CISC 7320X - COMPUTER SECURITY

Software Security

PROGRAMS AND PROGRAMMING

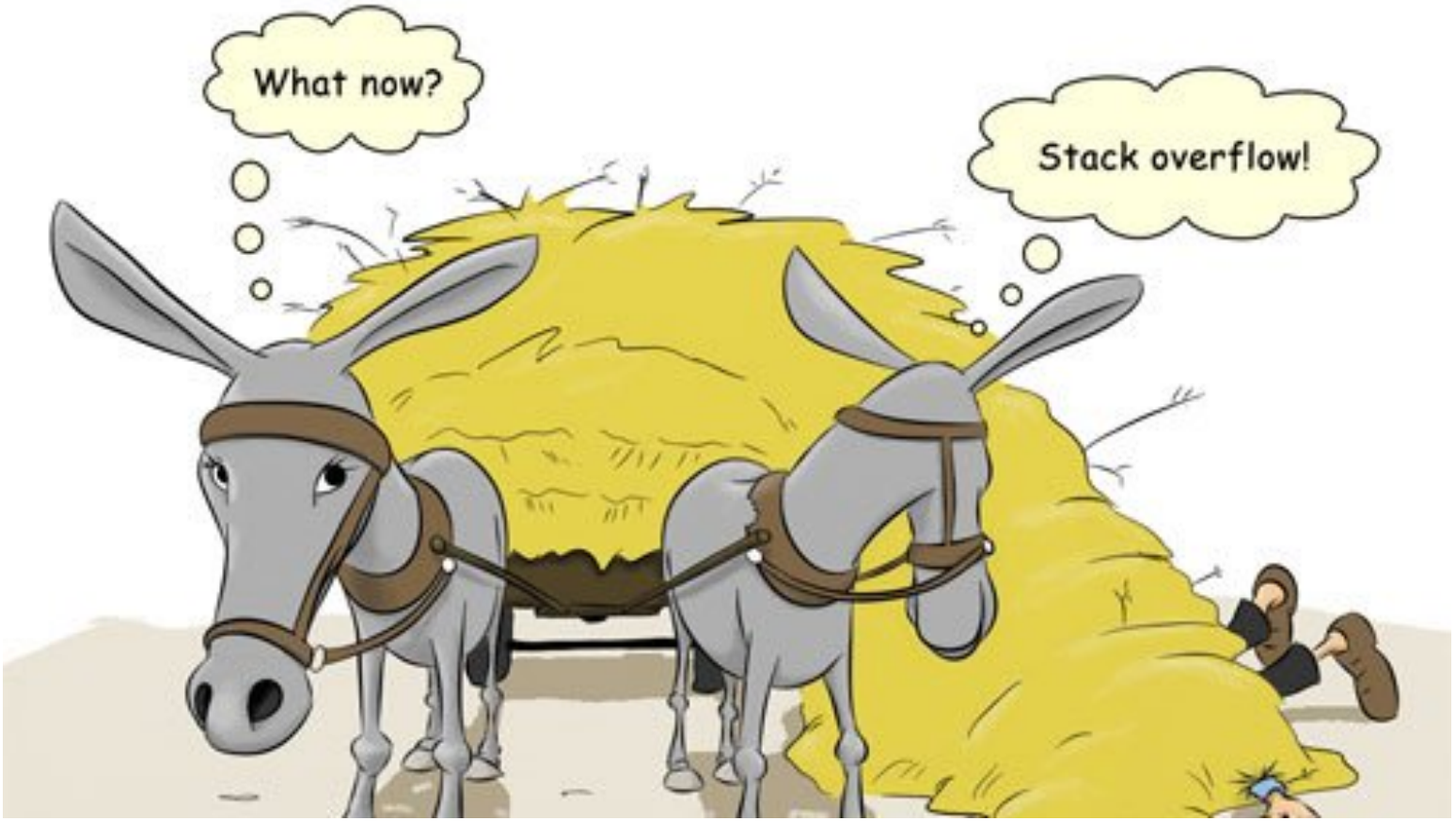
Objectives for today

- Learn about memory organization, buffer overflows, and relevant countermeasures
- Common programming bugs, such as off-by-one errors, race conditions, and incomplete mediation
- Survey of past malware and malware capabilities
- Virus detection
- Tips for programmers on writing code for security

Software Vulnerabilities

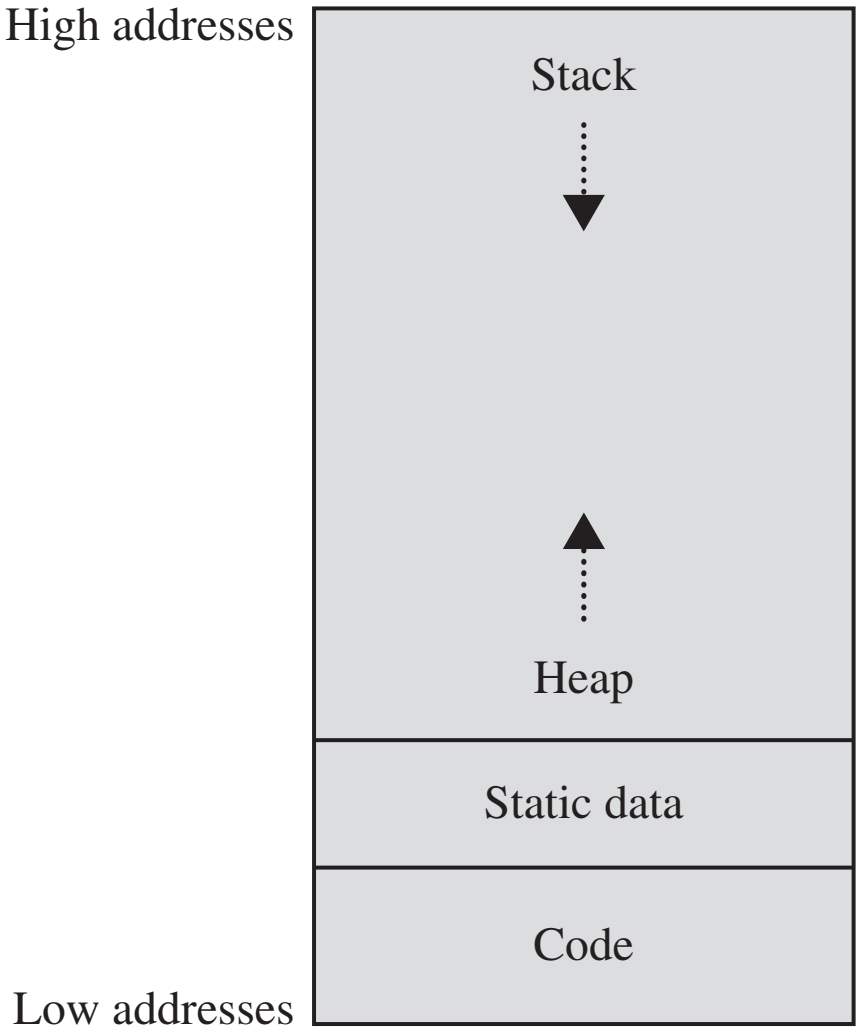
- A weakness in the program
  - May be explored by an attacker
    - Causing system to behave differently then expected
- Attacks may be based on the operating system and programming language
- Goals include manipulating the computer's memory or control program's execution

Buffer Overflow



https://www.linkedin.com/pulse/buffer-overflow-exploits-protection-mechanisms-roman-postaru/

Memory Allocation



Buffer Overflows

- Occur when data is written beyond the space allocated for it, such as a 10th byte in a 9-byte array
- In a typical exploitable buffer overflow, an attacker's inputs are expected to go into regions of memory allocated for data, but those inputs are instead allowed to overwrite memory holding executable code
- The trick for an attacker is finding buffer overflow opportunities that lead to overwritten memory being executed, and finding the right code to input

How Buffer Overflows Happen

```
char sample[10];

int i;

for (i=0; i<=9; i++)
   sample[i] = 'A';

sample[10] = 'B';
```

Programming Errors

- **Buffer overflow** attack can cause crash
  - Input is longer than variable buffer, overwrites other data
  - Example: program in C:

```
char A[8] = "";
unsigned short B = 1979;
```

Buffer Overflow (example)

Initially, A contains nothing but zero bytes, and B contains the number 1979

variable name	A								B	
value	[null string]								1979	
hex value	0 0	00	00	00	00	00	00	00	07	BB

Buffer Overflow (example)

- The program attempts to store the null-terminated string "excessive“

```
strcpy(A, "excessive");
```

- "excessive" is 9 characters long and encodes to 10 bytes including the null terminator
  - but A can take only 8 bytes
  - By failing to check the length of the string, it also overwrites the value of B:

