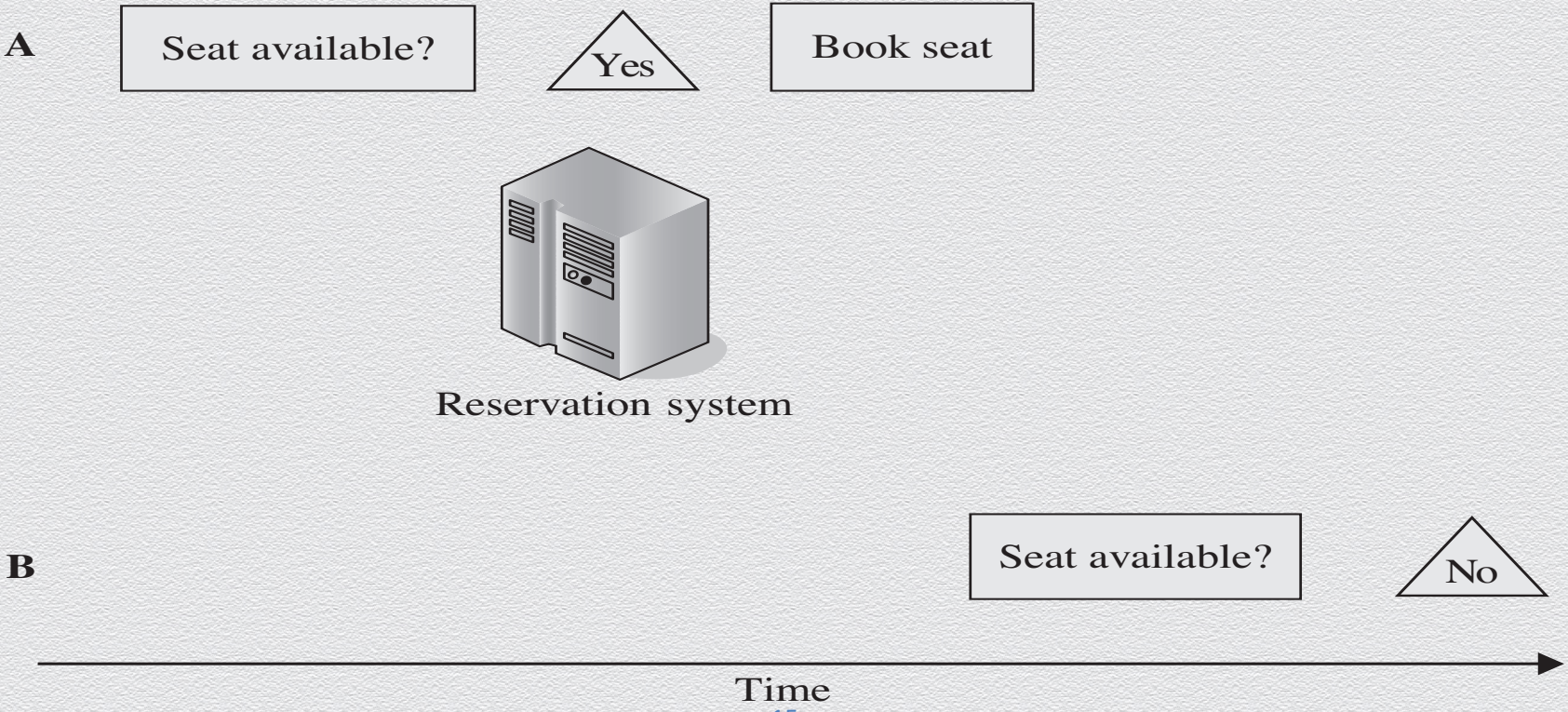
- Access control
  - what subject can perform what operation on what object
- Mediation (means checking)
  - verifying that the subject is authorized to perform the operation on an object
- Preventing incomplete mediation
  - validate all input
  - limit users' access to sensitive data and functions
  - complete mediation using a reference monitor
    - access control that is always invoked, tamperproof and verifiable

Time-of-Check to Time-of-Use

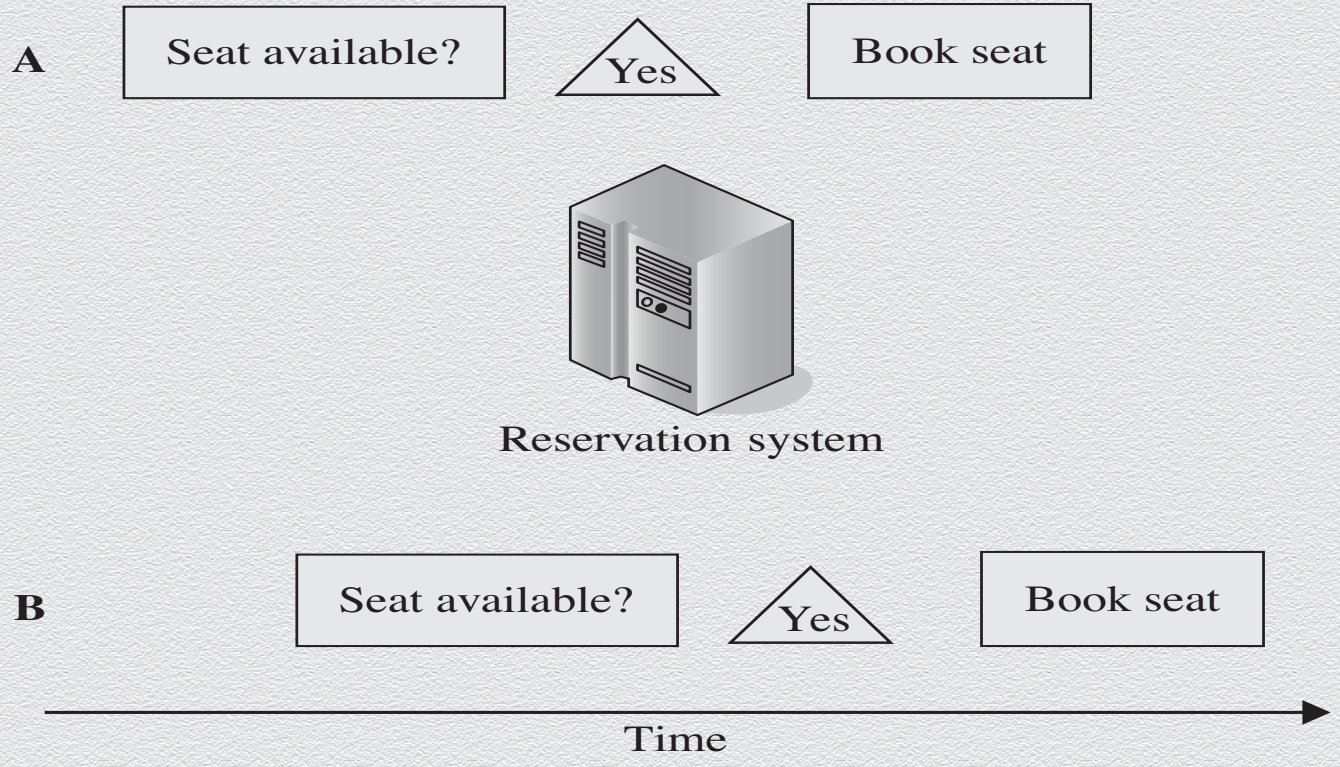
- mediation performed with a “bait and switch” in the middle
- between access check and resource use, data should remain unchanged
- exploits the details in the two processes



