

# CISC 3325 - INFORMATION SECURITY

---

## Software Security



Adapted from *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved



# Malware: Malicious Software

<https://www.welivesecurity.com/2017/10/19/malware-firmware-exploit-sense-security/>

# Malware

- Programs planted by an agent with malicious intent to cause unanticipated or undesired effects
  - Can be used to:
    - Obtain your sensitive information
    - Delete or modify files
    - Any unwanted/malicious purpose

# Malware

- 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 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

# Malware Payload

- Various types of payloads, ranging from annoyance to crime
- Payload examples
  - perform amusing or annoying pranks
  - destroy/corrupt files and applications
  - monitor and transmit user activity (spyware, logger)
  - install backdoor (makes the infected computer a zombie)
  - email spam
  - launch denial-of-service attack
  - alter browser settings to display ads
  - dial out international or 900 numbers (dialer)

# 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 from Malicious Code

- 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

# MALWARE TYPES

---



# Computer Viruses

- The most known type of malware
- A **computer virus** attaches itself to some program
  - Similar to biological viruses that attach themselves to human body
- Virus is computer code that can replicate itself by modifying other files/programs
  - to insert code that is capable of further replication.
  - such as logic bombs.

<https://www.yelp.com/biz/40-computer-virus-removal-reseda>

# Computer Viruses

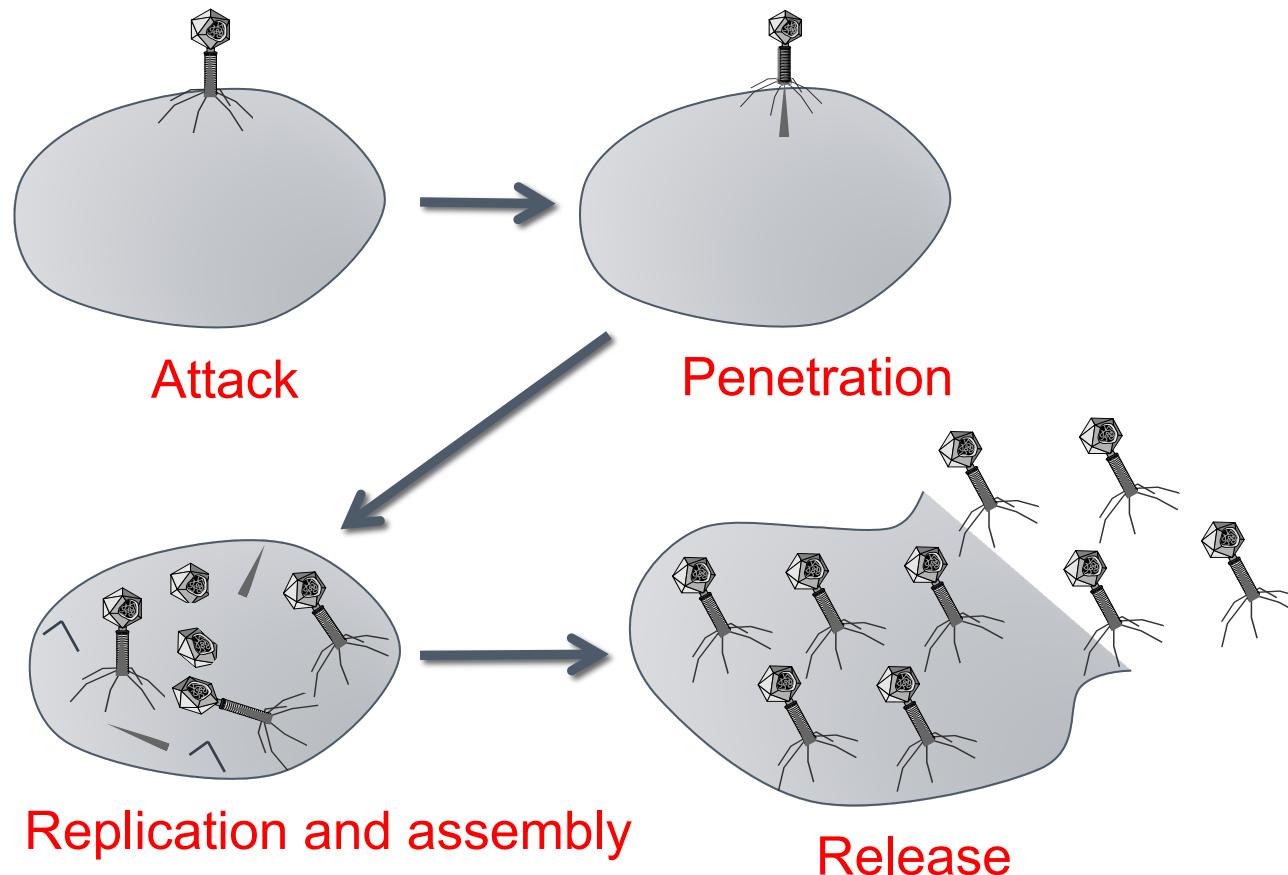


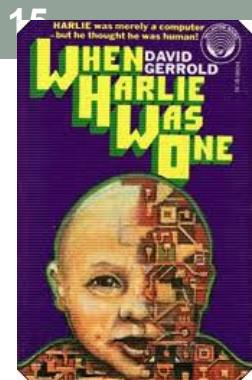
- This self-replication property distinguishes computer viruses from other kinds of malware
- Another distinguishing property is that virus replication requires some **user assistance**
  - such as clicking on an email attachment or sharing a USB drive.

<https://www.yelp.com/biz/40-computer-virus-removal-reseda>

# Biological Analogy

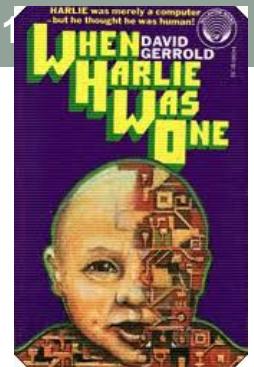
- Computer viruses share some properties with Biological viruses





# Early History

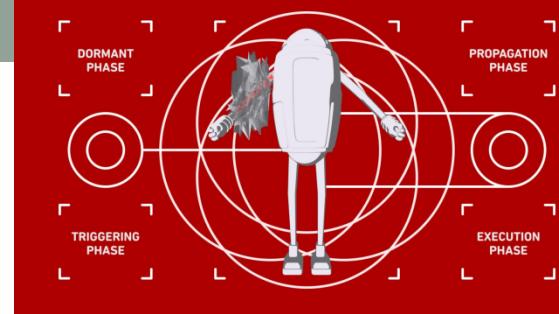
- 1972 sci-fi novel “When HARLIE Was One” features a program called VIRUS that reproduces itself
- First academic use of term ‘virus’ by PhD student Fred Cohen in 1984
  - credits advisor Len Adleman with coining it



# Early History

- In 1982, high-school student Rich Skrenta wrote first virus released in the wild: Elk Cloner
  - A boot sector virus
- Brain, by Basit and Amjood Farooq Alvi in 1986, credited with being the first virus to infect PCs

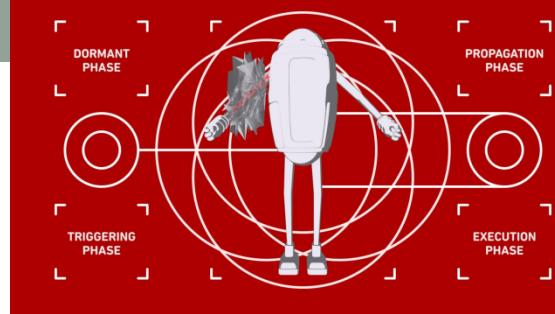
# Virus Phases



- **Dormant phase:**
  - The virus just exists—the virus is laying low and avoiding detection.
- **Propagation phase:**
  - The virus is replicating itself, infecting new files on new systems.
- **Triggering phase:**
  - Some logical condition causes the virus to move from a dormant or propagation phase to perform its intended action.

<https://festival.vconline.org/2017/films/nuwa/>

# Virus Phases



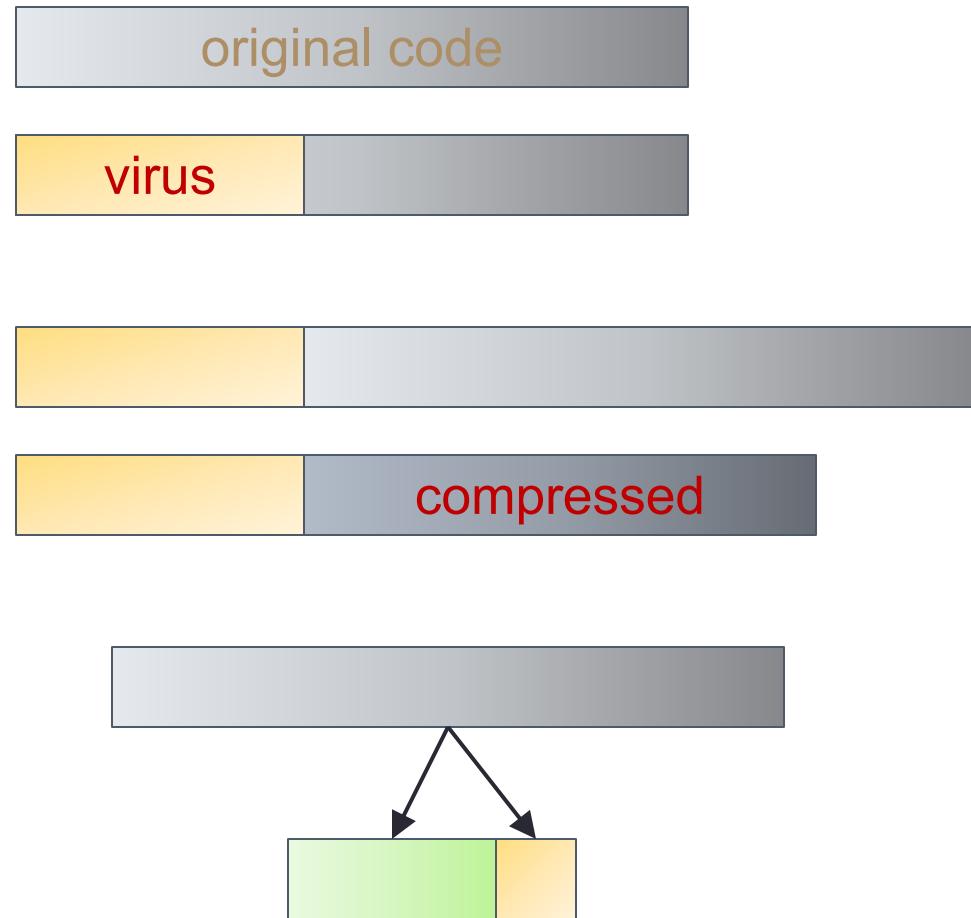
- **Action phase:**

- The virus performs the malicious action that it was designed to perform, called **payload**.
  - This action could include something seemingly innocent, like displaying a silly picture on a computer's screen
  - or something quite malicious, such as deleting all essential files on the hard drive.

<https://festival.vconline.org/2017/films/nuwa/>

# Infection Types

- Overwriting
  - Destroys original code
- Pre-pending
  - Keeps original code, possibly compressed
- Infection of libraries
  - Allows virus to be memory resident
  - E.g., kernel32.dll
- Macro viruses
  - Infects MS Office documents
  - Often installs in main document template



# Resident and Non-resident Viruses

- Resident viruses continue running after executing the infected file
  - Modified system calls
  - Modified DLLs
- Resident viruses are more common than non-resident viruses
  - essentially latch onto system calls, DLLs and the like, and stay resident
  - affecting every program run subsequent to them being introduced into memory.