Buffer Overflow (example)

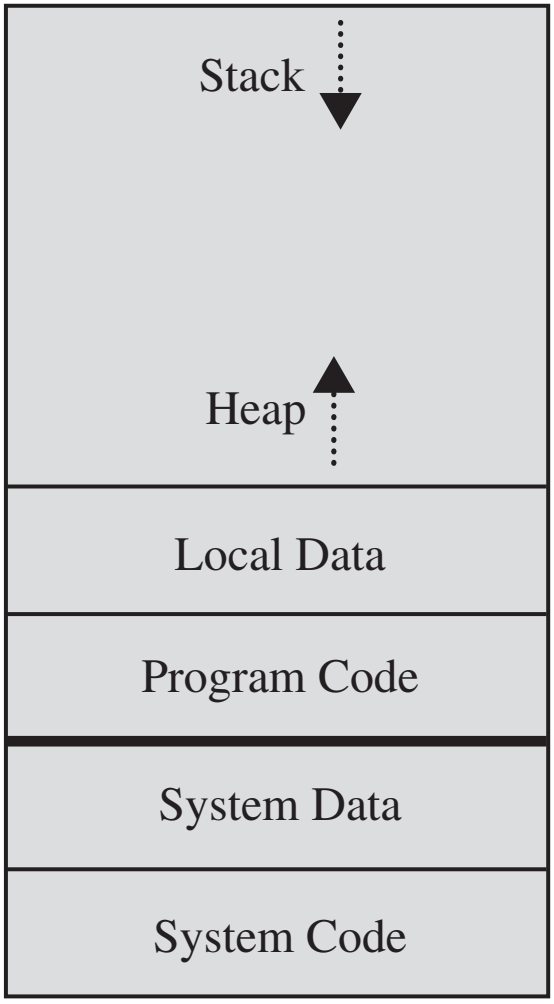
variable name	A								B	
value	'e'	'x'	'c'	'e'	's'	's'	'i'	'v'	25856	
hex	65	78	63	65	73	73	69	76	65	00

Buffer Overflow (cont.)

- How to prevent this?
- Replace call to strcpy with strncpy
  - strncpy takes the maximum capacity of A as an additional parameter
    - ensures that no more than this amount of data is written to A:

```
strncpy(A, "excessive", sizeof(A));
```

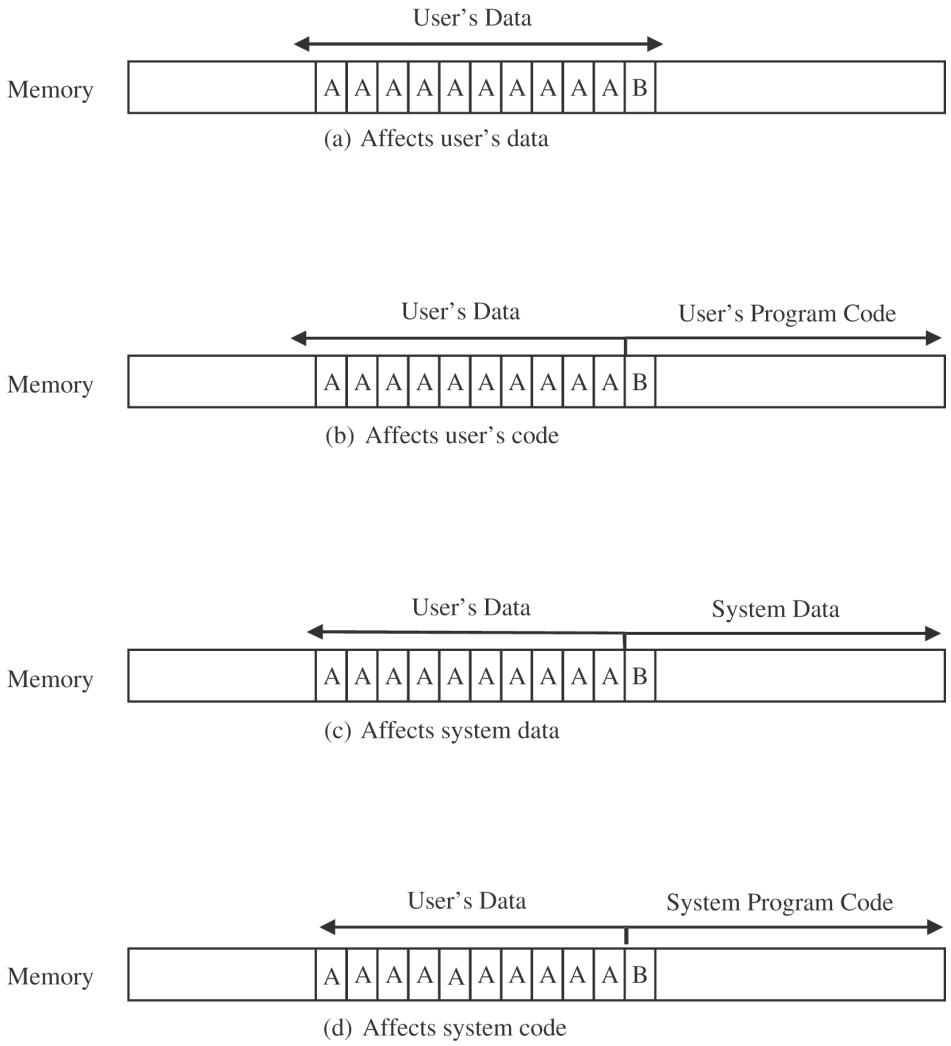
Memory Organization



High addresses

Low addresses

Where a Buffer Can Overflow



Buffer Overflow Attack

- Buffer overflow can happen by accident
  - Or a malicious attack
- Example of an attack on a web server:
  - The server accepts user input from a name field on a web page
    - Into an unchecked buffer variable
    - The attacker supplies a malicious code as input
  - The code read by the server overflows part of the application code
  - The web server now runs the malicious code

Harm from Buffer Overflows

- Overwrite:
  - Another piece of your program's data
  - An instruction in your program
  - Data or code belonging to another program
  - Data or code belonging to the operating system
- Overwriting a program's instructions gives attackers that program's execution privileges
- Overwriting operating system instructions gives attackers the operating system's execution privileges

Buffer Overflow Attack

- Buffer overflows are a major source of security issues
- Software tools help decrease server issues
  - But vulnerability still exists and exploited
- Buffer overflows account for 14% of all vulnerabilities in the last 25 years
  - But 23% of the high severity vulnerabilities and 35% of critical vulnerabilities!

HOME | CATEGORIES | ROGUEWAVE.COM

Buffer overflows are the top software security vulnerability of the past 25 years

ON MAR 11, 13 • BY CHRIS BUBINAS • WITH 2 COMMENTS

In a report analyzing the entire CVE and NVD databases, which date back to 1988, Sourcefire senior research engineer Yves Younan found that vulnerabilities have generally decreased over the past couple of years before rising again in 2012. Younan suggested that efforts to improve security through the use of tools such...

Tweet Like 0 G+

HOME » SOFTWARE SECURITY » BUFFER OVERFLOWS ARE THE TOP SOFTWARE SECURITY VULNERABILITY OF THE PAST 25 YEARS

Year after year, buffer overflows have been a major source of software security issues, ranking as a top vulnerability throughout many of the last 25 years, according to a recent analysis from Sourcefire. In a report analyzing the entire CVE and NVD databases, which date back to 1988, Sourcefire senior research engineer Yves Younan found that vulnerabilities have generally decreased over the past couple of years before rising again in 2012. Younan suggested that efforts to improve security through the use of tools such as static analysis have also helped reduce the number of issues with high severity classifications.

Search

OUR SOCIAL NETWORKS

RECENT POSTS

Migrating to CentOS saves you money

Learn why switching from RHEL to CentOS saves you money...