Race conditions



Race conditions



Other programming oversights

- Undocumented access points (backdoors)
- Off-by-one errors
- Integer overflows
- Un-terminated null-terminated string
- Parameter length, type, or number errors
- Unsafe utility libraries

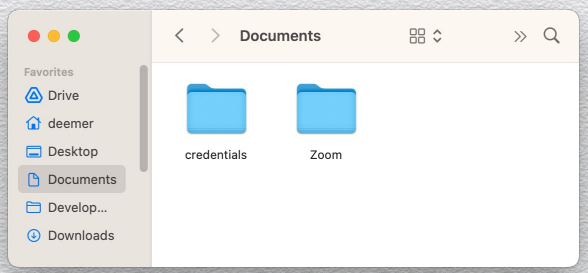
18.2 Sandboxing

Linux default AC model: Discretionary Access Control

- Owner of a resource decides on how it can be used
- Privileges depend on current user (and some groups)
- To elevate, carefully discriminate between admin user (root) vs. other users
  - E.g., setuid, setgid, …

Yet, things can be complicated

- How many applications can read your browser history?



Yet, things can be complicated (cont.)

- Probably all of them…
- Just a syscall! Works as long as permissions check out…

```
deemer@ceres$ ls la ~/.mozilla/firefox/Standard/cookies.sqlite
-rw-r--r-- 1 deemer deemer 524288 Jan 10  2023 cookies.sqlite

deemer@ceres$ sqlite3 ~/.mozilla/firefox/Standard/cookies.sqlite
SQLite version 3.44.2 2023-11-24 11:41:44
Enter ".help" for usage hints.
sqlite> .tables
moz_cookies
```

```
deemer@ceres:~$ strace -- sqlite3 cookies.sqlite
. . .
access("cookies.sqlite", F_OK)          = 0
openat(AT_FDCWD, "cookies.sqlite", O_RDONLY) = 3
. . .
```

Why…

It happens?

- File permissions are very coarse

It matters?

- Applications might not be trusted
- Applications might get compromised

Need for a more secure design

- Restrict application privileges so they can only access what they need

Principle of Least Privilege

Goal

- An application should only be able to perform only the operations that are necessary for its intended purpose…

Means

- It depends on context
- Affects design of different systems or consideration of different abstractions

Linux capabilities

Offer finer control than just “root vs. non-root” by bestowing processes with more precise permissions for certain actions

- API to start processes or threads with or without certain privileges
- Possible to “drop” permissions for unsafe operations (e.g., untrusted inputs to webservers, sshd,…)
- Once permissions are dropped, process can’t get them back

```
CAPABILITIES (7)

DESCRIPTION
       Starting with Linux 2.2, Linux divides the privileges traditionally associated with superuser into
distinct units, known as capabilities, which can be independently enabled and  disabled
   Capabilities list

       CAP_AUDIT_WRITE (since Linux 2.6.11)
              Write records to kernel auditing log.
       CAP_NET_ADMIN
              Perform various network-related operations
       CAP_SYS_BOOT
              Use reboot(2) and kexec_load(2).
                                                    . . .
```

Process separation

- System service runs as privileged user
- Client program run by unprivileged users
- Some API for how these programs communicate
  - Local network connection
  - Unix socket
  - dbus or other IPC mechanism
  - ...
- Better control over how privileged code runs
  - Interface between privileged/unprivileged defined more clearly

Isolation within OS

Separation of processes, memory, and resources

- Prevents one program from interfering with or accessing another
- System stability and security via virtual memory and protected spaces (sandboxes)

Linux namespaces (+ related features)

- give processes/users separate views of user space components

Examples

- schroot (separate filesystem trees), processes trees, …
- UIDs/GIDs, cgroups (resource limits/quotas for CPU, memory, and network bandwidth)

Example: chroot (1980s)

- "Change root"
- Run command with separate root directory
- All child processes inherit this root directory

Example: Containers (e.g., docker)

```
[root@ceres run]# ls -la /run/docker.sock
srw-rw---- 1 root docker 0 Jan  4 07:26 /run/docker.sock


deemer@ceres$ id
uid=1000(deemer) gid=1000(deemer) groups=1000(deemer),...,966(docker),...
```

```
[root@ceres run]# ps aux | grep docker
root        1417  0.0  0.1 4350944 80252 ?       Ssl  Jan04  87:22
        /usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock
. . .

deemer    309604  0.0  0.0  12300   512 ?        S+   Feb26   0:00 /bin/bash
        /home/deemer/cs1660/env/run-container
```

Example: Containers (cont.)

Automated way to run applications

- Leverages lots of Linux namespaces at once; great for deploying software
- Separate filesystem
- Separate UIDs/GIDs
  - Can be root in the container
- Separate network interfaces, etc.
- Isolation mediated by Docker and OS kernel
  - When running the container, we decide what resources are shared with the host (files, network, etc.)

Example: Containers (cont.)

What does it also mean?