# Resident and Non-resident Viruses

- Non resident viruses are executed every time an infected file is executed

# Viruses and DLL's

- All Windows DLLs have an export table listing the functions provided and their addresses
- A virus can hook onto a DLL
- Fairly easy for viruses using DLLs to get memory resident

# Viruses and DLL's

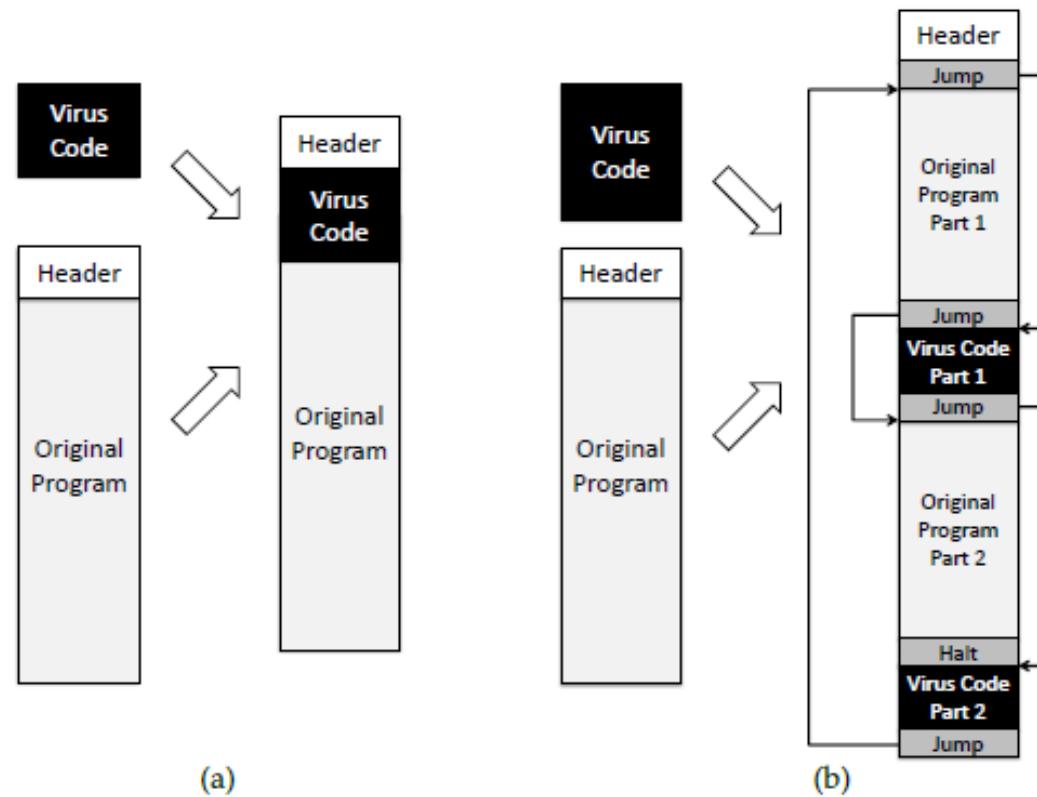
- Example: kernel32.dll is a collection of core Windows API calls (system calls)
  - imported by most applications
- Most viruses relying on patching DLLs usually attack kernel32.dll

# Viruses and Windows DLL's

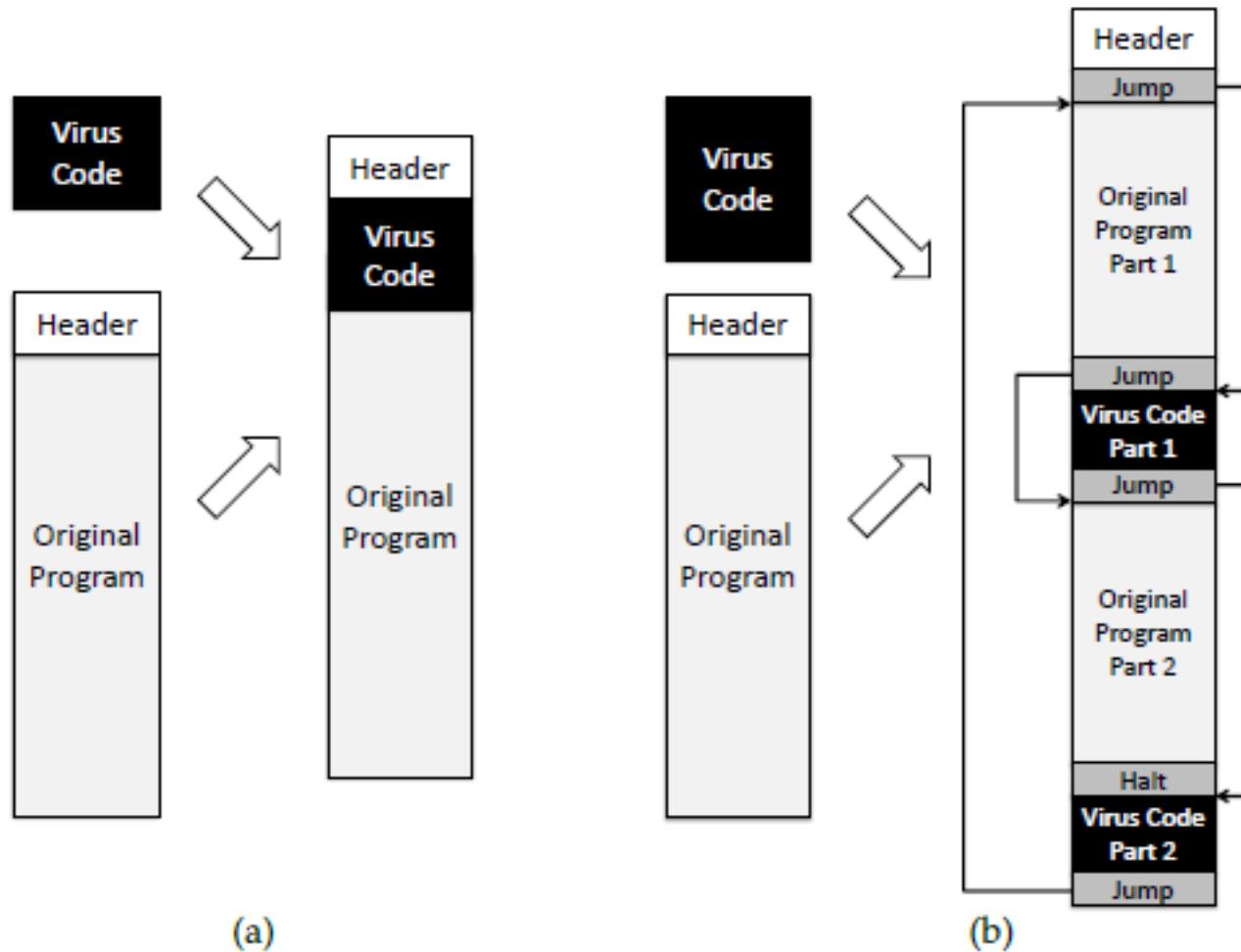
- Example: W32.Kriz will attack any PE executable, and kernel32.dll
  - to get a hook on system calls
- Hooking system calls may be done by legitimate programs, such as Regmon (a registry monitoring utility)
- Viruses hook onto DLLs by either:
  - changing their exported symbol table, so as to call malicious code
  - adding malicious code to the DLL.

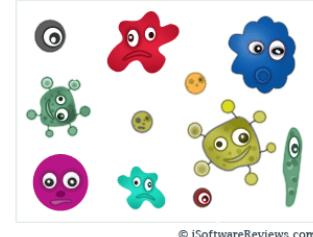
# Degrees of Complication

- Viruses have various degrees of complication in how they can insert themselves in computer code.



# Degrees of Complication





© iSoftwareReviews.com

# Concealment

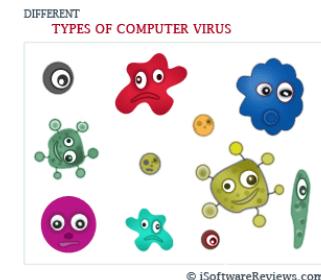
- Encrypted virus
  - Decryption engine + encrypted body
  - Randomly generate encryption key
  - Detection looks for decryption engine
- Polymorphic virus
  - Encrypted virus with random variations of the decryption engine (e.g., padding code)
  - Detection using CPU emulator



<http://www.hoax-slayer.com/really-bad-virus-warning.shtml>, <https://sites.google.com/site/a14g32/home/different-types-of-computer-viruses>

# Concealment

- Metamorphic virus
  - Different virus bodies
  - Approaches include code permutation and instruction replacement
  - Challenging to detect

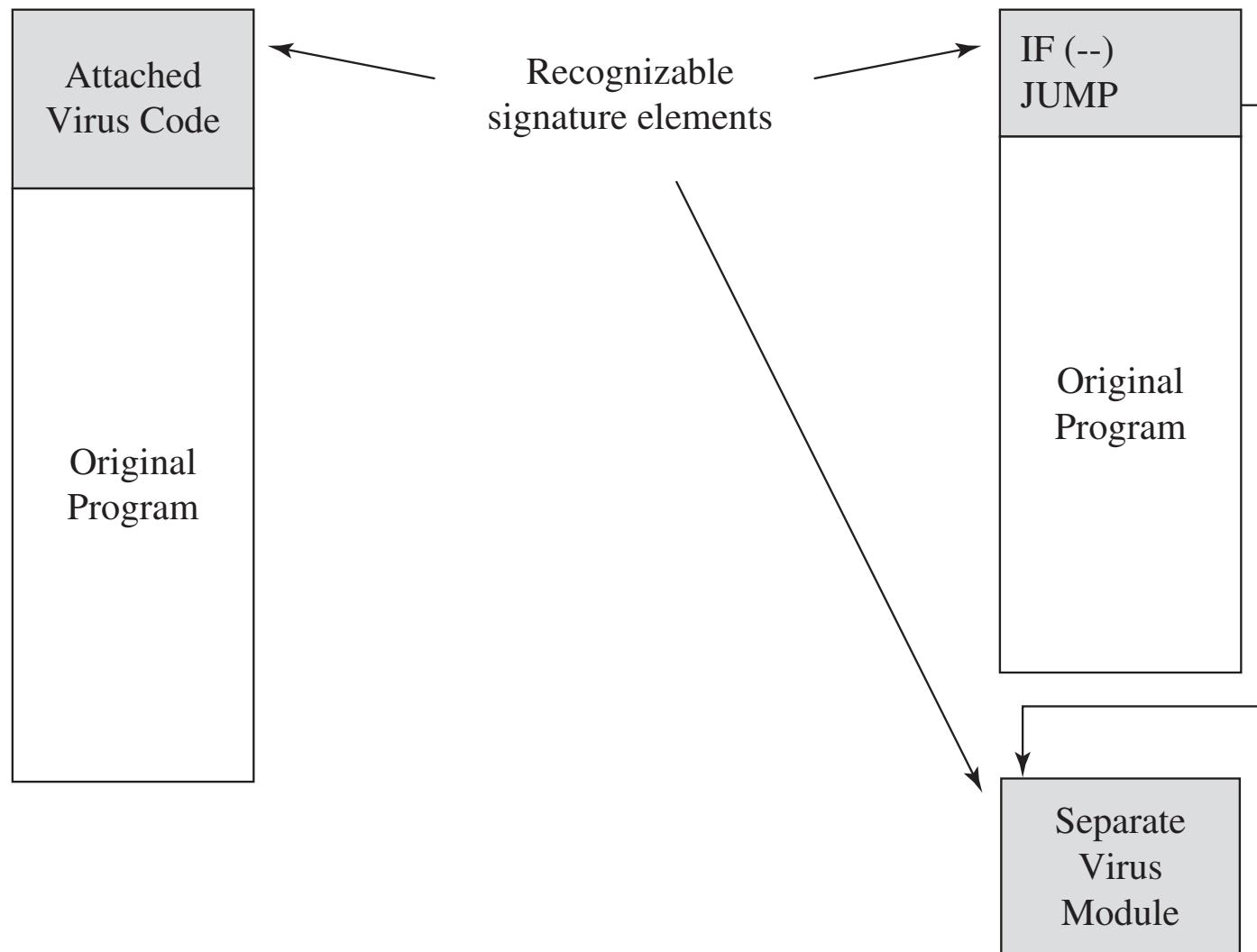


<http://www.hoax-slayer.com/really-bad-virus-warning.shtml>, <https://sites.google.com/site/a14g32/home/different-types-of-computer-viruses>

# 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





# Computer Worms

- A **computer worm** is a malware program that spreads copies of itself
  - without the need to inject itself in other programs, and usually without human interaction.
- Thus, computer worms are technically not computer viruses
  - since they don't infect other programs
    - but some people nevertheless confuse the terms, since both spread by self-replication.