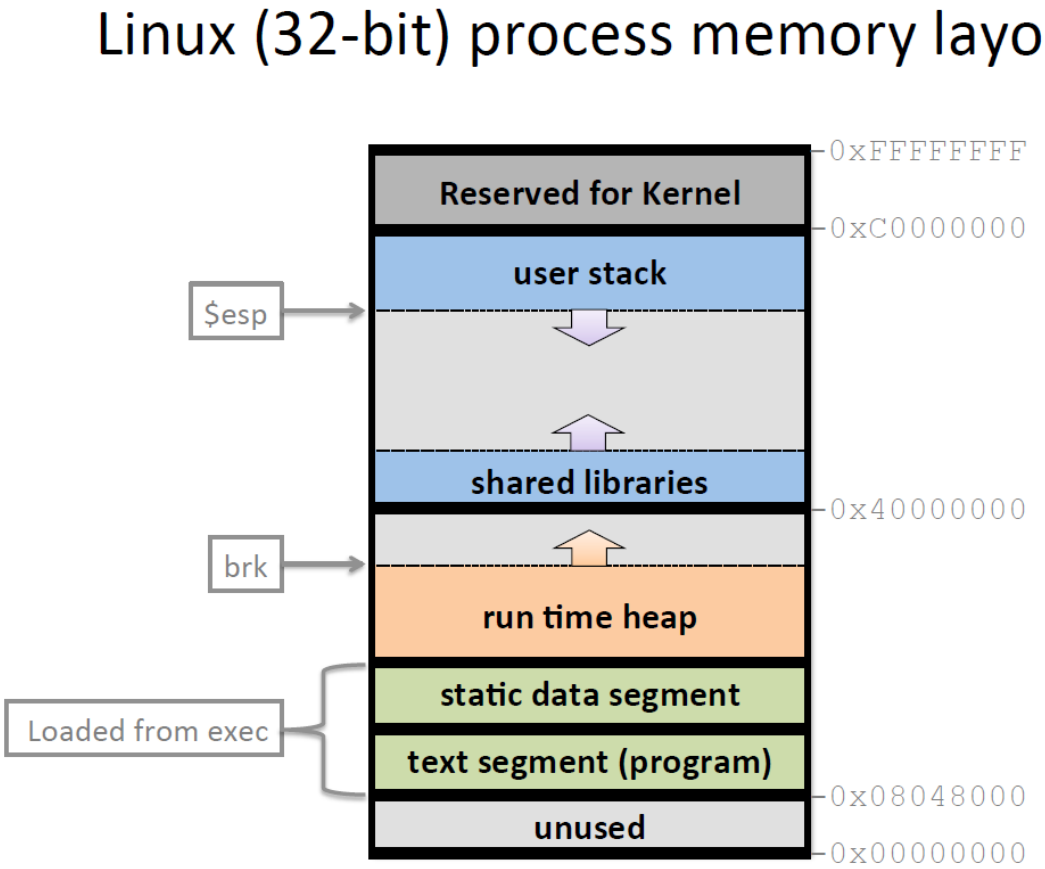
Meltdown and Spectre: How they work and how to patch

Details on the Meltdown and Spectre vulnerabilities, how they work and how to patch your system...

Code Injection

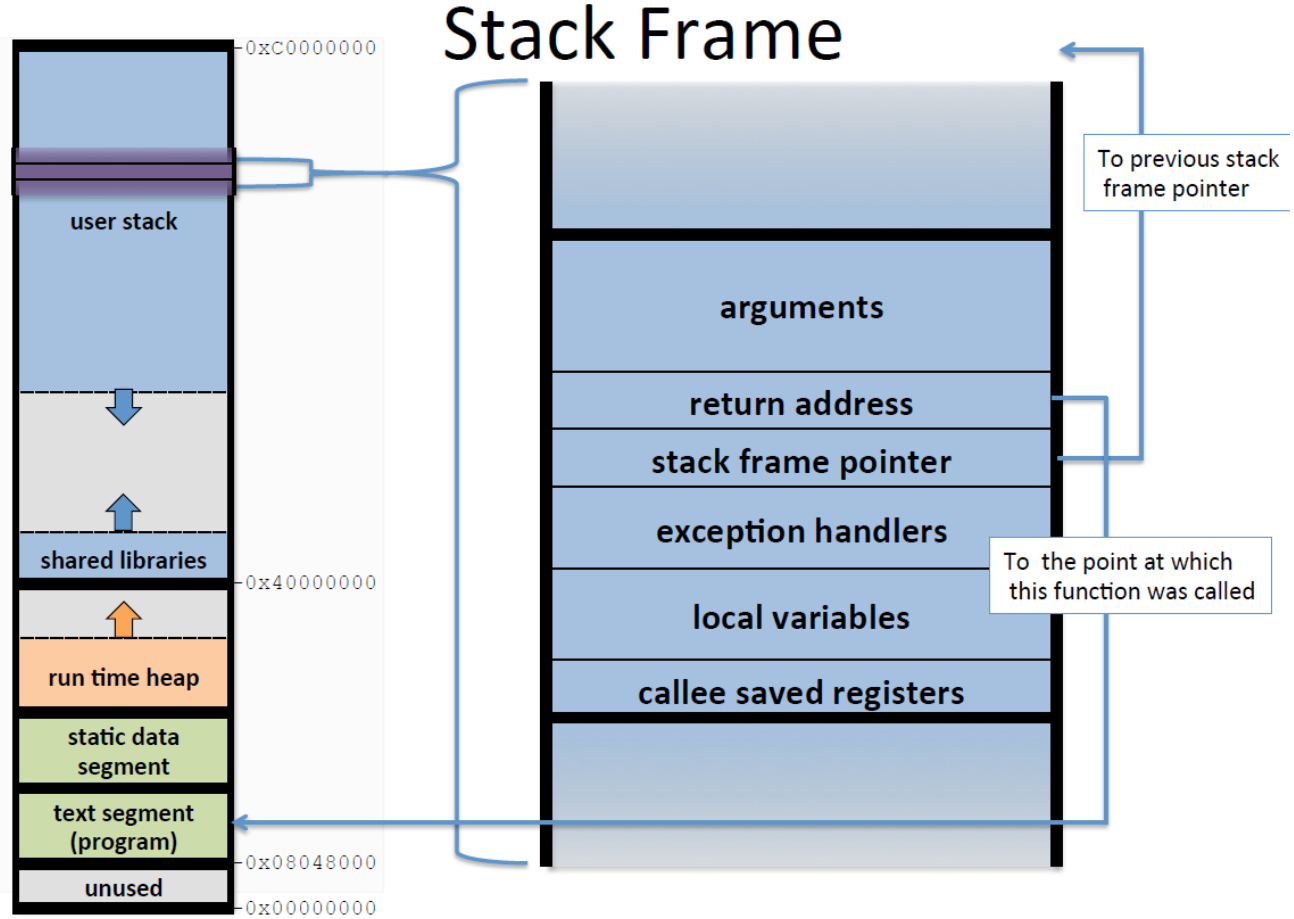
- Exploitation of a computer bug that is caused by processing invalid data.
- Code injected into a vulnerable computer program and changes the course of execution
- Results can be disastrous
  - May allow computer worms to propagate