- Easy to "punch holes" depending on configuration
  - Shared directories, "privileged containers", ...
- Namespaces are growing all the time
- Docker has lots of permissions levels for what privileges containers can use

What if the configuration is incorrect?

What if the kernel has a bug?

What if you don’t trust the software you’re running?

Virtual Machines (VMs)

Isolated way to run an entire system (hardware, kernel, ...)

- A whole OS could run as a program

Modern systems

- Hardware support for isolating memory, page tables, etc. and preserving performance
- Virtual hardware/drivers to interact with host

"Stronger" isolation

- Possibly more overhead for configuration/performance vs. containers

So where should we run our untrusted code?

- Functionality
  - What privileges should the code (or the user) have?
- Threat model
  - What are the attacker's capabilities?
- Other considerations
  - What does it mean for the user to be "unprivileged"?
  - What does it mean for code run by a user to be "unprivileged"?
  - What do we want that code to be able to do?
  - How much do we trust the user? The code?

Docker on Windows, Mac?

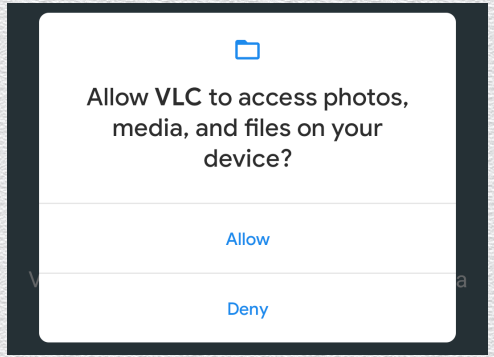
Windows/Mac don't have Linux namespaces...

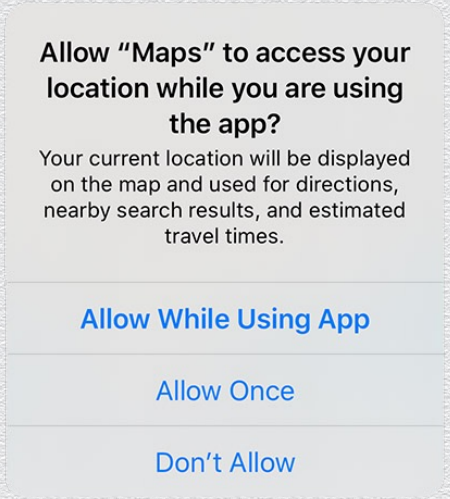
- E.g., Docker on Mac uses a Linux VM to run containers

Isolation mechanisms: Comparison

Mechanism	"Interface" to privileged operations
setuid/setgid application	Application code
Process isolation (client/server process)	API between client program and service (network protocol, socket file, IPC calls, ...)
Container	OS kernel (+ any host features turned on by container author)
VM	Virtualization platform (hypervisor, virtual device drivers, shared folders, ...)

Fine-grain permissions at run time…





18.3 Malware

Malware

- Programs planted by an agent with malicious intent
  - to cause unanticipated or undesired effects
- Virus
  - a program that can replicate itself
    - pass on malicious code to other non-malicious programs by modifying them
- Worm
  - a program that spreads copies of itself through a network
- Trojan horse
  - code that, in addition to its stated effect, has a second, nonobvious, malicious effect

Viruses, Worms, Trojans, Rootkits

Malware

- Software that is specifically designed to disrupt, damage, or gain unauthorized access to a computer system – breaks Confidentiality, Integrity, and Availability
- It can be classified into several categories, depending on propagation and concealment

Propagation

- Virus: human-assisted propagation (e.g., open email attachment)
- Worm: automatic propagation without human assistance

Concealment

- Rootkit: modifies operating system to hide its existence
- Trojan: provides desirable functionality but hides malicious operation (i.e. *payload*)

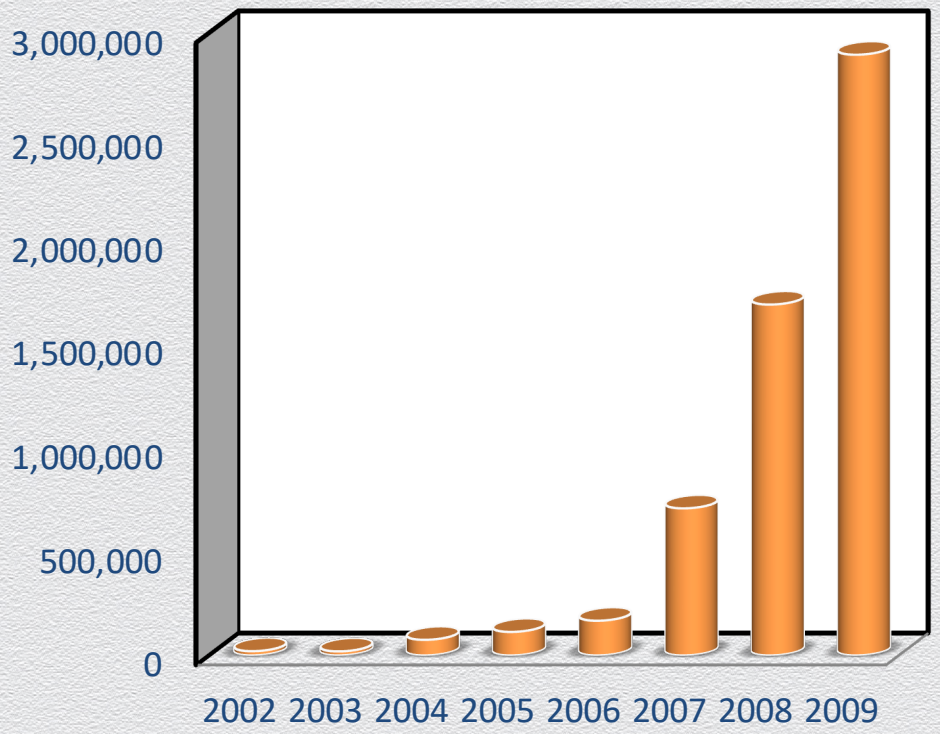
Payload: ranging from annoyance to crime…