<https://www.youtube.com/watch?v=Jz83nFMH-04>



# Computer Worms

- In most cases, a computer worm will carry a malicious payload
  - such as deleting files or installing a backdoor.

<https://www.youtube.com/watch?v=Jz83nFMH-04>

# Early History

- First worms built in the labs of John Shock and Jon Hepps at Xerox PARC in the early 80s
- CHRISTMA EXEC written in REXX, released in December 1987, and targeting IBM VM/CMS systems was the first worm to use e-mail service

# Early History (cont.)

- The first internet worm was the Morris Worm, written by Cornell student Robert Tappan Morris and released on November 2, 1988
  - First person to be indicted under the Computer Fraud and Abuse Act
    - Sentenced to probation

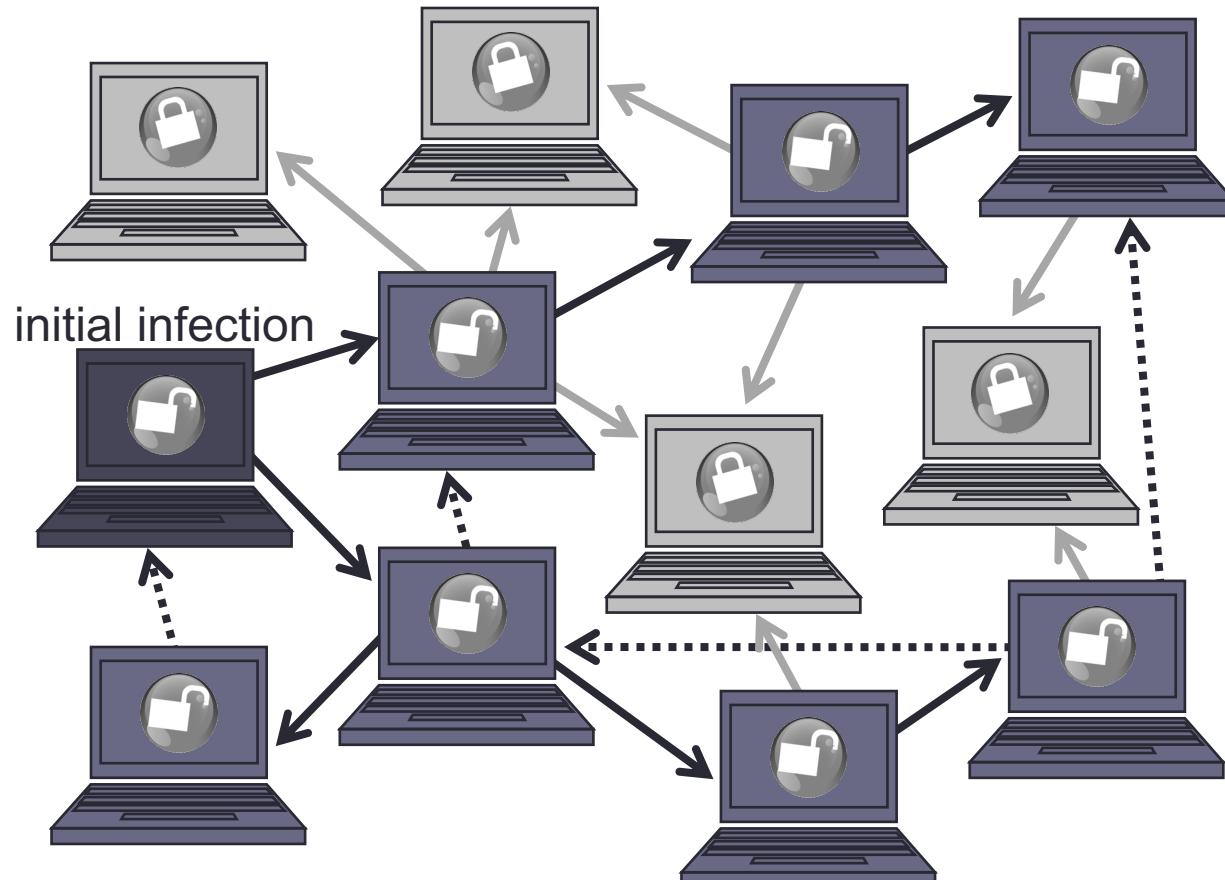


# I love you Worm

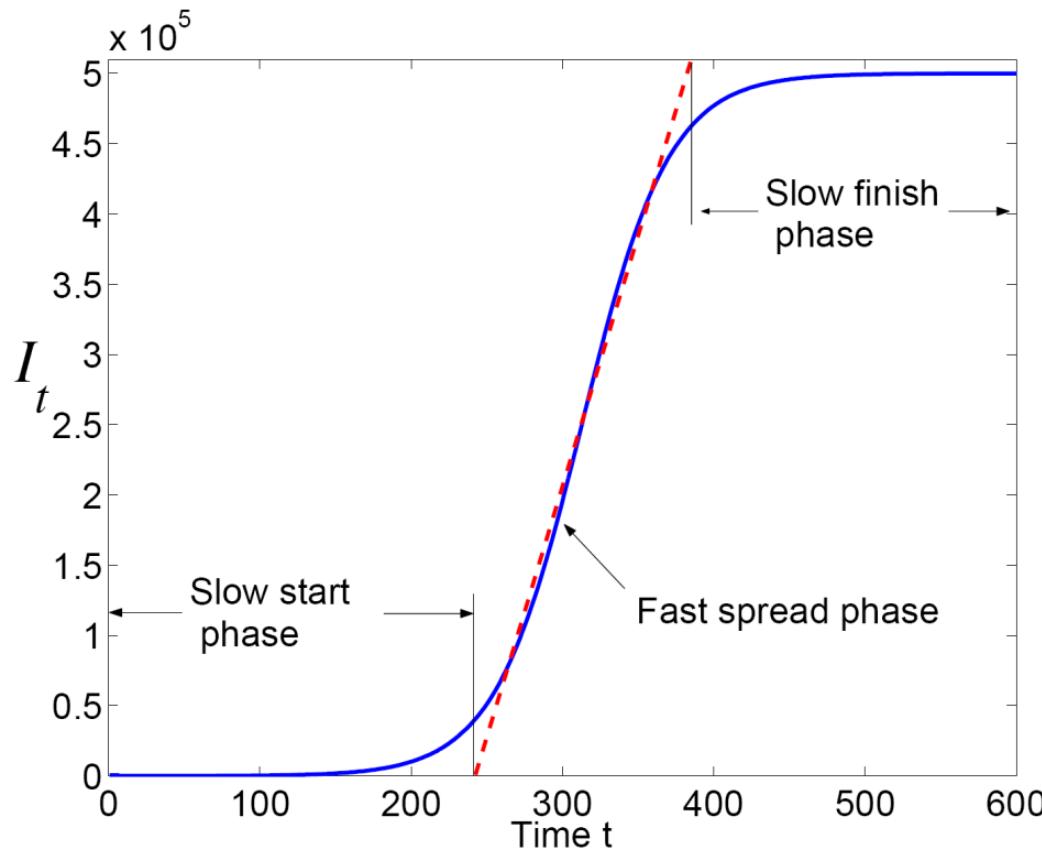
- A famous worm
- Spread to millions of Windows machines
  - By Email
  - When opened, would execute many attacks
  - Spread by stealing email addresses from the computer
    - Resending the emails to multiple new targets
- I love you worm

# Worm Propagation

- Worms propagate by finding and infecting vulnerable hosts.
  - They need a way to tell if a host is vulnerable
  - They need a way to tell if a host is already infected.



# Worm Propagation: Theory

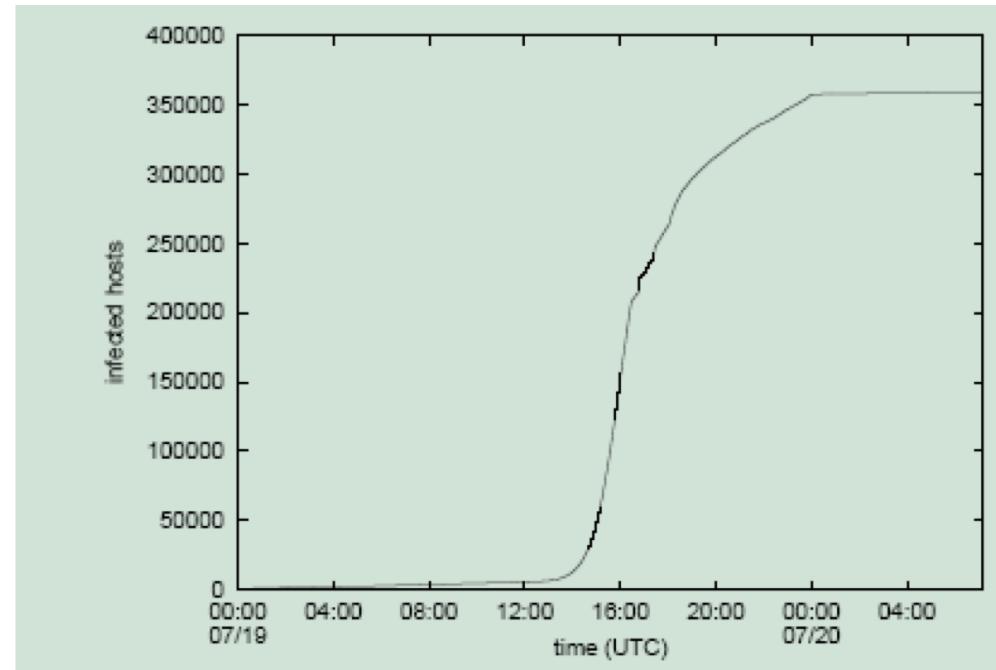


Source:  
Cliff C. Zou, Weibo Gong, Don Towsley,  
and Lixin Gao. [The Monitoring and Early  
Detection of Internet Worms](#), IEEE/ACM  
Transactions on Networking, 2005.