Code Injection Attack Example

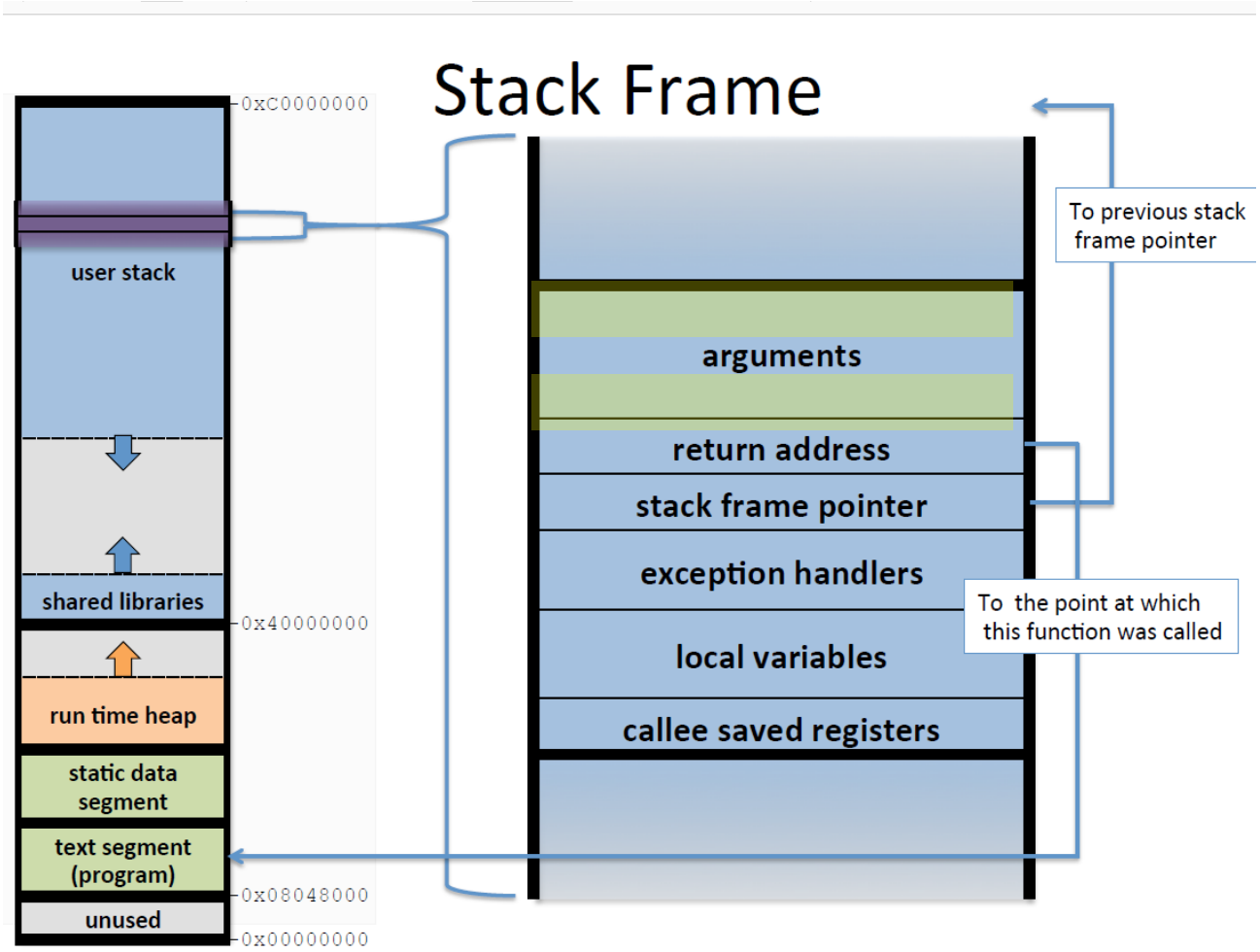


Code Injection Attack Example

- What makes frame vulnerable to attacks?



Code Injection Attack Example



Code Injection

- Basic Stack Exploit:
  - Overwriting the return address allows an attacker to redirect the flow of program control.
  - Instead of crashing, this can allow arbitrary code to be executed.

Code Injection

- Basic Stack Exploit example:
  - attacker chooses malicious code he wants executed ("shellcode"), compiles to bytes
  - includes this in the input to the program so it will get stored in memory somewhere
  - overwrites return address to point to it.

Code Example 1

- Is this function safe?

```
void vulnerable() {
        char buf[64];
        …
        gets(buf);
        …
        }
```

- How can you make it safer?

Code Example 1 (cont.)

- Is this function safe?

```
void safe() {
        char buf[64];
        …
        fgets(buf, 64, stdin);
        …
        }
```

- Can we make it safer?

Code Example 1 (cont.)

- Is this function safe?

```
void safe() {
        char buf[64];
        …
        fgets(buf, 64, stdin);
        …
        }
```

- What happens when the function changes over time?

Code Example 1 (cont.)

- Function grows after a while...

```
void safe() {
        char buf[64];
        …
        …
        …
        …
        …
        fgets(buf, 64, stdin);
        …
        }
```

Code Example 1 (cont.)

- A bigger buffer may be needed, a change may (eventually) occur:

```
void safe() {
        Char buf[64];
        …
        …



        …
        …
        fgets(buf, 128,stdin)
```

- How can we make it safer?

Code Example 1 (cont.)

```
void safer() {
        char buf[64];
        …
        fgets(buf, sizeof buf, stdin);
        …
        }
```

Code Example 2

- Is this function safe?

```
void vulnerable(int len, char *data) {
  char buf[64];
  if (len > 64)
    return;
  memcpy(buf, data, len);
}
```

```
memcpy(void *s1, const void *s2, size_t n);
```

Code Example 2

- Size_t is an unsigned integer type
- All signed integer negative values change to a high positive number when converted to an unsigned type
  - In libc routines, such as memcpy
- How can an attacker exploit this?

Code Example 2

- Size_t is an unsigned integer type
- All signed integer negative values change to a high positive number when converted to an unsigned type
- Malicious users can often specify negative integers through various program interfaces
  - undermine an application's logic
- This happens commonly when a maximum length check is performed on a user-supplied integer, but no check is made to see whether the integer is negative

Code Example 2

```
void safe(size_t len, char *data) {
  char buf[64];
  if (len > 64)
    return;
  memcpy(buf, data, len);
}
```

Code Example 3

```
void f(size_t len, char *data) {
  char *buf = malloc(len+2);
  if (buf == NULL) return;
  memcpy(buf, data, len);
  buf[len] = '\n';
  buf[len+1] = '\0';
}
```

- Is it safe?
  - No, vulnerable!
  - If len = 0xffffffff, then program allocates only 1 byte
    - Overflows

Causes of Software Vulnerabilities

- Human-errors: human error is a significant source of software vulnerabilities and security vulnerabilities
  - Solution? Use automated tools
- Awareness: programmers not be focused on security
  - May be unaware of consequences
  - Solution: Learn about common types of security flaws
- Design flaws: software and hardware have design flaws and bugs
  - May not be designed well for security
    - Use better languages (Java, Python…)

Causes of Software Vulnerabilities

- Complexity: software vulnerabilities rise proportionally with complexity
  - Solution: design simple and well-documented software
- User input: accepting user input by internet can introduce software vulnerabilities
  - Data may be incorrect or fraudulent
    - Data can be designed to attack the receiving system
  - Solution: sanitize user input

Testing for Software Security Issues

- What makes testing a program for security problems difficult?
  - If programmer doesn't make mistake, program will run correctly
  - We need to test for the absence of something
    - Security is a negative property!
      - "nothing bad happens, even in really unusual circumstances"
  - Normal inputs rarely stress security-vulnerable code

Testing for Software Security Issues

- How can we test more thoroughly?
  - Use random inputs
  - Create a testing plan
    - Allows defining a range of inputs

https://www.avyaancom/blog/why-hire-a-software-security-testing-company/

http://www.testnbug.com/2015/02/software-testing-types/

Testing for Software Security Issues

- How can we test more thoroughly?
  - Use random inputs
  - Create a testing plan
    - Allows defining a range of inputs
- How do we tell if a problem was found?
  - Crash or other deviant behavior;
    - now more expensive checks justified

https://www.avyaan.com/blog/why-hire-a-software-security-testing-company/

http://www.testnbug.com/2015/02/software-testing-types/

WORKING TOWARDS SECURE SYSTEMS

Common Software Vulnerabilities

- Memory safety vulnerabilities
- Input validation vulnerabilities
- Race conditions
- Time-of-Check to Time-of-Use (TOCTTOU) vulnerability

Memory safety vulnerabilities

- Software bugs and security vulnerabilities when dealing with memory access, such as buffer overflows and dangling pointers
- Java is said to be memory-safe
  - its runtime error detection checks array bounds and pointer dereferences (*p).
- In contrast, C and C++ support arbitrary pointers with no provision for bounds checking
  - thus are termed **memory-unsafe**

Input validation vulnerabilities

- Program requires certain assumptions on inputs to run properly
- When program does not validate input correctly, it may get exploited
- Buffer overflow and SQL injection are just a few of the **attacks** that can result from improper data **validation**

Input validation vulnerabilities

- Input Validation Example:
  - Bank money transfer:
    - Check that amount to be transferred is non-negative and no larger than payer's current balance

Input validation vulnerabilities

- **SQL injection** is a code **injection** technique
  - used to attack data-driven applications
  - nefarious **SQL** statements are inserted into an entry field for execution
    - e.g. to dump the database contents to the attacker
- **Directory traversal** or **Path Traversal** is an HTTP attack
  - allows attackers to access restricted **directories**
  - execute commands outside of the web server's root **directory**

Input validation vulnerabilities

- **Cross-site scripting** is a computer security vulnerability
  - typically found in web applications
  - XSS enables attackers to inject client-side scripts into web pages viewed by other users

Overflow Countermeasures

- Staying within bounds
  - Check lengths before writing
  - Confirm that array subscripts are within limits
  - Double-check boundary condition code for off-by-one errors
  - Limit input to the number of acceptable characters
  - Limit programs' privileges to reduce potential harm

Overflow Countermeasures

- Many languages have overflow protections
- Code analyzers can identify many overflow vulnerabilities
- Canary values in stack to signal modification

Incomplete Mediation

- Mediation: 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

Race Condition

- Output is dependent on the sequence or timing of other uncontrollable events
- Becomes a vulnerability when events happen in a different order
  - than the programmer intended