Early history

- 1972: sci-fi novel “When HARLIE Was One” features self-reproducing computer program called VIRUS
- 1982: high-school student Rich Skrenta wrote first virus released in the wild, Elk Cloner, a boot sector virus
- 1984: first academic use of “virus” by PhD student Fred Cohen, who credits advisor Len Adleman
- 1986: (c)Brain, by Basit and Amjood Farooq Alvi, credited with being first virus to infect PCs
- 1987: CHRISTMA EXEC targeting IBM VM/CMS systems was first email worm
- 1988: first internet worm, Morris Worm by Cornell student Robert Tappan Morris

Previous decade 2000-2009

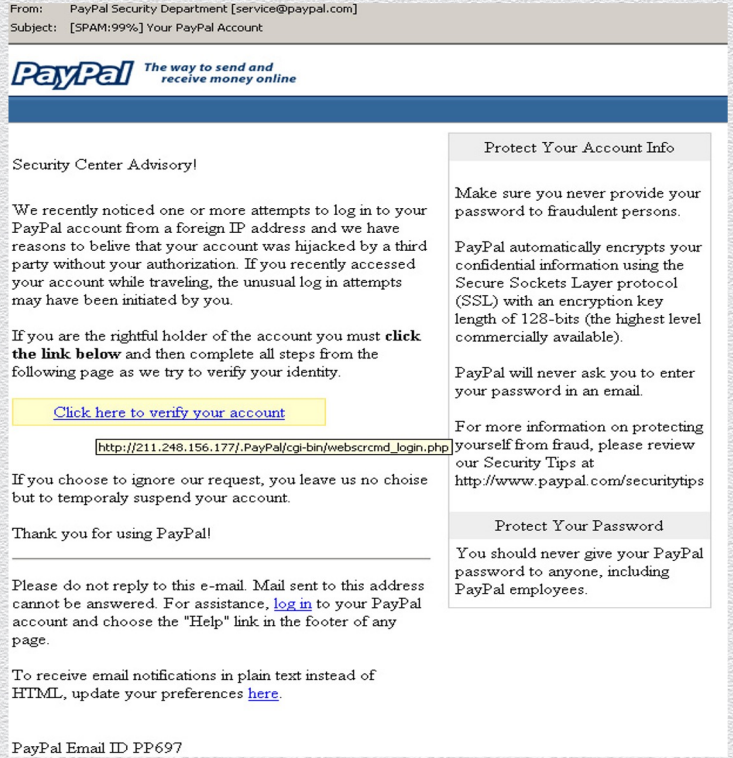
- New malware threats have grown from 20K to 3M in the period 2002-2009
- Most of the growth has been from 2006 to 2009
- Growth in professional cybercrime and online fraud led to demand for professionally developed malware
- New malware often a custom-designed variation of known one
- Most notable:MELISSA, ILOVEYOU, CODE RED, NIMDA, etc.
- Let see the modern malwares…



Source: Symantec Internet Security Threat Report

Malware vectors

- Attempt to fraudulently acquire sensitive information
- Passwords, credit card numbers, etc.
- Usually copies the HTML of a website and tries to pass off as a sub-site of that page.
- Phishers create a page or e-mail (spam) that appears to be from another source
- Usually relies on the user not exploring the page in depth
- Famous phishing attempts are PayPal and eBay scams
- Examples on www.phishtank.com, openphish.com


From: PayPal Security Department [service@paypal.com]
Subject: [SPAM:99%] Your PayPal Account

PayPal The way to send and receive money online

Security Center Advisory!

We recently noticed one or more attempts to log in to your PayPal account from a foreign IP address and we have reasons to belive that your account was hijacked by a third party without your authorization. If you recently accessed your account while traveling, the unusual log in attempts may have been initiated by you.

If you are the rightful holder of the account you must **click the link below** and then complete all steps from the following page as we try to verify your identity.

Click here to verify your account

http://211.248.156.177/.PayPal/cgi-bin/webscrcmd_login.php

If you choose to ignore our request, you leave us no choise but to temporaly suspend your account.

Thank you for using PayPal!

Please do not reply to this e-mail. Mail sent to this address cannot be answered. For assistance, log in to your PayPal account and choose the "Help" link in the footer of any page.

To receive email notifications in plain text instead of HTML, update your preferences here.

PayPal Email ID PP697

Protect Your Account Info

Make sure you never provide your password to fraudulent persons.

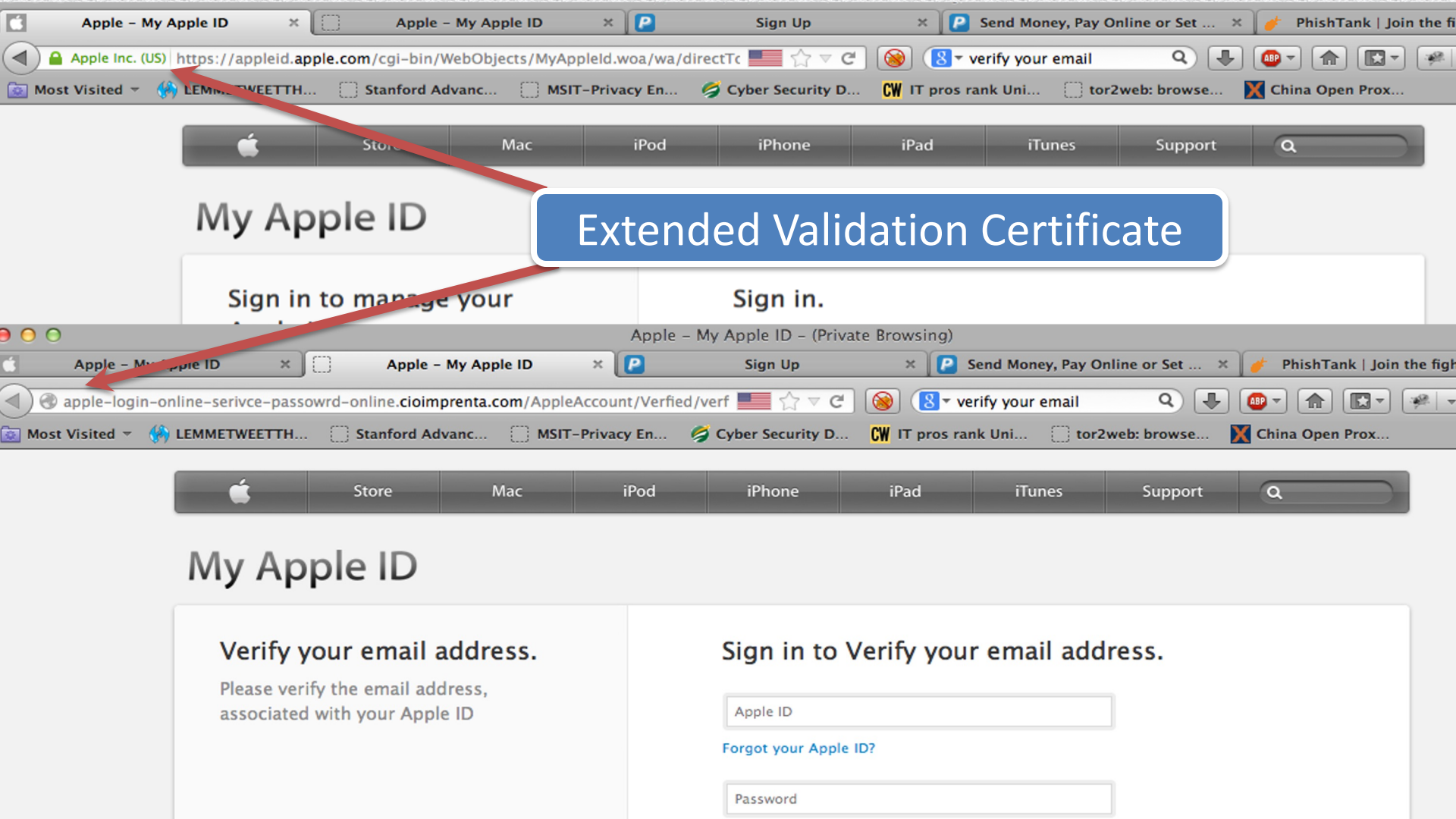
PayPal automatically encrypts your confidential information using the Secure Sockets Layer protocol (SSL) with an encryption key length of 128-bits (the highest level commercially available).