# Propagation: Practice

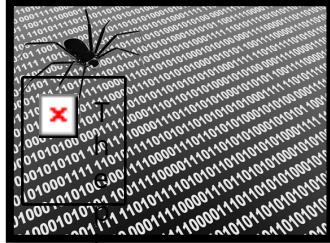
- Cumulative total of unique IP addresses infected by the first outbreak of Code-Red v2 on July 19-20, 2001

Source:  
David Moore, Colleen  
Shannon, and Jeffery  
Brown. [Code-Red: a  
case study on the spread  
and victims of an Internet  
worm](#), CAIDA, 2002



# Adware

Adware software payload

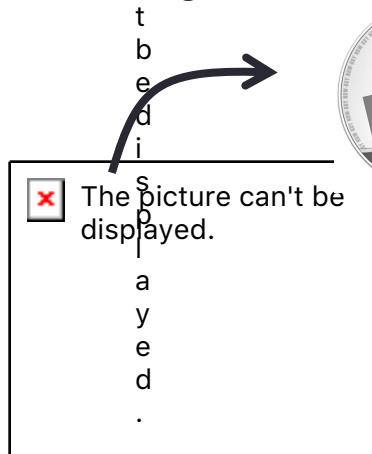


Adware engine infects  
a user's computer

Computer user



Advertisers contract with  
adware agent for content



Adware engine requests  
advertisements  
from adware agent



Adware agent

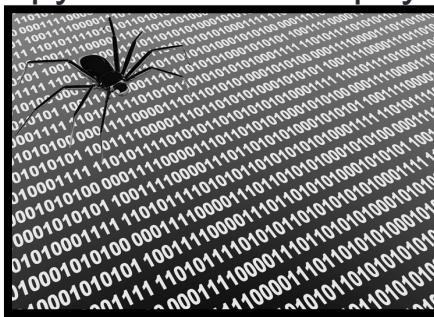
Adware agent delivers  
ad content to user



Advertisers

# Spyware

Spyware software payload



1. Spyware engine infects a user's computer.

Computer user



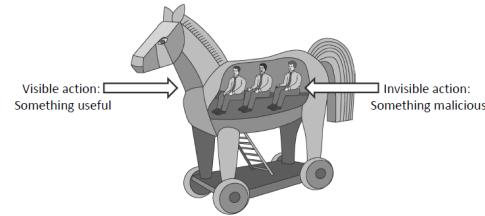
2. Spyware process collects keystrokes, passwords, and screen captures.



3. Spyware process periodically sends collected data to spyware data collection agent.

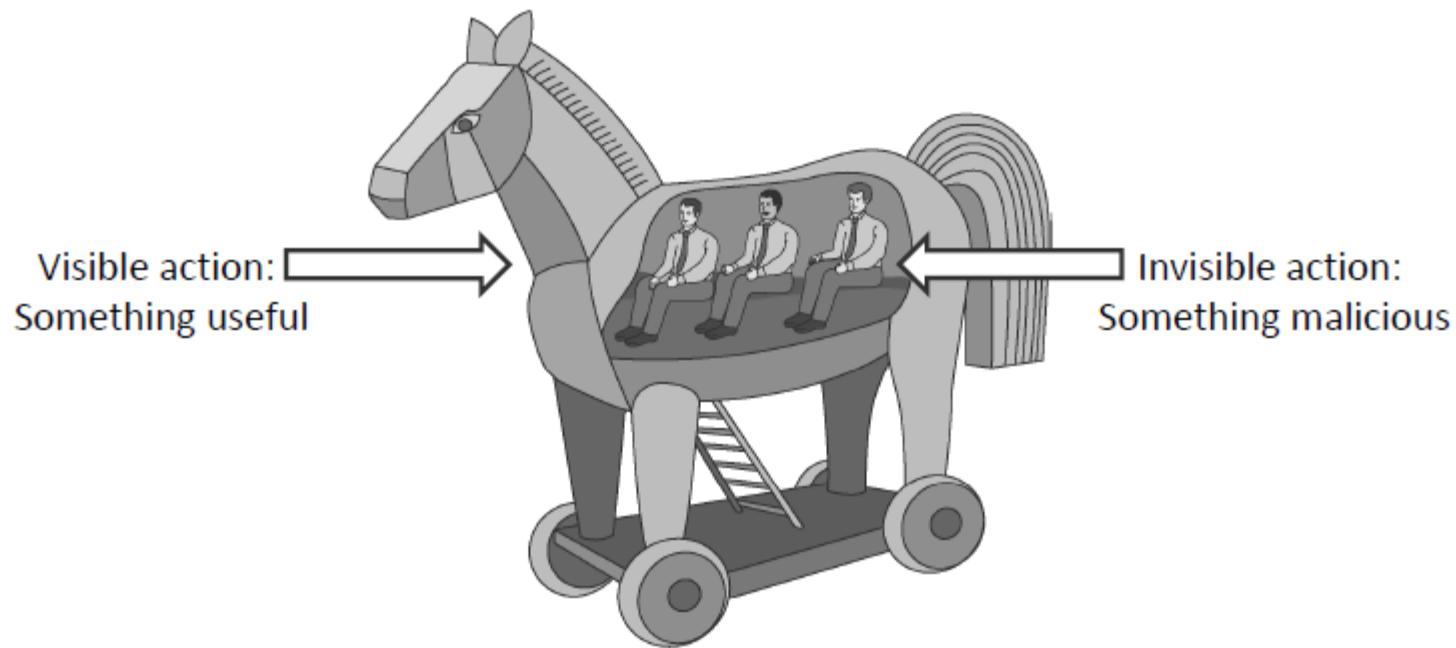
Spyware data collection agent

# Trojan Horses



- A **Trojan horse (or Trojan)** is a malware program that appears to perform some useful task
  - but also does something with negative consequences (e.g., launches a keylogger).
- Trojan horses can be installed as part of the payload of other malware
- Often installed by a user or administrator, either deliberately or accidentally.

# Trojan Horses



# Trojan Horses

- Name derives from the wooden horse left by the Greeks at the gates of Troy
- A Trojan horse program intentionally hides malicious activity while pretending to be something else
- Usually described as innocuous looking, or software delivered through innocuous means which either allows to take control of systems
- Trojan horse programs do not replicate themselves

# Trojan Horses

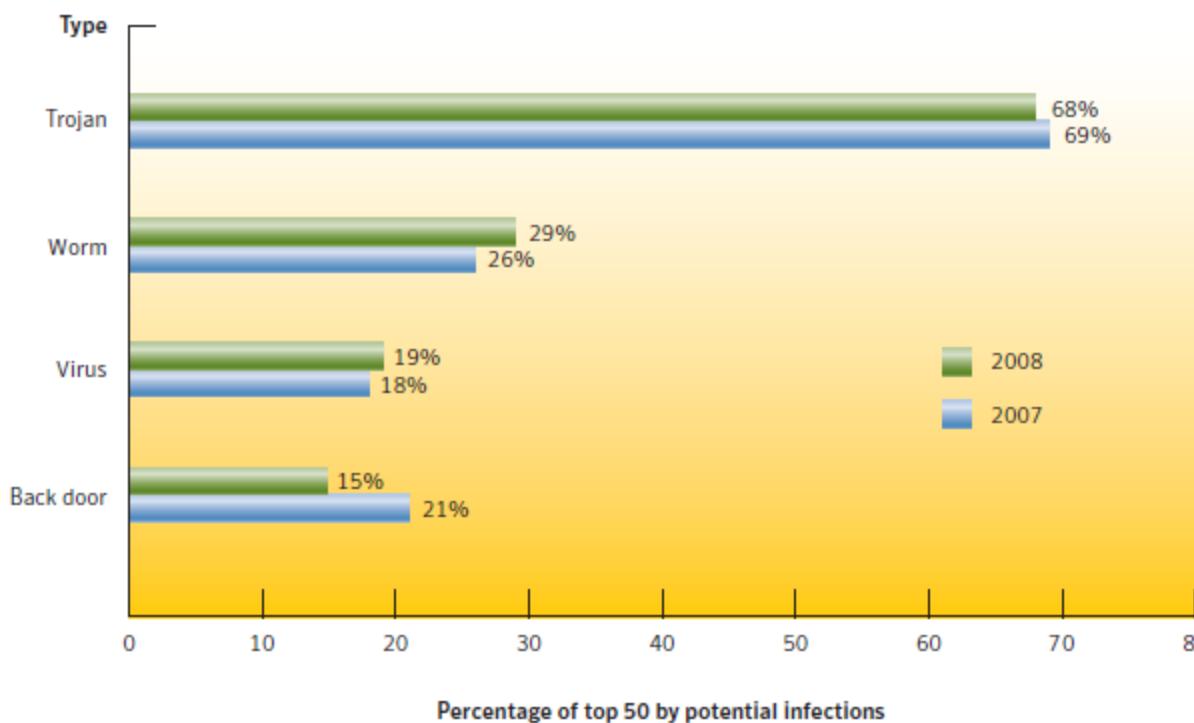
- Sometimes passed on using commonly passed executables, things like jokes forwarded by e-mail
- Sometimes marketed/distributed as “remote administration tool”
- Often combined with rootkits to disguise activity and remote access
- Popularized to an extent by software like Cult of the Dead Cow’s Back Orifice
  - offered as a free download for running “remote administration” tasks or playing spooky jokes on friends

# Trojan Horses

- The line between user-launched worms and Trojans is highly blurred
  - many user-launched worms behaving in a manner similar to worms.
- Trojans are by definition malicious
  - The classic movie/television exploit of remotely opening disk drives is a definite symptom of being infected by a Trojan.
  - Have lately begun using much of the same defense mechanisms used by viruses

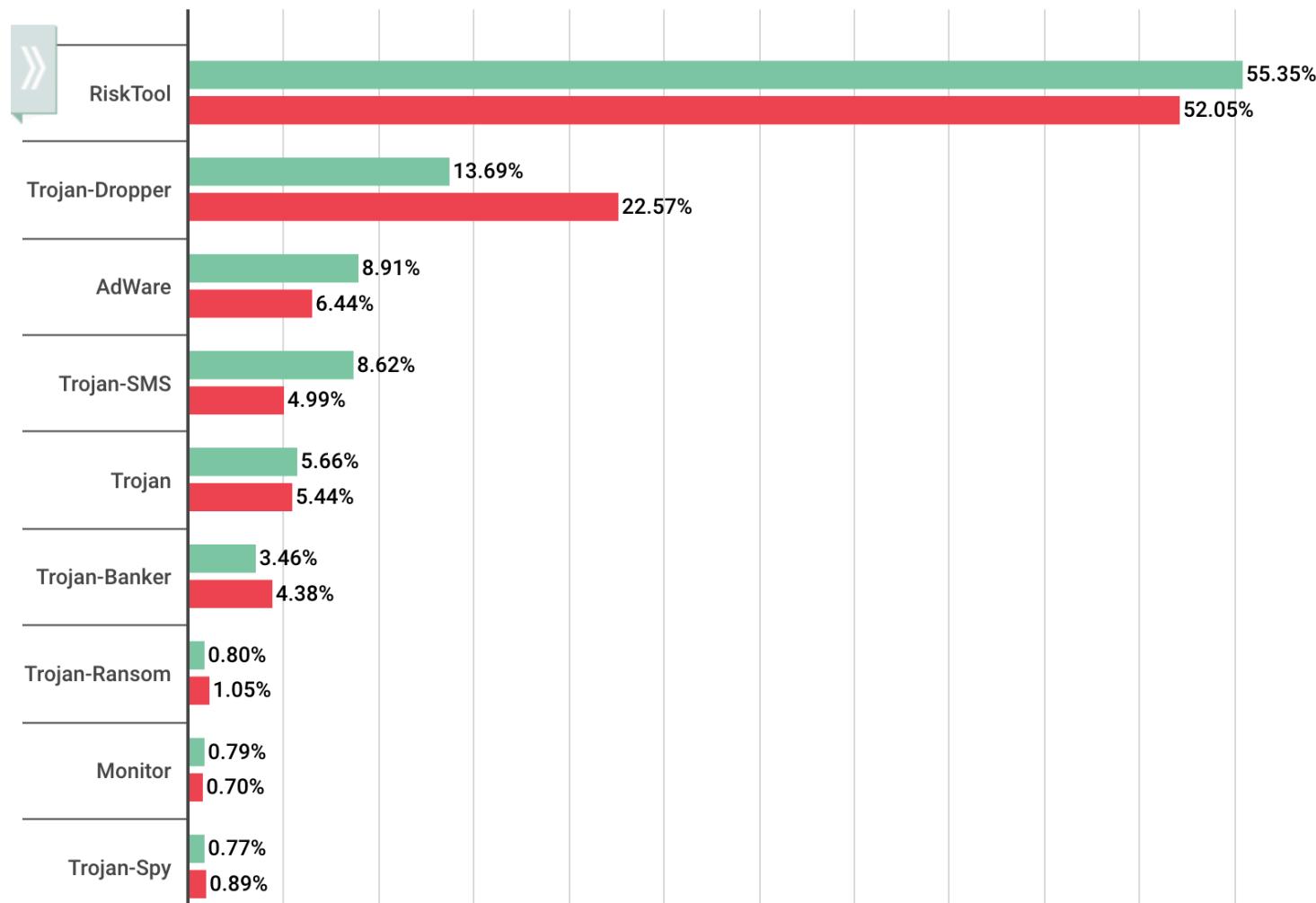
# Current Trends

- Trojans currently have largest infection potential
  - Often exploit browser vulnerabilities
  - Typically used to download other malware in multi-stage attacks