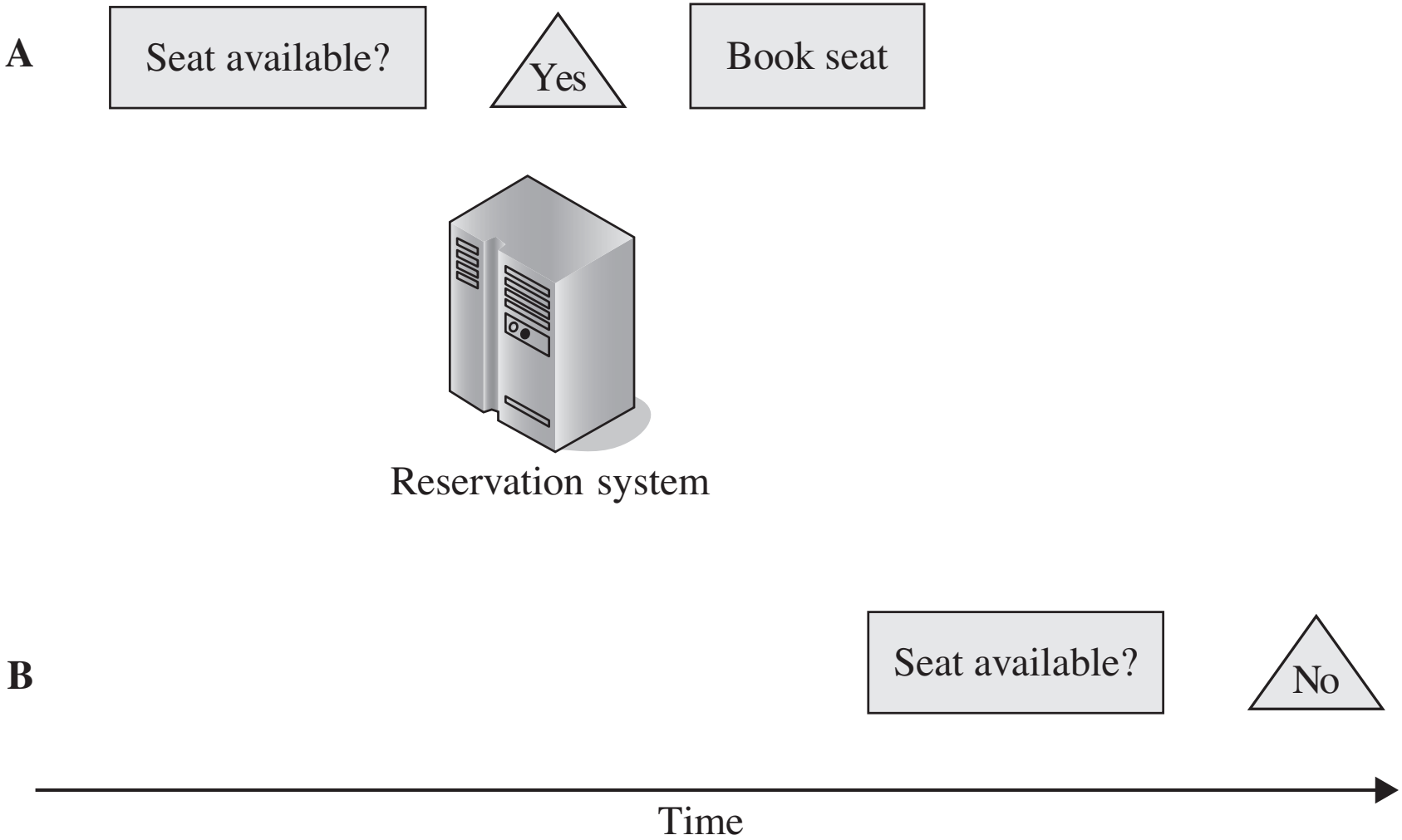
Time-of-Check to Time-of-Use (TOCTTOU) vulnerability

- Software bugs caused by changes in a system
  - Between the checking of a condition (such as a security credential) and the use of the results of that check
- An example of a race condition

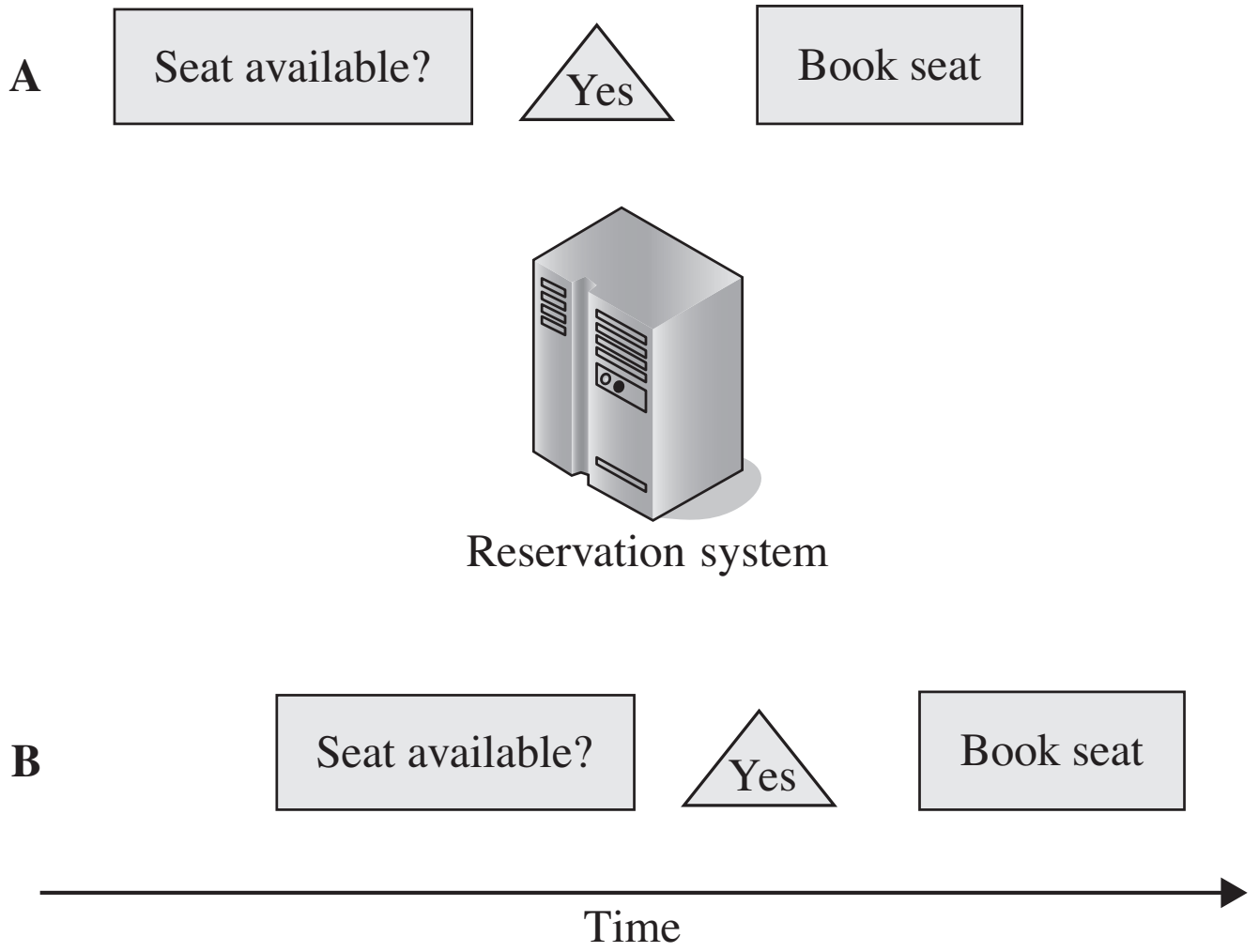
Race Condition Examples:

- Booking a seat on a place:
  - No race condition:
    - A booker books the last seat on the plane
    - thereafter the system shows no seat available
  - Race condition:
    - Before the first booker can complete the booking for the last available seat, a second booker looks for available seats
    - This system has a race condition, where the overlap in timing of the requests causes errant behavior

Example: No Race Condition



Example: Race Condition Exists



Other Programming Oversights

- Undocumented access points (backdoors)
- Off-by-one errors
  - Occurs when an iterative loop iterates one time too many or too few
  - May occur when:
    - Using “<=" where “<" should have been used in a comparison
    - A sequence starts at zero rather than one (as with array indices in many languages)

Other Programming Oversights

- Integer overflows
  - Attempt to create a numeric value outside of the range that can be represented with a given number of digits
    - either larger than the maximum or lower than the minimum representable value
- Unterminated null-terminated string
- Parameter length, type, or number errors
- Unsafe utility libraries

Working Towards Secure Systems

- Along with securing individual components, we need to keep them up to date …
  - New software versions are constantly released
    - Keeping up with other competitive applications, OS changes, add new features.
- What's hard about patching?
  - Can break crucial functionality inadvertently
  - Management burden:
    - It never stops (the "patch treadmill") …