PayPal will never ask you to enter your password in an email.

For more information on protecting yourself from fraud, please review our Security Tips at http://www.paypal.com/securitytips

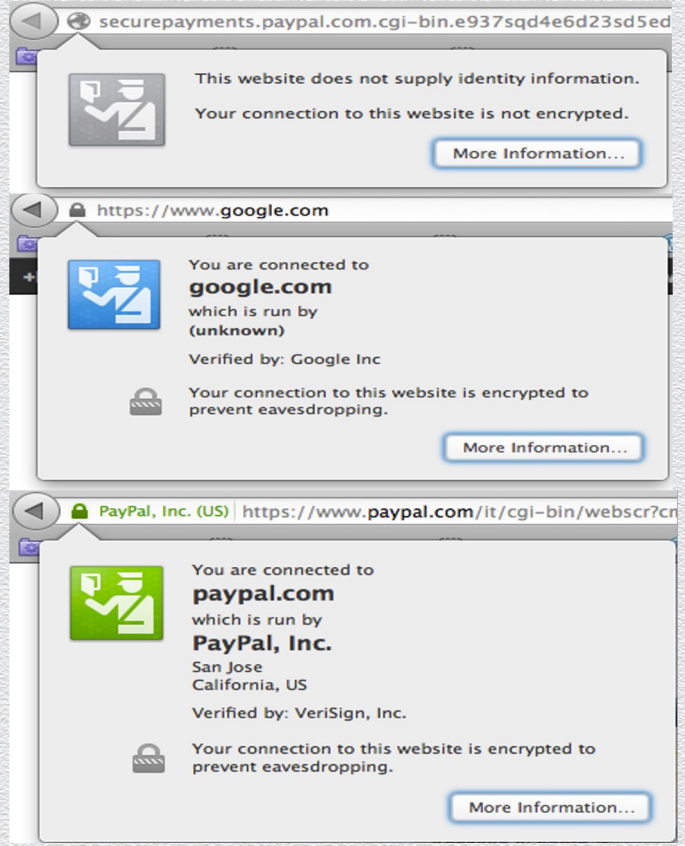
Protect Your Password

You should never give your PayPal password to anyone, including PayPal employees.


Apple – My Apple ID
Apple – My Apple ID
Sign Up
Send Money, Pay Online or Set ...
PhishTank | Join the fi
Apple Inc. (US)
https://appleid.apple.com/cgi-bin/WebObjects/MyAppleId.woa/wa/directTc
verify your email
Most Visited
Stanford Advanc...
MSIT-Privacy En...
Cyber Security D...
IT pros rank Uni...
tor2web: browse...
China Open Prox...
Store
Mac
iPod
iPhone
iPad
iTunes
Support
My Apple ID
Extended Validation Certificate
Sign in to manage your
Sign in.
Apple – My Apple ID – (Private Browsing)
Apple – My Apple ID
Sign Up
Send Money, Pay Online or Set ...
PhishTank | Join the figh
apple-login-online-serivce-passowrd-online.cioimprenta.com/AppleAccount/Verfied/verf
verify your email
Most Visited
LEMMETWEETTH...
Stanford Advanc...
MSIT-Privacy En...
Cyber Security D...
IT pros rank Uni...
tor2web: browse...
China Open Prox...
Store
Mac
iPod
iPhone
iPad
iTunes
Support
My Apple ID
Verify your email address.
Please verify the email address,
associated with your Apple ID
Sign in to Verify your email address.
Apple ID
Forgot your Apple ID?
Password

Extended Validation Certificate: Firefox

- Instant Website ID
- A color-coded system makes it easy to check on suspicious sites and avoid Web forgeries
- Anti-Phishing & Anti-Malware
- Firefox protects you from trojan horses and spyware, and warns you away from fraudulent sites



“why would anyone give their personal data to a phisher?”