Source:  
Symantec Internet  
Security Threat  
Report, April 2009

# Detected mobile apps by type (2017)





# Rootkits

- A collection of software that an administrator would use
  - Hides itself from the system
  - Processes will not show in task manager
    - Can hide its own presence
  - Hard to detect using software that relies on the OS itself

<https://www.softshop.eu/en/information/library/computer-threats/rootkit/>

# 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 from Malicious Code

- 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 nonmalicious 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

# Types of Malware

<b>Code Type</b>	<b>Characteristics</b>
<b>Virus</b>	Code that causes malicious behavior and propagates copies of itself to other programs
<b>Trojan horse</b>	Code that contains unexpected, undocumented, additional functionality
<b>Worm</b>	Code that propagates copies of itself through a network; impact is usually degraded performance
<b>Rabbit</b>	Code that replicates itself without limit to exhaust resources
<b>Logic bomb</b>	Code that triggers action when a predetermined condition occurs
<b>Time bomb</b>	Code that triggers action when a predetermined time occurs
<b>Dropper</b>	Transfer agent code only to drop other malicious code, such as virus or Trojan horse
<b>Hostile mobile code agent</b>	Code communicated semi-autonomously by programs transmitted through the web
<b>Script attack, JavaScript, Active code attack</b>	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
<b>RAT (remote access Trojan)</b>	Trojan horse that, once planted, gives access from remote location
<b>Spyware</b>	Program that intercepts and covertly communicates data on the user or the user's activity
<b>Bot</b>	Semi-autonomous agent, under control of a (usually remote) controller or "herder"; not necessarily malicious
<b>Zombie</b>	Code or entire computer under control of a (usually remote) program
<b>Browser hijacker</b>	Code that changes browser settings, disallows access to certain sites, or redirects browser to others
<b>Rootkit</b>	Code installed in "root" or most privileged section of operating system; hard to detect
<b>Trapdoor or backdoor</b>	Code feature that allows unauthorized access to a machine or program; bypasses normal access control and authentication
<b>Tool or toolkit</b>	Program containing a set of tests for vulnerabilities; not dangerous itself, but each successful test identifies a vulnerable host that can be attacked
<b>Scareware</b>	Not code; false warning of malicious code attack

# Malware

- We see that many types of malware exist
- Consequences of attacks can be significant
- How do we protect against them?

# 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

# Questions?



# Rise of the Hackers

- Rise of the Hackers

# Questions?



# Countermeasures for Developers

- Modular code: Each code module should be
  - Single-purpose
  - Small
  - Simple
  - Independent
- Encapsulation
  - Restrict direct access to some of the object's components
    - Using built-in language mechanism

# Countermeasures for Developers

- Mutual Suspicion
  - Mutually suspicious programs operate as if other routines in the system were malicious or incorrect.
  - A calling program cannot trust its called subprocedures to be correct
    - a called subprocedure cannot trust its calling program to be correct.
  - Each protects its interface data so the other has only limited access

# Countermeasures for Developers

- Information hiding
  - Segregating the program design decisions most likely to change
    - => protecting other parts of the program from extensive modification if the design decision is changed
- Confinement
  - Isolate program data, functionality
    - For example, server should send only certain data to client

# Code Testing

- Unit testing:
  - individual units/ components of a software are tested.
  - Validate that each **unit** of the software performs as designed
    - A **unit** is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.
- Integration testing:
  - individual software modules are combined and **tested** as a group. It occurs after unit **testing** and before **validation testing**

# Code Testing

- Functional testing:
  - Ensure the software has all the required functionality that's specified within its **functional** requirements
- Performance testing:
  - A testing practice to determine how a system performs
    - in terms of responsiveness and stability under a particular workload
- Acceptance testing:
  - Evaluate the system's compliance with the business requirements
    - assess whether it is acceptable for delivery

# Code Testing

- Installation testing:
  - Most software systems have installation procedures
    - Needed before they can be used for their main purpose.
  - Test these procedures to achieve a usable installed software system
- Regression testing:
  - Testing changes to computer programs
    - to make sure that the older programming still works with the new changes

# Working Towards Secure Systems

- Additional approaches:
  - Use a vulnerability scanner
  - Penetration Testing
  - Design by contract approach

# Vulnerability scanner



- A computer program designed to assess computers, computer systems, networks or applications for known weaknesses
  - probe your systems/networks for known flaws
- Typically used to detect vulnerabilities in software that runs on a network component
  - firewall, router, web server, application server, etc.

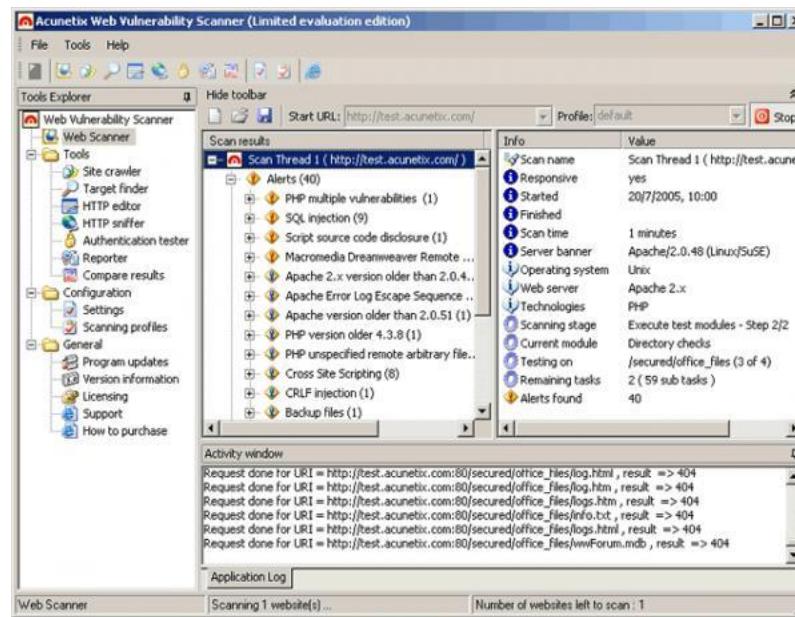
<https://www.businesscomputingworld.co.uk/6-reasons-every-business-needs-a-network-vulnerability-scanner/>

# Vulnerability scanner



- Two types of scans:
  - Authenticated scans: scanner may access low-level data, provide detailed info
    - about OS, installed software, missing security patches, etc.
  - Unauthenticated scans: unable to provide detailed information, may result in high false positives
    - used to determine the security posture of externally accessible assets
      - By attackers or security analysts

# Vulnerability scanner



<http://acunetix-web-vulnerability-scanner.networkice.com/>

# Penetration testing (“pen-testing”)



- Authorized simulated attack on a computer system (“ethical hacking”)
- Hire someone to break into your systems ...
  - Act as if they were in fact an adversary trying to compromise the integrity of their target organization
  - Attacker should remain stealthy and undetected
    - while executing targeted attack patterns
      - observed in real-world corporate breaches
  - Provide detailed documentation about the attack and the vulnerabilities exploited!

# Penetration testing (“pen-testing”)

- Skilled Penetration testing engineers may be hard to distinguish from real-world hackers
  - Other than first obtaining written permission before engaging their targets.
- Goal is to detect both vulnerabilities and strengths
  - enabling a full risk assessment



# Design By Contract Approach

- How can we verify that our code executes in a safe (and correct, ideally) fashion?
  - Build up confidence on a function-by-function module-by-module basis
    - Modularity: the extent to which a software/Web application may be divided into smaller module
  - By using the Design By Contract approach

# Design By Contract Approach

- Modularity provides boundaries for our verification:
  - Preconditions: what must hold for function to operate correctly
  - Postconditions: what holds after function completes
  - Invariants: a condition that is true BEFORE and AFTER running the code

# Design By Contract Approach

- Notions also apply to individual statements (what must hold for correctness; what holds after execution)
  - Statement #1's postcondition should logically imply statement #2's precondition

# Software Examples

- /\* requires: p != NULL  
(and p a valid pointer) \*/
- int deref(int \*p) {  
 return \*p;  
}
- What is the precondition here?
  - What needs to hold for function to operate correctly?

# Software Examples

```
/* requires: p != NULL  
           (and p a valid pointer) */
```

```
int deref(int *p) {  
    return *p;  
}
```

- What is the precondition here?
  - What needs to hold for function to operate correctly?
  - P needs to be a valid pointer
    - Otherwise, return call fails

# Software Examples (cont.)

```
/* ensures: retval != NULL (and a valid pointer) */  
void *mymalloc(size_t n) {  
    void *p = malloc(n);  
    if (!p) { perror("malloc"); exit(1); }  
    return p; }
```

- Postcondition?
  - what does the function promise will hold upon its return

# Software Examples (cont.)

```
/* ensures: retval != NULL (and a valid pointer) */  
void *mymalloc(size_t n) {  
    void *p = malloc(n);  
    if (!p) { perror("malloc"); exit(1); }  
    return p; }
```

- Postcondition?
  - what does the function promise will hold upon its return
  - Function returns a valid pointer
    - With n bits allocated

# Software Examples (cont.)

```
int sum(int a[], size_t n) {  
    int total = 0;  
    for (size_t i=0; i<n; i++)  
        total += a[i];  
    return total;  
}
```

- Precondition?

# Software Examples (cont.)

```
int sum(int a[], size_t n) {  
    int total = 0;  
    for (size_t i=0; i<n; i++)  
        total += a[i];  
    return total;  
}
```

- Precondition?
  - a is a valid pointer to a buffer of size n or larger

# Increase Memory Safety

- General correctness proof strategy for memory safety:
  - Identify each point of memory access
  - Write down precondition it requires
  - Propagate requirement up to beginning of function
    - Document accordingly
  - Write down post-conditions
    - Verify that function fulfills them

# Software Examples (cont.)

```
int sum(int a[], size_t n) {  
    int total = 0;  
    for (size_t i=0; i<n; i++)  
        total += a[i];  
    return total;  
}
```

- Precondition?
  - Identify each point of memory access

# Software Examples (cont.)

```
int sum(int a[], size_t n) {  
    int total = 0;  
    for (size_t i=0; i<n; i++)  
        total += a[i];  
    return total;  
}
```

- Precondition?
  - Identify each point of memory access
  - Write down precondition it requires?