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
- Virus: A program that can replicate itself
  - pass on malicious code to other nonmalicious 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

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

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
- Various types of payloads, ranging from annoyance to crime

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

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

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

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

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

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

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

Malware Activation

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

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

MALWARE TYPES

Computer Viruses

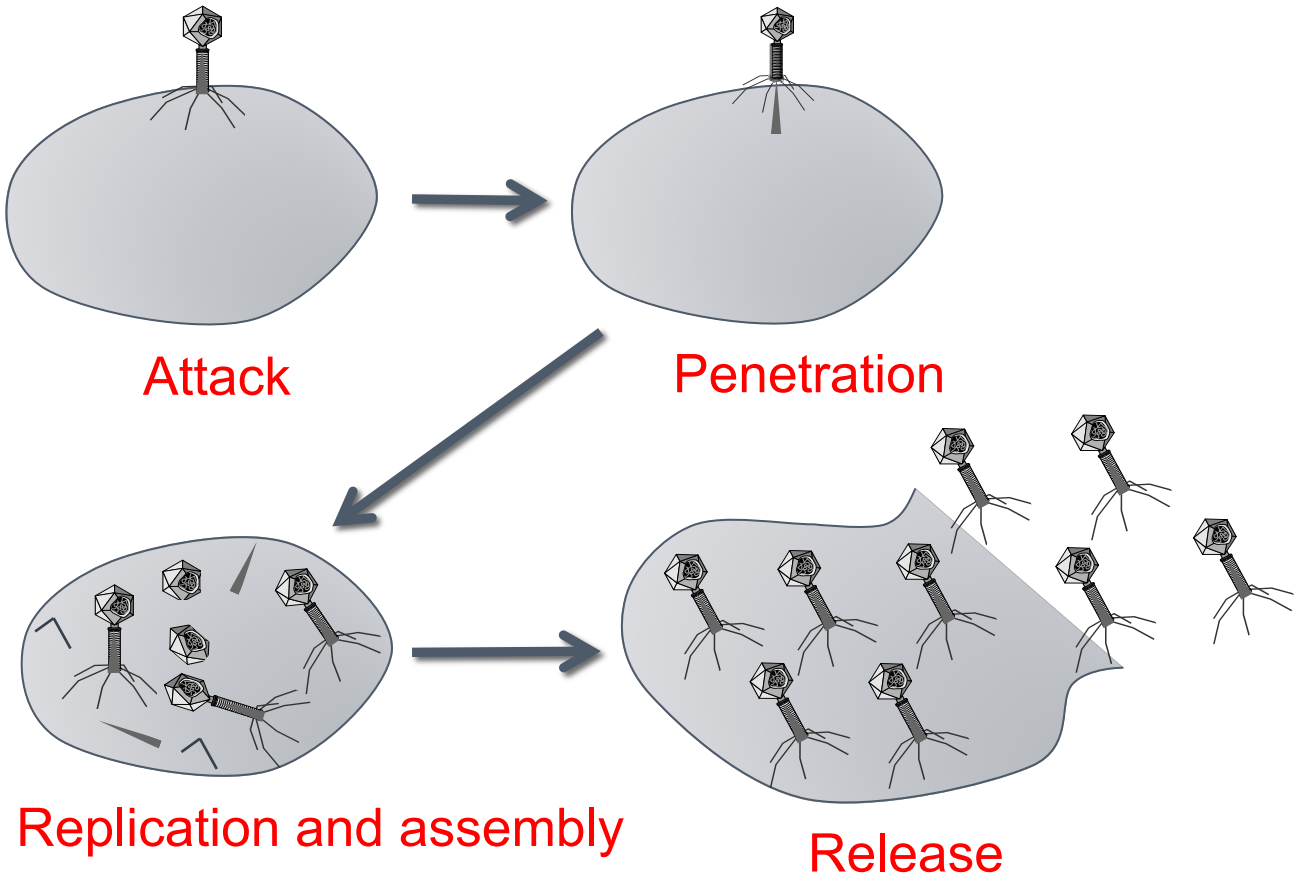
- A **computer virus** is computer code that can replicate itself by modifying other files/programs
  - to insert code that is capable of further replication.
- This self-replication property distinguishes computer viruses from other kinds of malware
  - such as logic bombs.
- Another distinguishing 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

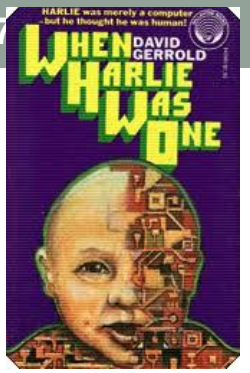
From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

Biological Analogy

- Computer viruses share some properties with Biological viruses



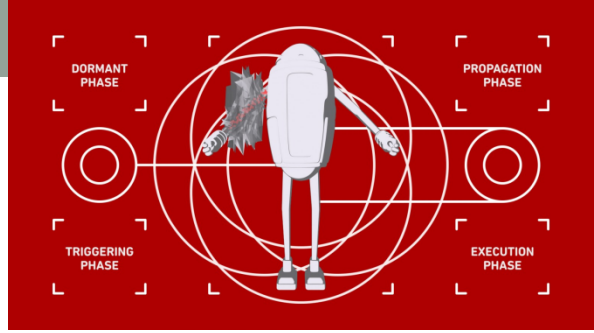
From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.



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
- 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

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

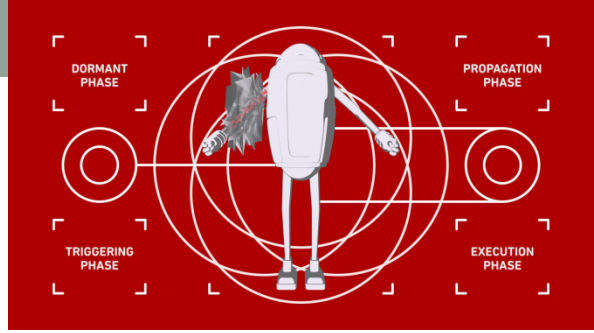


Virus Phases

- **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/

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



Virus Phases

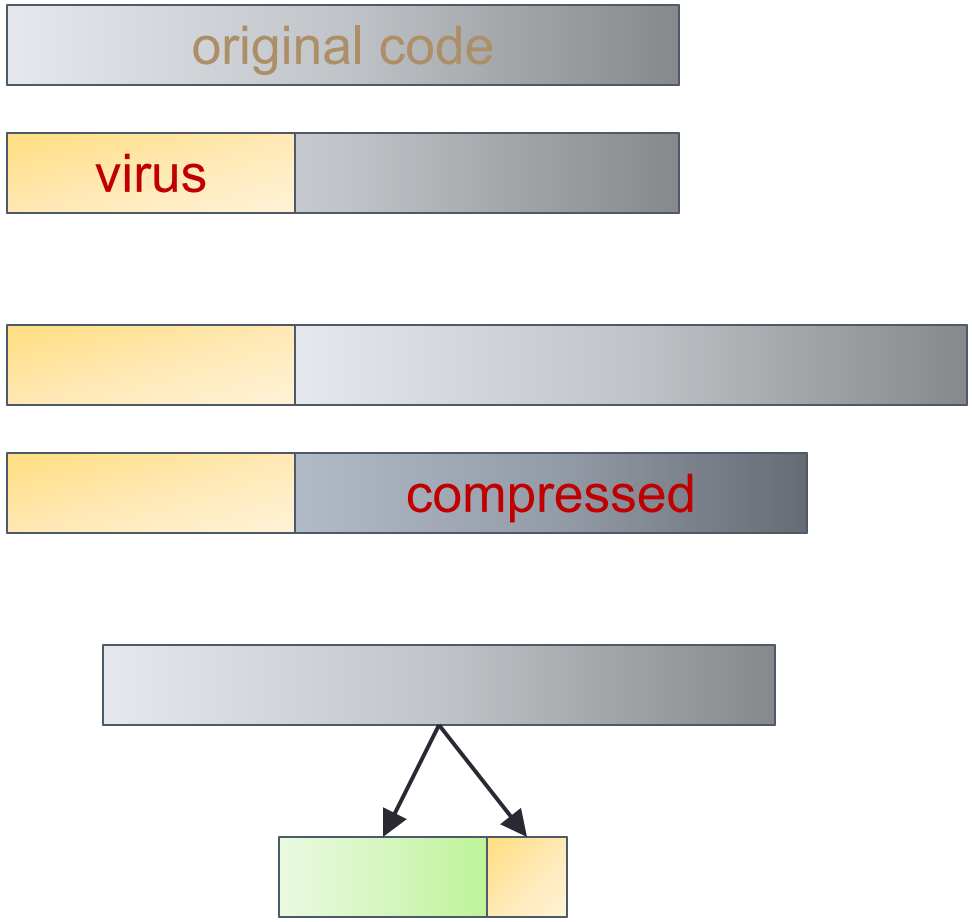
- **Action phase:**
  - The virus performs the malicious action that it was designed to perform, called **payload.**
    - This action could include something seemingly innocent, like displaying a silly picture on a computer's screen
      - or something quite malicious, such as deleting all essential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