Spear Phishing

- Phishing attempts directed at specific individuals or companies
- Attackers may gather personal information about their target to increase their probability of success

Whaling

- Attacks directed specifically at senior executives and other high-profile targets within businesses

These attacks are very difficult to understand and usually use email system

Types of malware

Code Type	Characteristics
Virus	Code that causes malicious behavior and propagates copies of itself to other programs
Trojan horse	Code that contains unexpected, undocumented, additional functionality
Worm	Code that propagates copies of itself through a network; impact is usually degraded performance
Rabbit	Code that replicates itself without limit to exhaust resources
Logic bomb	Code that triggers action when a predetermined condition occurs
Time bomb	Code that triggers action when a predetermined time occurs
Dropper	Transfer agent code only to drop other malicious code, such as virus or Trojan horse
Hostile mobile code agent	Code communicated semi-autonomously by programs transmitted through the web
Script attack, JavaScript, Active code attack	Malicious code communicated in JavaScript, ActiveX, or another scripting language, downloaded as part of displaying a web page

Types of malware (cont.)

Code Type	Characteristics
RAT (remote access Trojan)	Trojan horse that, once planted, gives access from remote location
Spyware	Program that intercepts and covertly communicates data on the user or the user's activity
Bot	Semi-autonomous agent, under control of a (usually remote) controller or "herder"; not necessarily malicious
Zombie	Code or entire computer under control of a (usually remote) program
Browser hijacker	Code that changes browser settings, disallows access to certain sites, or redirects browser to others
Rootkit	Code installed in "root" or most privileged section of operating system; hard to detect
Trapdoor or backdoor	Code feature that allows unauthorized access to a machine or program; bypasses normal access control and authentication
Tool or toolkit	Program containing a set of tests for vulnerabilities; not dangerous itself, but each successful test identifies a vulnerable host that can be attacked
Scareware	Not code; false warning of malicious code attack

History of malware

Year	Name	Characteristics
1982	Elk Cloner	First virus; targets Apple II computers
1985	Brain	First virus to attack IBM PC
1988	Morris worm	Allegedly accidental infection disabled large portion of the ARPANET, precursor to today's Internet
1989	Ghostballs	First multipartite (has more than one executable piece) virus
1990	Chameleon	First polymorphic (changes form to avoid detection) virus
1995	Concept	First virus spread via Microsoft Word document macro
1998	Back Orifice	Tool allows remote execution and monitoring of infected computer
1999	Melissa	Virus spreads through email address book
2000	IloveYou	Worm propagates by email containing malicious script. Retrieves victim's address book to expand infection. Estimated 50 million computers affected.
2000	Timofonica	First virus targeting mobile phones (through SMS text messaging)
2001	Code Red	Virus propagates from 1st to 20th of month, attacks whitehouse.gov web site from 20th to 28th, rests until end of month, and restarts at beginning of next month; resides only in memory, making it undetected by file-searching antivirus products

History of malware (cont.)

Year	Name	Characteristics
2001	Code Red II	Like Code Red, but also installing code to permit remote access to compromised machines
2001	Nimda	Exploits known vulnerabilities; reported to have spread through 2 million machines in a 24-hour period
2003	Slammer worm	Attacks SQL database servers; has unintended denial-of-service impact due to massive amount of traffic it generates
2003	SoBig worm	Propagates by sending itself to all email addresses it finds; can fake From: field; can retrieve stored passwords
2004	MyDoom worm	Mass-mailing worm with remote-access capability
2004	Bagle or Beagle worm	Gathers email addresses to be used for subsequent spam mailings; SoBig, MyDoom, and Bagle seemed to enter a war to determine who could capture the most email addresses
2008	Rustock.C	Spam bot and rootkit virus
2008	Conficker	Virus believed to have infected as many as 10 million machines; has gone through five major code versions
2010	Stuxnet	Worm attacks SCADA automated processing systems; zero-day attack
2011	Duqu	Believed to be variant on Stuxnet
2013	CryptoLocker	Ransomware Trojan that encrypts victim's data storage and demands a ransom for the decryption key

Ransoware

- Ransomware takes control of the victim machine as in a botnet
- Ransomware encrypts a victim's data (local hard-disk, or networked file-system)
- Attacker requests a ransom in exchange for the decryption key
- Usually with hard-to-trace cryptocurrencies
- No guarantee that you actually get the key

- WannaCry worm (May 2017)
  - 200,000 computers infected in 150 countries, including
    - Large network of hospitals in UK
    - Mobile carrier in Spain
    - TSMC in Taiwan
  - Propagated through an exploit in Windows 7 and older
  - Microsoft had released a security update 1 month earlier

Harm from malicious code

- Harm to users and systems
  - Sending email to user contacts
  - Deleting or encrypting files
  - Modifying system information, such as the Windows registry
  - Stealing sensitive information, such as passwords
  - Attaching to critical system files
  - Hide copies of malware in multiple complementary locations
- Harm to the world
  - Some malware has been known to infect millions of systems, growing at a geometric rate
  - Infected systems often become staging areas for new infections

Transmission and propagation

- Setup and installer program
- Attached file
- Document viruses
- Autorun
- Using non-malicious programs:
  - appended viruses
  - viruses that surround a program
  - integrated viruses and replacements

Malware activation

- One-time execution (implanting)
- Boot sector viruses
- Memory-resident viruses
- Application files
- Code libraries

Virus effects

Virus Effect	How It Is Caused
Attach to executable program	• Modify file directory • Write to executable program file
Attach to data or control file	• Modify directory • Rewrite data • Append to data • Append data to self
Remain in memory	• Intercept interrupt by modifying interrupt handler address table • Load self in non-transient memory area
Infect disks	• Intercept interrupt • Intercept operating system call (to format disk, for example) • Modify system file • Modify ordinary executable program
Conceal self	• Intercept system calls that would reveal self and falsify result • Classify self as “hidden” file
Spread infection	• Infect boot sector • Infect systems program • Infect ordinary program • Infect data ordinary program reads to control its execution
Prevent deactivation	• Activate before deactivating program and block deactivation • Store copy to reinfect after deactivation