# Software Examples (cont.)

```
int sum(int a[], size_t n) {  
    int total = 0;  
    for (size_t i=0; i<n; i++)  
        /* ?? */  
        total += a[i];  
    return total;  
}
```

- Precondition?
  - Identify each point of memory access
  - Write down precondition it requires?

# Software Examples (cont.)

```
/* ?? */  
int sum(int a[], size_t n) {  
    int total = 0;  
    for (size_t i=0; i<n; i++)  
        /* requires: a != NULL && 0 <= i && i < size(a) */  
        total += a[i];  
    return total;  
}
```

- Precondition?
  - Identify each point of memory access
  - Write down precondition it requires?
  - Propagate requirement up to beginning of function?

# Software Examples (cont.)

```
/* requires: a != NULL */  
int sum(int a[], size_t n) {  
    int total = 0;  
    for (size_t i=0; i<n; i++)  
        /* requires: a != NULL && 0 <= i && i < size(a) */  
        total += a[i];  
    return total;  
}
```

- Precondition?
  - Identify each point of memory access
  - Write down precondition it requires?
  - Propagate requirement up to beginning of function?
    - Is that sufficient? How do we combine both requirements?

# Software Examples (cont.)

```
/* requires: a != NULL && n <= size(a) */  
int sum(int a[], size_t n) {  
    int total = 0;  
    for (size_t i=0; i<n; i++)  
        /* requires: a != NULL && 0 <= i && i < size(a) */  
        total += a[i];  
    return total;  
}
```

- Precondition?
  - Identify each point of memory access
  - Write down precondition it requires?
  - Propagate requirement up to beginning of function?
    - Is that sufficient? How do we combine both requirements?
    - At this point the proposed invariant will always hold
      - Invariants: conditions that always hold at a given point in a function

# Increase Software Security

- Induction:
  - Another methods of verifying correctness.
  - In case of a more complicated loop
  - Consists of 2 steps:
    - Step 0: verify conditions upon entering the loop
    - Step 1 - Induction: show that the *postcondition* of last statement of loop plus loop test condition implies invariant

# Increase Software Security

- Perform both verification and validation
  - Validation: checking whether the specification captures the customer's needs
  - Verification: checking that the software meets the specifications

# Increase Software Security

- Use formal verification
  - Prove or disprove the correctness of the algorithm
    - Using analysis and mathematical tools
    - Assert that the algorithm is correct with respect to the software specifications

# Bad Practices

- Penetrate-and-patch
  - Fixing problems only after the product has been publicly (and often spectacularly) broken by someone
  - Why is it bad?
    - security should not be an add-on feature
    - problem that is being actively exploited by attackers
- Security by obscurity

# Security through Obscurity



[http://www.treachery.net/articles\\_papers/tutorials/why\\_security\\_through\\_obscurity\\_isnt/index.htm](http://www.treachery.net/articles_papers/tutorials/why_security_through_obscurity_isnt/index.htm)

# Security through Obscurity

- Reliance on the secrecy of the design or implementation
  - as the main system security method
    - or component of a system
- System may have security vulnerabilities
  - System designers believe that if the flaws are not known, it prevents a successful attack



<http://ithare.com/advocating-obscurity-pockets-as-a-complement-to-security-part-ii-deployment-scenarios-and-crypto-primitives-and-obscurity-pocket-as-security>

From *Security in Computing*, Fifth Edition, by Charles P. Pfleiderer, (ISBN 0134085048). Copyright 2015 by Pearson Education, Inc. All rights reserved.

# Security through Obscurity

- Reliance on the secrecy of the design or implementation
  - as the main system security method
    - or component of a system
- Rejected by security experts!
  - obscurity should never be the only security mechanism!
  - has been historically used without success by several organizations



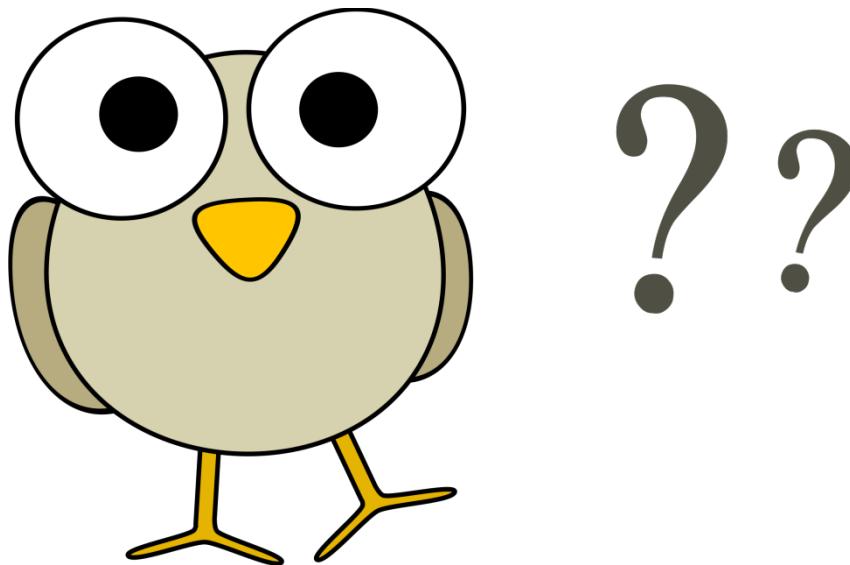
<http://ithare.com/advocating-obscurity-pockets-as-a-complement-to-security-part-ii-deployment-scenarios-and-crypto-primitives-and-obscurity-pocket-as-security>

From *Security in Computing*, Fifth Edition, by Charles P. Pfleiderer, (ISBN: 0789734095048). Copyright 2015 by Pearson Education, Inc. All rights reserved.

# Summary

- Buffer overflow attacks can take advantage of the fact that code and data are stored in the same memory
  - in order to maliciously modify executing programs
- Programs can have a number of other types of vulnerabilities
  - including off-by-one errors, incomplete mediation, and race conditions
- Developers can use a variety of techniques for writing and testing code for security

- Questions?



# COMPUTER SECURITY QUIZ

---

# What is penetration testing?

- A. A procedure for testing libraries or other program components for vulnerabilities
- B. Whole-system testing for security flaws and bugs
- C. A security-minded form of unit testing that applies early in the development process
- D. All of the above

# What is penetration testing?

- A. A procedure for testing libraries or other program components for vulnerabilities
- **B. Whole-system testing for security flaws and bugs**
- C. A security-minded form of unit testing that applies early in the development process
- D. All of the above



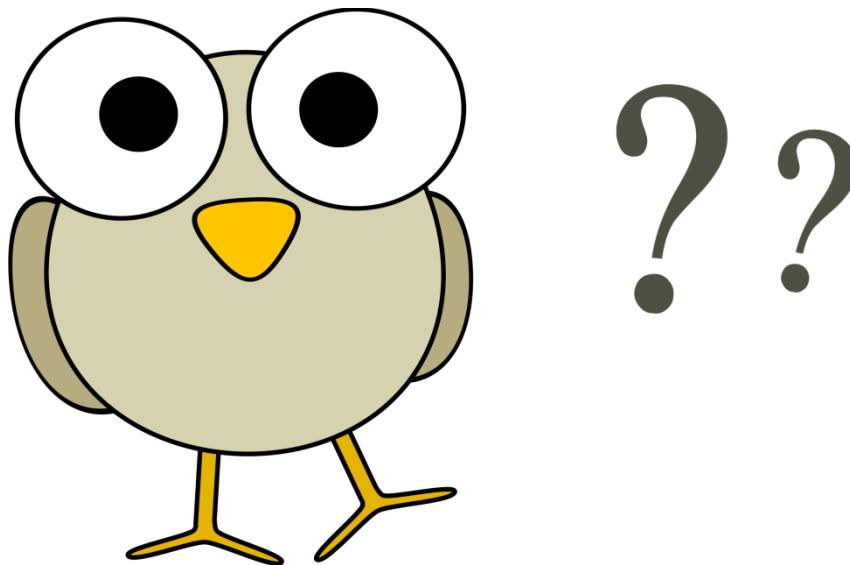
# Which of the following are benefits of penetration testing?

- A. Results are often reproducible
- B. Full evidence of security: a clean test means a secure system
- C. Compositionality of security properties means tested components are secure even if others change
- D. They specifically consider adversarial thinking, which is not usually necessary for normal tests

# Which of the following are benefits of penetration testing?

- A. Results are often reproducible
- B. Full evidence of security: a clean test means a secure system
- C. Compositionality of security properties means tested components are secure even if others change
- D. They specifically consider adversarial thinking, which is not usually necessary for normal tests

- Questions?





# SECURITY PRINCIPLES

---

Making decisions about technology in an uncertain world

<https://criticaluncertainties.com/category/security/saltzer-and-schroeder-principles/>

From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

# Security Principles

- How to design of secure software systems?
- Saltzer and Schroeder's design principles
  - Published in The Protection of Information in Computer Systems [1975]

# Design Principles for Security

- Least privilege
- Economy of mechanism
- Fail-safe defaults
- Open design
- Complete mediation
- Separation of privilege
- Least common mechanism
- Psychological Acceptability - Ease of use

# Least privilege

- Each program and user of a computer system should operate with the bare minimum privileges necessary to function properly.
  - If this principle is enforced, abuse of privileges is restricted, and the damage caused by the compromise of a particular application or user account is minimized.
  - The military concept of need-to-know information is an example of this principle.

# Fail-safe defaults

- Default configuration of a system should have a conservative protection scheme
  - E.g., when adding a new user to an operating system
    - the default group of the user should have minimal access rights to files and services.
  - Unfortunately, operating systems and applications often have default options that favor usability over security.
  - This has been historically the case for a number of popular applications
    - such as web browsers that allow the execution of code downloaded from the web server.

# Economy of mechanism

- This principle stresses simplicity in the design and implementation of security measures.
  - simplicity is especially important in the security domain
    - since a simple security framework facilitates its understanding by developers and users
    - enables the efficient development and verification of enforcement methods for it.
  - This is also applicable to most engineering projects

# Open design

- The security architecture and design of a system should be made publicly available.
  - Security should rely only on keeping crypto keys secret.
  - Open design allows for a system to be scrutinized by multiple parties
    - leads to the early discovery and correction of security vulnerabilities caused by design errors.

# Kerckhoffs's principle

- A cryptosystem should be secure even if everything about the system, except the key, is public knowledge
  - Auguste Kerckhoffs, [19th century]

# Kerckhoffs's principle

- Design principles for military ciphers:
  - System must be practically indecipherable
    - If not mathematically
  - Should not require secrecy
    - it should not be a problem if it falls into enemy hands
  - It must be possible to communicate and remember the key
    - Without written notes
    - Parties must be able to change or modify it at will;

# Kerckhoffs's principle

- Design principles for military ciphers (cont.):
  - Should be portable and applicable to telegraph communications
    - should not require several persons to handle or operate;
  - System must be easy to use
    - should not require users to know and comply with a long list of rules.
      - given the circumstances in which it is to be used, the

# Open design

- The security architecture and design of a system should be made publicly available.
  - Security should rely only on keeping crypto keys secret.
  - Open design allows for a system to be scrutinized by multiple parties
    - leads to the early discovery and correction of security vulnerabilities caused by design errors.
  - The open design principle is the opposite of the approach known as security by obscurity
    - which tries to achieve security by keeping cryptographic algorithms secret
      - has been historically used without success by several organizations.