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

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

Infection Types

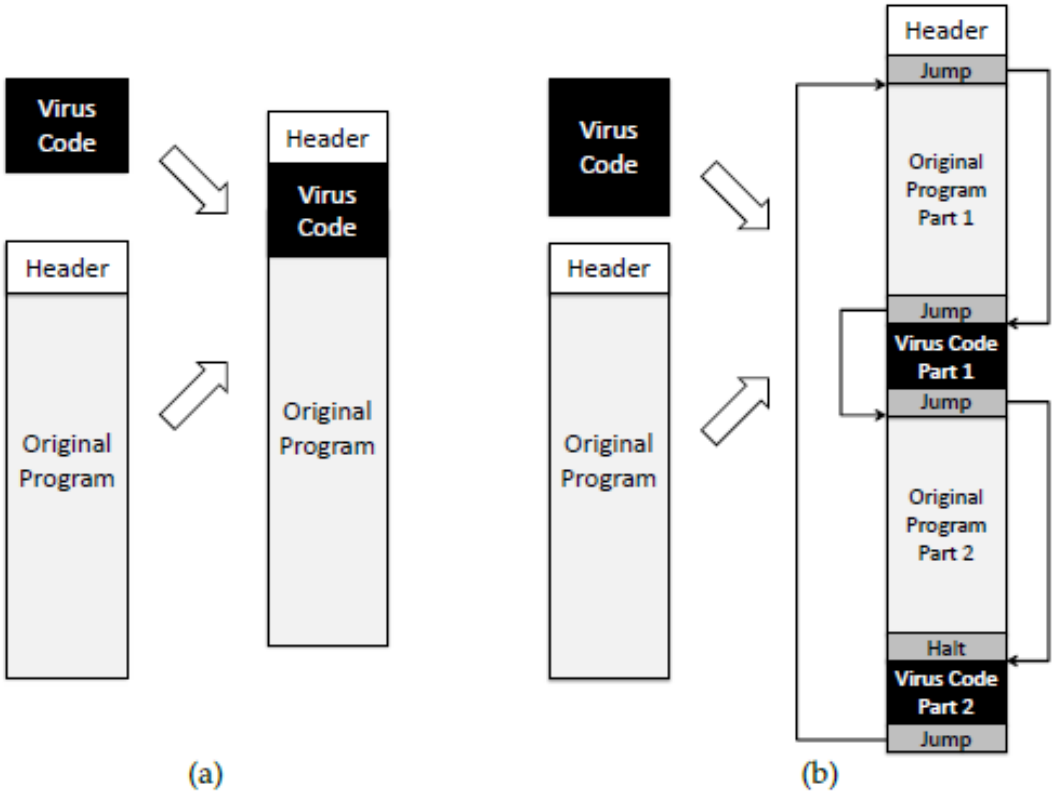
- Overwriting
  - Destroys original code
- Pre-pending
  - Keeps original code, possibly compressed
- Infection of libraries
  - Allows virus to be memory resident
  - E.g., kernel32.dll
- Macro viruses
  - Infects MS Office documents
  - Often installs in main document template



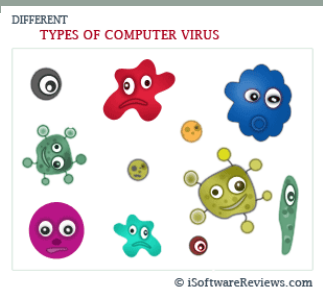
From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

Degrees of Complication

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



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



Concealment

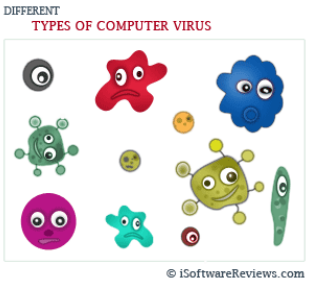
- Encrypted virus
  - Decryption engine + encrypted body
  - Randomly generate encryption key
  - Detection looks for decryption engine

- 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

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

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

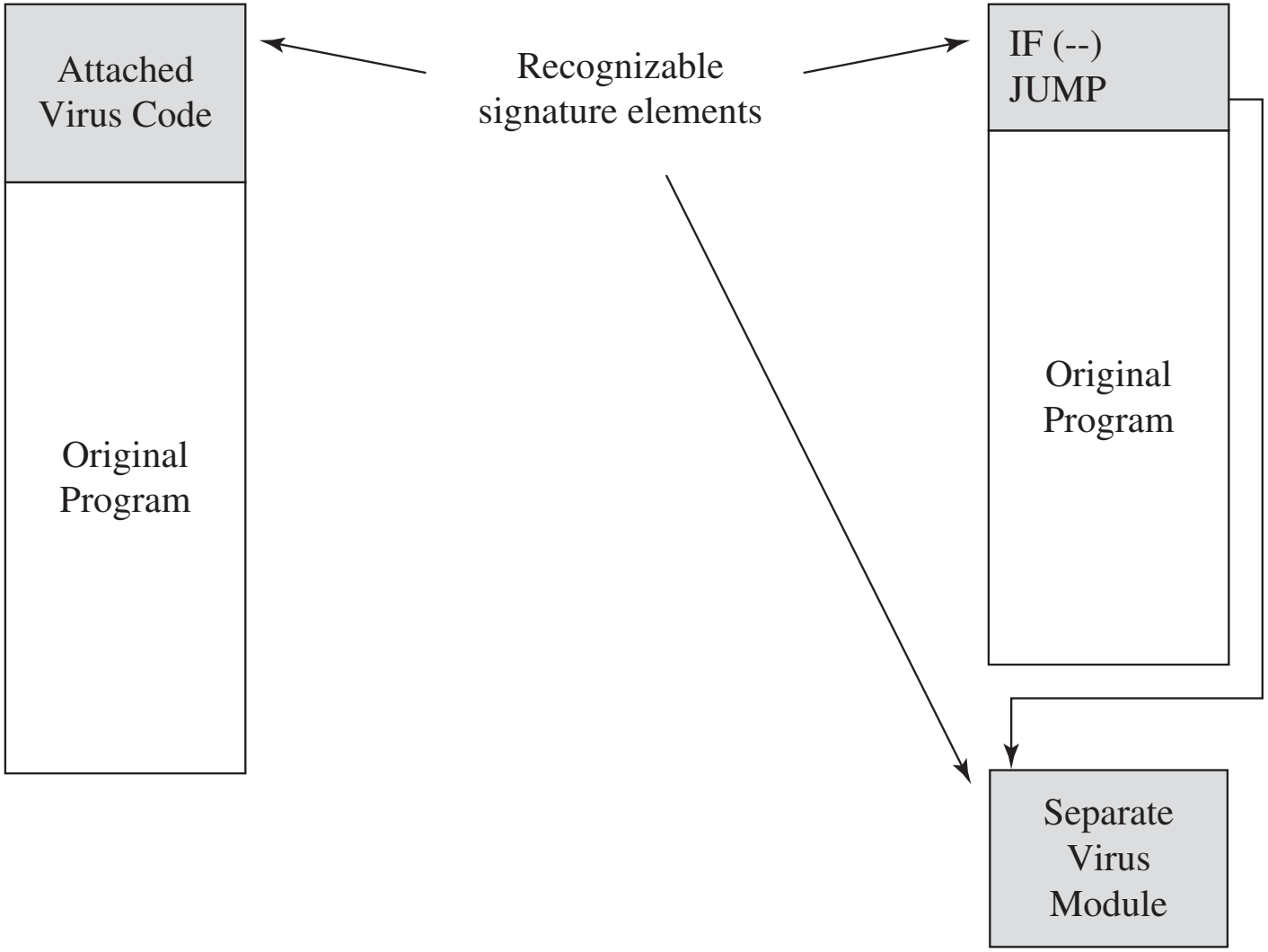
From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