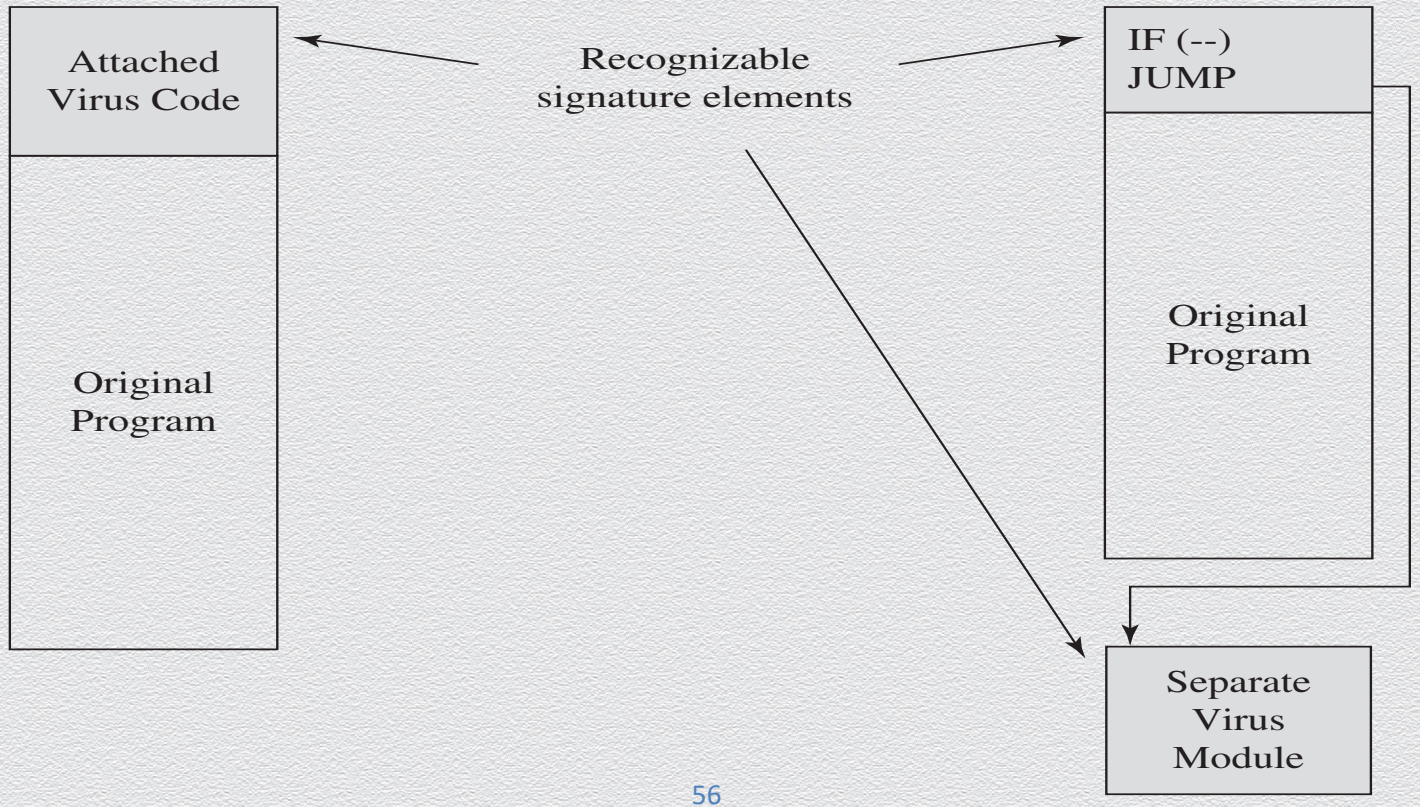
Countermeasures for users

- Use software acquired from reliable sources
- Test software in an isolated environment
- Only open attachments when you know them to be safe
- Treat every website as potentially harmful
- Create and maintain backups

Virus detection

- Virus scanners look for signs of malicious code infection using signatures in program files and memory
- Traditional virus scanners have trouble keeping up with new malware—detect about 45% of infections
- Detection mechanisms
  - Known string patterns in files or memory
  - Execution patterns
  - Storage patterns

Virus signatures



Countermeasures for developers

- Modular code: Each code module should be
  - Single-purpose
  - Small
  - Simple
  - Independent
- Encapsulation
- Information hiding
- Mutual suspicion
- Confinement
- Genetic diversity

Code testing

- Unit testing
- Integration testing
- Function testing
- Performance testing
- Acceptance testing
- Installation testing
- Regression testing
- Penetration testing

Design principles for security

- Least privilege
- Economy of mechanism
- Open design
- Complete mediation
- Permission based
- Separation of privilege
- Least common mechanism
- Ease of use

Other countermeasures

- Good
  - Proofs of program correctness—where possible
  - Defensive programming
  - Design by contract
- Bad
  - Penetrate-and-patch
  - Security by obscurity

18.4 Cloud security

Another example: Tax return preparation…

Involves information collection & processing

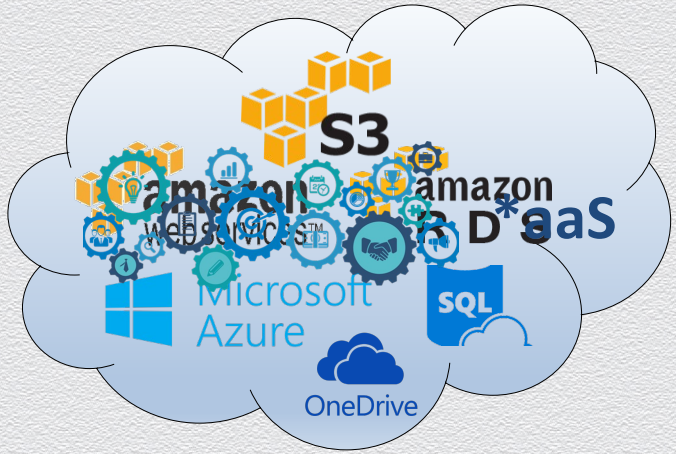
- calculate financial data
  - payroll, profits, stock quotes, …
- manage data
  - search emails, store records, …
- submit – done!

… by many unknown machines!

Data & computation outsourcing

Cloud-based services

- hardware, OS, software, apps, …
- storage, computation, databases, analytics, …



Transformative multi-platform technology

- businesses, organizations or individuals
- client-server, distributed, P2P, Web-based, …

Internet protocols

social networks

big-data analytics

sharing economy

FinTech

Security consequences

Fact: Untrusted interactions

- information is processed outside one's administration control or "trust perimeter"

Risk: Falsified / leaked information

- information may (un)intentionally altered by or shared with unauthorized entities

Goal: Integrity / privacy safeguards for outsourced assets

- need to protect information against change, damage / unauthorized access

What is cloud computing?