# Complete mediation

- The idea behind this principle is that every access to a resource must be checked for compliance with a protection scheme.
  - => one should be wary of performance improvement techniques that save the results of previous authorization checks
    - since permissions can change over time.
  - For example, an online banking web site should require users to sign on again after a certain amount of time
    - say, 15 minutes, has elapsed.

# Separation of privilege

- This principle dictates that multiple conditions should be required to achieve access to restricted resources
  - or have a program perform some action.

# Least common mechanism

- In systems with multiple users, mechanisms allowing resources to be shared by more than one user should be minimized.
  - E.g., if a file or application needs to be accessed by more than one user, then these users should have separate channels by which to access these resources
    - to prevent unforeseen consequences that could cause security problems.

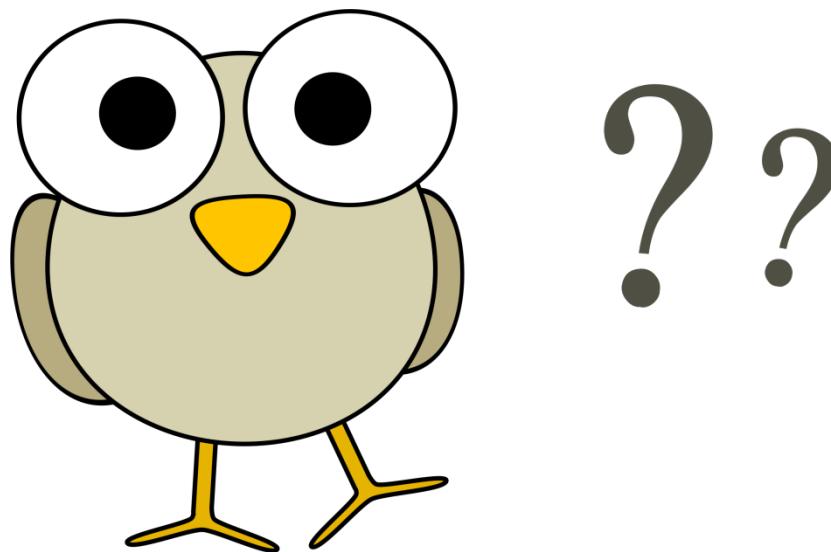
# Psychological acceptability - Ease of use

- This principle states that user interfaces should be **well designed and intuitive**
- All security-related settings should adhere to what an ordinary user might expect.

# Summary

- Malware can have a variety of harmful effects depending on its characteristics, including resource usage, infection vector, and payload
- Developers can use a variety of techniques for writing and testing code for security
- Following design principles helps protect systems against attacks

- Questions?



# WHAT ABOUT INTERNET OF THINGS?

---

# Internet of Things (IoT)

- A network of physical devices, vehicles, home appliances and other items
- Devices are embedded with electronics, software, sensors, actuators
- Network connectivity enables these objects to connect and exchange data
  - But can be a source of vulnerabilities

# “Internet of Things” devices

- “Internet of things” devices are connected through the network
- Attackers may access device vulnerabilities through the network
- Devices are typically very cost sensitive
  - Therefore, very little support after purchase
  - User can not tell if they are secure

# Example of IoT devices

- Wearable devices
- Medical devices
- Home automation
  - Refrigerators, stoves, etc.
- Automative
- Etc.



<https://www.pcworld.com/article/3150801/security/privacy-protections-for-wearable-devices-are-weak-study-says.html>

From *Smart Internet of Things: From Fundamentals to Applications* by Charles P. Eberle et al. (ISBN 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

# IoT Devices

## 10 Smartest IoT Devices



<https://www.pentasecurity.com/blog/10-smallest-iot-devices-2017/>

# “Internet of Things” devices

- Device only communicates through a central service
- Most of the companies running the service are “Data Asset” companies
  - Make their money from advertising, not from the product itself
    - Product may actually be subsidized considerably
  - Companies include Google, Amazon, Salesforce, etc...
  - Apple HomeKit provides a higher level of security
    - But you still need to trust that it does not report to a third party

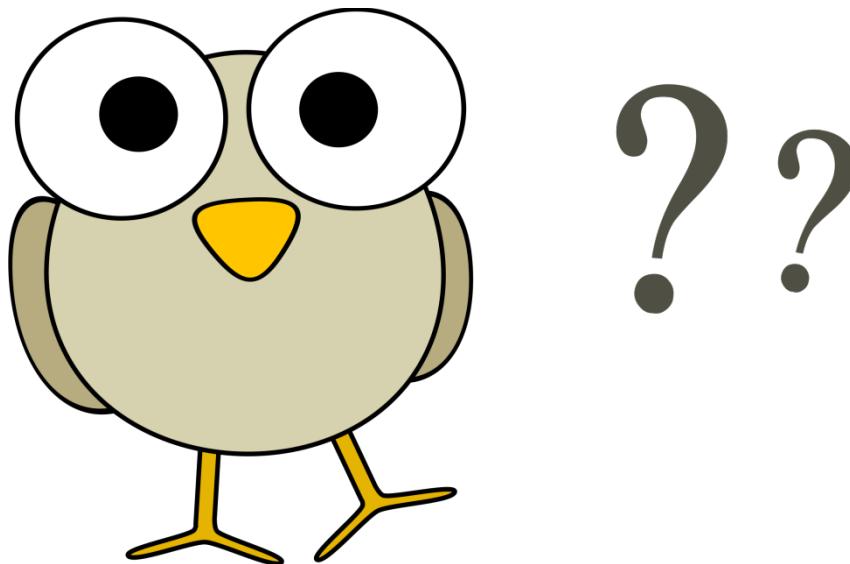
# Risks in using IoT devices

- Devices collect information about user via device
- Many devices do not require strong passwords (or any passwords)
- Devices may use unencrypted network services
- User interface of applications may be vulnerable to different attacks
  - Shoulder surfing, etc.

# “Internet of Things” devices

- Goal: Increase security of devices
- Possible defenses:
  - Develop applications with security in mind
    - Consider security throughout the entire development lifecycle of IoT devices
      - Not treat it separately or as an “add-on”
    - Integrate human understanding and algorithms
      - Take in account the “human factor”

- Questions?



# COMPUTER SECURITY QUIZ

---

# What is a computer network?

- A. A super computer owned only by the government
- B. A web of connected computers or devices
- C. A computer vulnerability
- D. An Internet service provider
- E. All of the above

# What is a computer network?

- A. A super computer owned only by the government
-  B. A web of connected computers or devices
- C. A computer vulnerability
- D. An Internet service provider
- E. All of the above

# Why are cyber vulnerabilities unlikely to ever go away?

- A. Criminals need them to steal identities.
- B. They are side effects of the freedom and ease of communicating online.
- C. They're protected in a secret base on the moon.
- D. The government won't allow people to fix them.

# Why are cyber vulnerabilities unlikely to ever go away?

- A. Criminals need them to steal identities.
- ✓ B. They are side effects of the freedom and ease of communicating online.
- C. They're protected in a secret base on the moon.
- D. The government won't allow people to fix them.

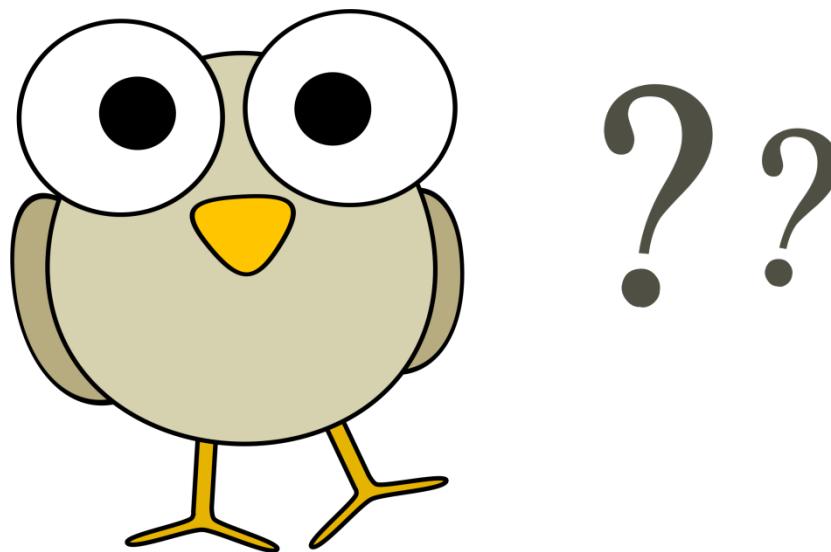
# The size and complexity of networks grew enormously when:

- A. Only governments and universities owned computers.
- B. Spamware caused some computers to break down
- C. The number of personal computers greatly increased.
- D. The hacktivists started using the internet

The size and complexity of networks grew enormously when:

- A. Only governments and universities owned computers.
- B. Spamware caused some computers to break down
-  C. The number of personal computers greatly increased.
- D. The hacktivists started using the internet

- Questions?

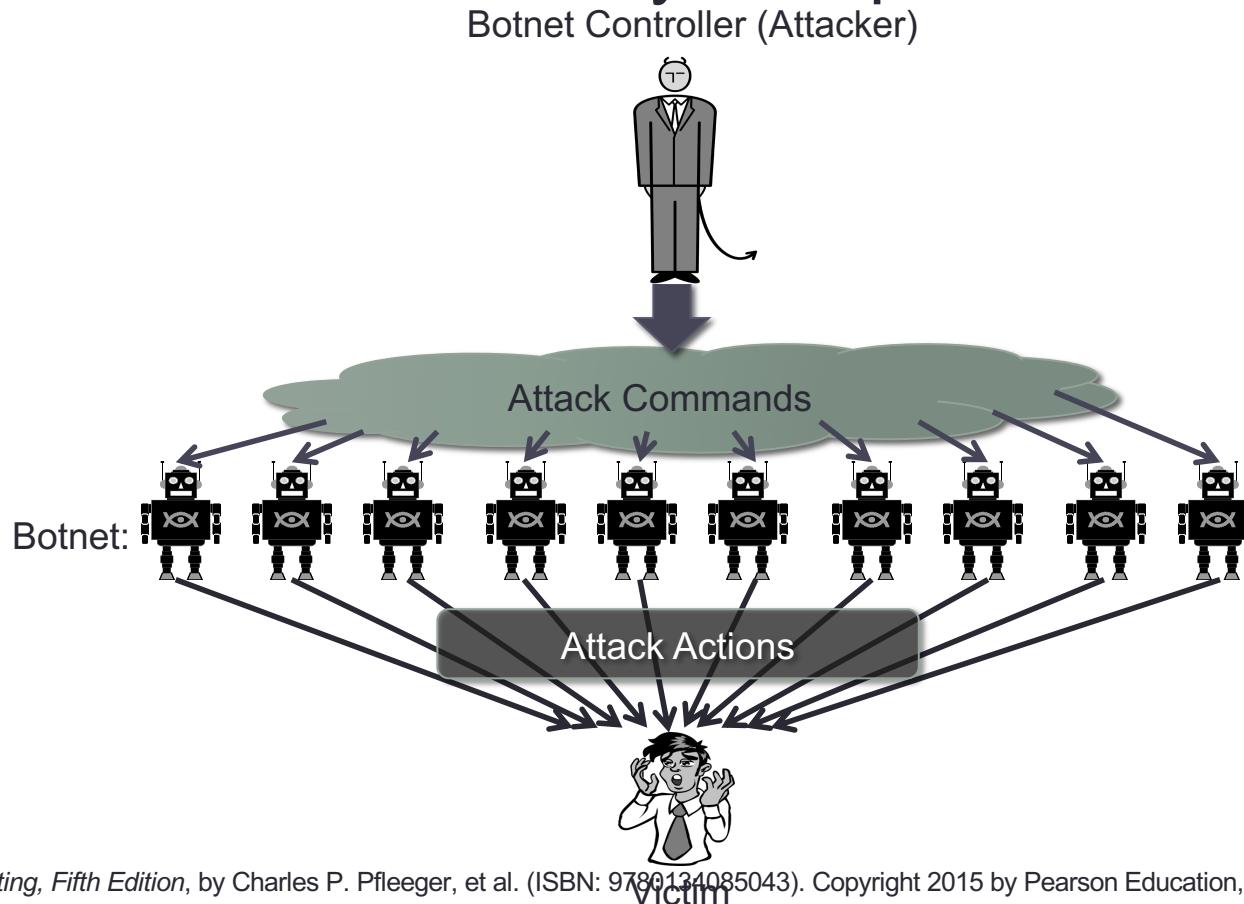


# ADDITIONAL MALWARE

---

# Malware Zombies

- Malware can turn a computer in to a **zombie**, which is a machine that is controlled externally to perform malicious attacks, usually as a part of a **botnet**.



# Insider Attacks



- An **insider attack** is a security breach that is caused or facilitated by someone who is a part of the organization
  - The organization that controls or builds the asset that should be protected
- An insider attack refers to a security hole that is created in a software system
  - by one of its programmers.

<https://www.observeit.com/blog/top-motivating-factors-inside-threats/>

# Backdoors



- A **backdoor**, A.K.A. a **trapdoor**, is a hidden feature or command in a program
- A Backdoor allows a user to perform actions he or she would not normally be allowed to do.
- When used in a normal way, this program performs completely as expected and advertised.
- But if the hidden feature is activated, the program does something unexpected
  - often in violation of security policies, such as performing a privilege escalation.

<https://www.computerhope.com/jargon/b/backdoor.htm>

# Backdoors



- A **backdoor**, A.K.A. a **trapdoor**, is a hidden feature or command in a program
- A Backdoor allows a user to perform actions he or she would not normally be allowed to do.
- Benign example: **Easter Eggs** in DVDs and software

<https://www.computerhope.com/jargon/b/backdoor.htm>

# Logic Bombs



- A **logic bomb** is a program that performs a malicious action as a result of a certain logic condition.
- The classic example of a logic bomb is a programmer coding up the software for the payroll system
  - The programmer puts in code to make the program crash
    - should it ever process two consecutive payrolls without paying him.
- Another classic example combines a logic bomb with a backdoor:
  - a programmer puts in a logic bomb that will crash the program on a certain date.

# The Omega Engineering Logic Bomb



- An example of a logic bomb that was actually triggered and caused damage:
  - Programmer Tim Lloyd was convicted of using on his former employer, Omega Engineering Corporation.
  - On July 31, 1996, a logic bomb was triggered on the server for Omega Engineering's manufacturing operations
  - Ultimately cost the company millions of dollars in damages, led to lay-off of many of its employees.



# The Omega Bomb Code

- The Logic Behind the Omega Engineering Time Bomb included the following strings:
- 7/30/96
  - Event that triggered the bomb
- F:
  - Focused attention to volume F, which had critical files
- F:\LOGIN\LOGIN 12345
  - Login a fictitious user, 12345 (the back door)
- CD \PUBLIC
  - Moves to the public folder of programs
- FIX.EXE /Y F:\\*.\*
  - Run a program, called FIX, which actually deletes everything
- PURGE F:\ALL
  - Prevent recovery of the deleted files

# Defenses against Insider Attacks

- Avoid single points of failure.
- Use code walk-throughs.
- Use archiving and reporting tools.
- Limit authority and permissions.
- Physically secure critical systems.
- Monitor employee behavior.
- Control software installations.

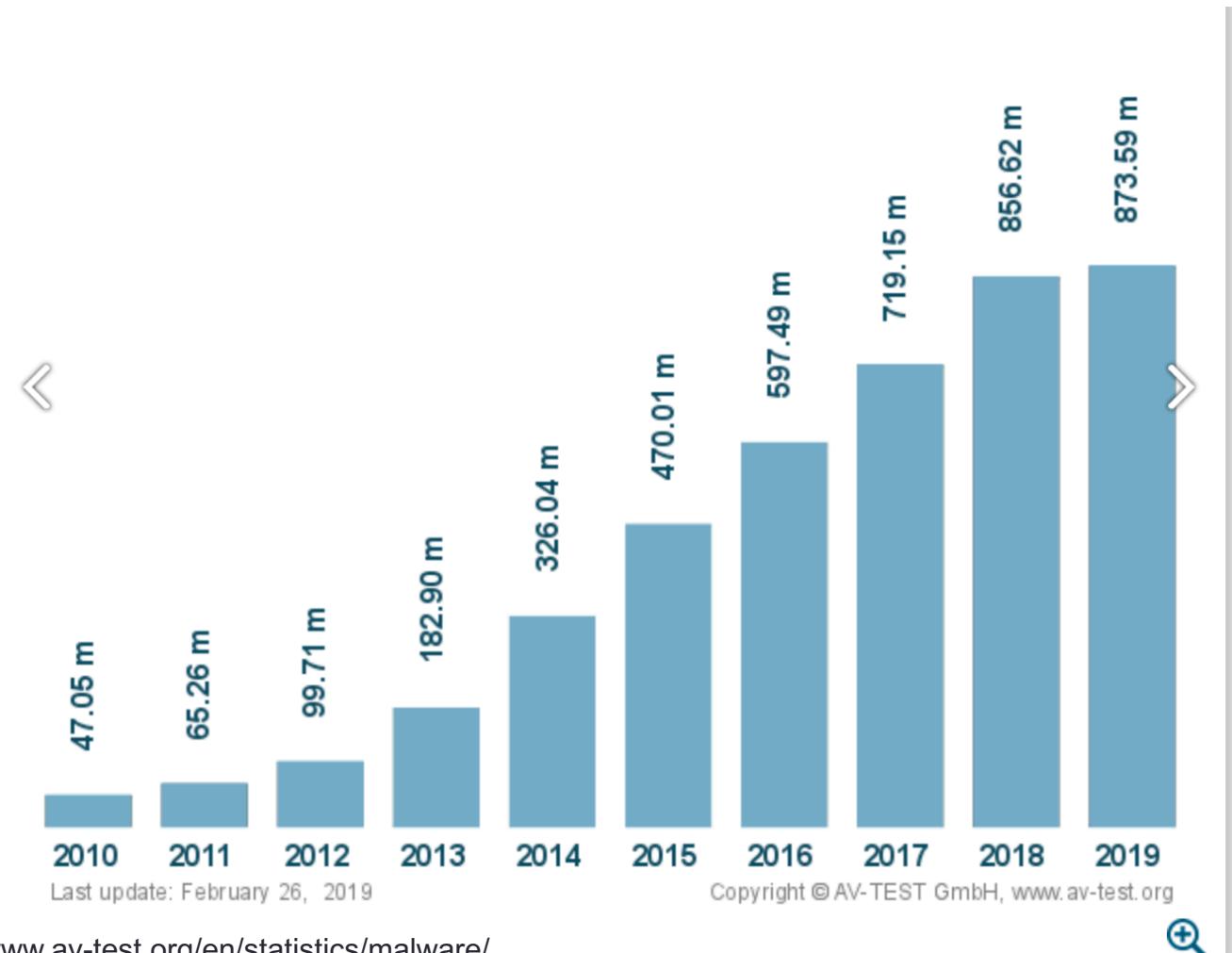


<https://www.cpni.gov.uk/insider-risk-assessment>

# Financial Impact

- Malware often affects a large user population
- Significant financial impact, though estimates vary widely, up to \$100B per year (mi2g)
- Examples
  - LoveBug (2000) caused \$8.75B in damages and shut down the British parliament
  - In 2004, 8% of emails infected by W32/MyDoom.A at its peak
  - In February 2006, the Russian Stock Exchange was taken down by a virus.

# Malware Growth



<https://www.av-test.org/en/statistics/malware/>



From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

# Professional Malware

- Growth in professional cybercrime and online fraud has led to demand for professionally developed malware
- New malware is often a custom-designed variations of known exploits
  - => malware designer can sell different “products” to his/her customers.
- Like every product, professional malware is subject to the laws of supply and demand.
  - Recent studies put the price of a software keystroke logger at \$23 and a botnet use at \$225.

Image by User:SilverStar from <http://commons.wikimedia.org/wiki/File:Supply-demand-equilibrium.svg>  
used by permission under the *Creative Commons Attribution ShareAlike 3.0 License*

# Professional Malware

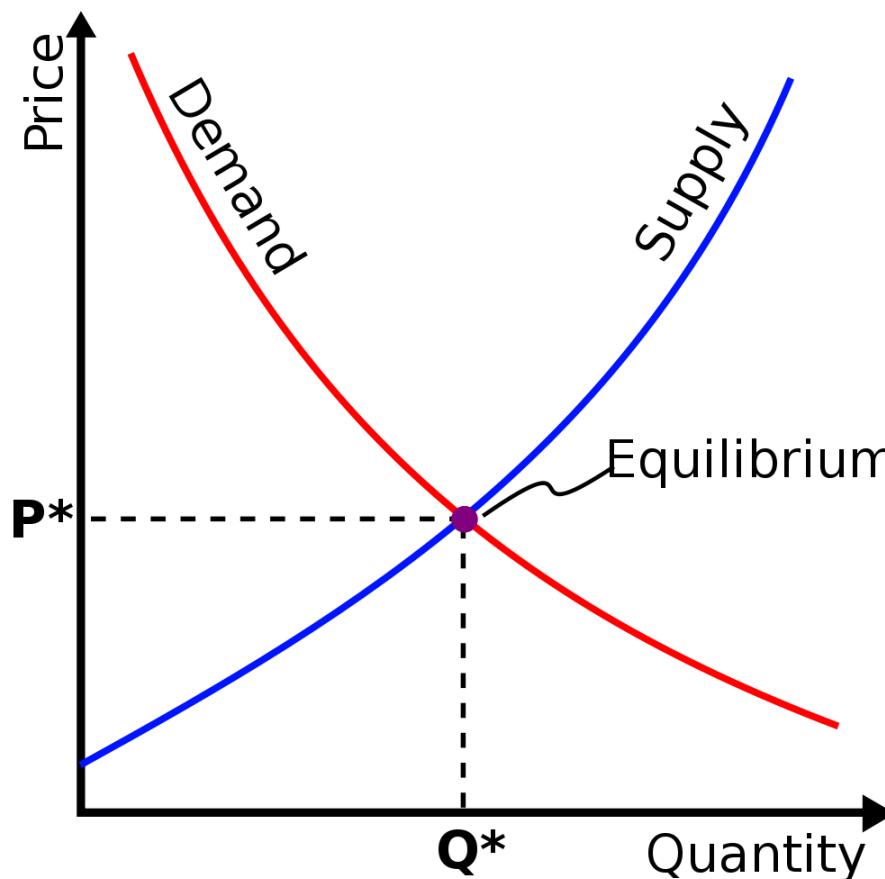


Image by User:SilverStar from <http://commons.wikimedia.org/wiki/File:Supply-demand-equilibrium.svg> used by permission under the *Creative Commons Attribution ShareAlike 3.0 License*

- Questions?