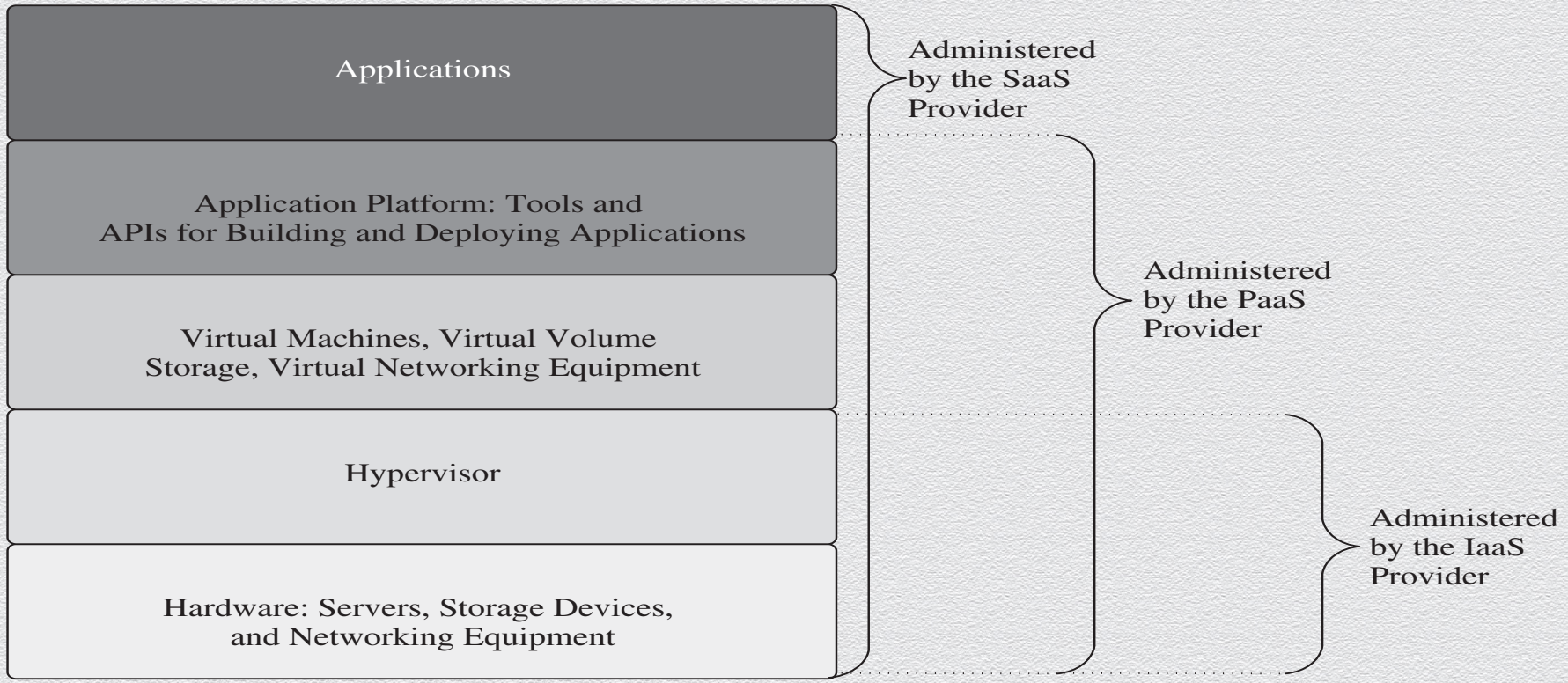
- On-demand self-service
  - Add or subtract resources as necessary
- Broad network access
  - Mobile, desktop, mainframe
- Resource pooling
  - Multiple tenants share resources that can be reassigned dynamically according to need and invisibly to the tenants
- Rapid elasticity
  - Services can quickly and automatically scale up or down to meet customer need
- Measure service
  - Like water, gas, or telephone service, usage can be monitored for billing

Cloud-computing service models

On-demand self-service computing

- Software as a service (SaaS)
  - the cloud provider gives the customer access to applications running in the cloud
- Platform as a service (PaaS)
  - the customer has his or her own applications,
    but the cloud provides the languages and tools for creating and running them
- Infrastructure as a service (IaaS)
  - the cloud provider offers processing, storage, networks, and other computing resources that enable customers to run any kind of software

Service models



Deployment models

- Private cloud
  - Infrastructure that is operated exclusively by and for the organization that owns it
- Community cloud
  - Shared by several organizations with common needs, interests, or goals
- Public cloud
  - Owned by a cloud service provider and offered to the general public
- Hybrid cloud
  - Composed of two or more types of clouds, connected by technology that enables data and applications to balance loads among those clouds

Cloud provider assessment

- Security issues to consider
  - Authentication, authorization, and access control options
  - Encryption options
  - Audit logging capabilities
  - Incident response capabilities
  - Reliability and uptime

Security benefits of cloud services

- Geographic diversity
  - many cloud providers run data centers in disparate geographic locations and mirror data across locations, providing protection from natural and other local disasters
- Platform & infrastructure diversity
  - different platforms and infrastructures mean different bugs and vulnerabilities, which makes a single attack or error less likely to bring a system down
  - using cloud services as part of a larger system can be a good way to diversify your technology stack
  - e.g., Honeywords, virtualization, etc.

Cloud-based security functions

Some security functions may be best handled by cloud service providers

- Email filtering
  - since email is already hopping through a variety of SMTP servers, adding a cloud-based email filter is as simple as adding another hop
- DDoS protection
  - cloud-based DDoS protection services update your DNS records to insert their servers as proxies in front of yours
  - they maintain sufficient bandwidth to handle the flood of attack traffic
- Network monitoring
  - cloud-based solutions can help customers deal with steep hardware requirements and can provide monitoring and incident response expertise

Switching cloud providers

- Switching cloud providers is expensive and difficult but sometimes becomes necessary and urgent
- It is best to have backup options in place in case a migration away from a cloud provider is necessary, but many cloud providers make that practically impossible
- SaaS providers are generally hardest to migrate away from, followed by PaaS, then IaaS