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

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

Virus Signatures



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

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

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

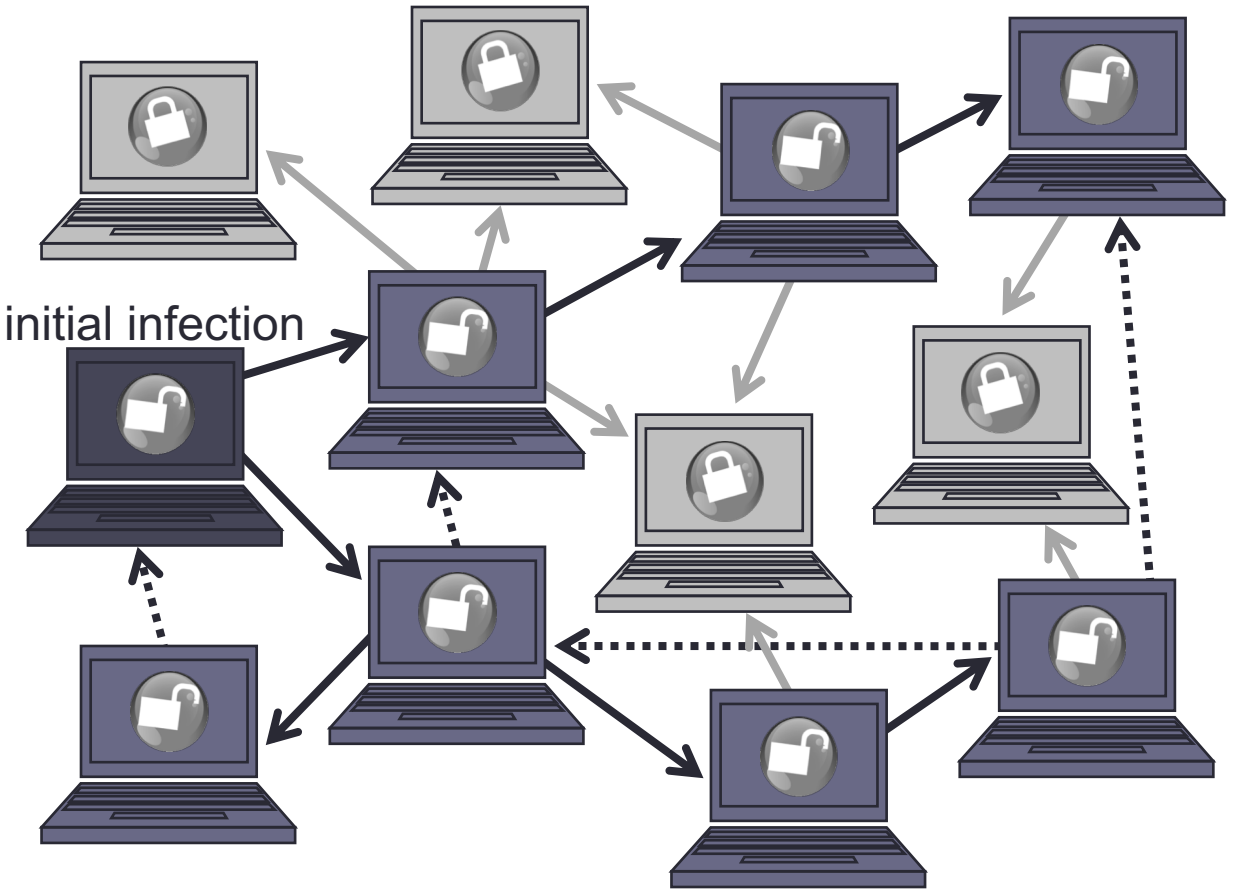
Early History (cont.)

- 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

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

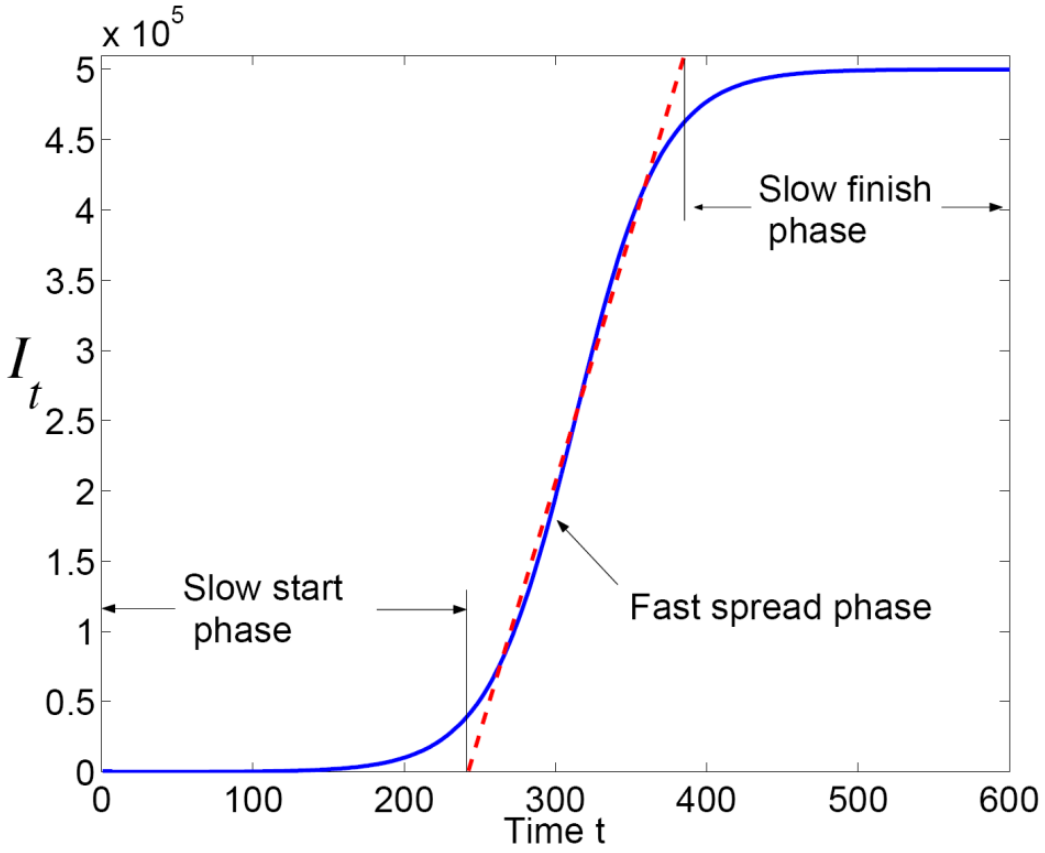
Worm Propagation

- 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.



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

Worm Propagation: Theory



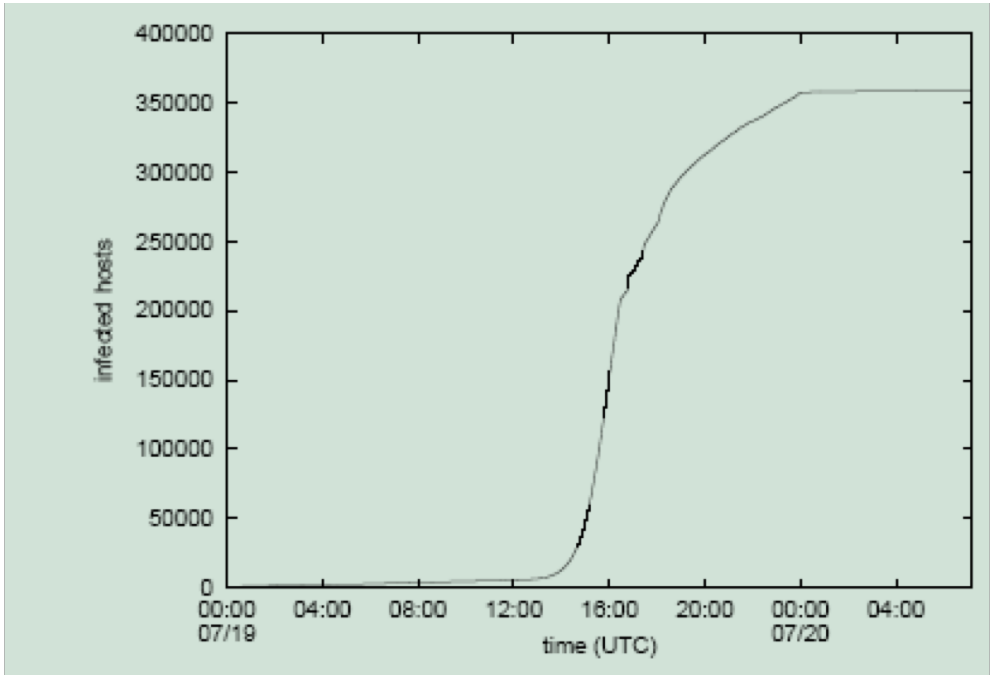
Source:
Cliff C. Zou, Weibo Gong, Don Towsley, and Lixin Gao. The Monitoring and Early Detection of Internet Worms, IEEE/ACM Transactions on Networking, 2005.

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

Propagation: Practice

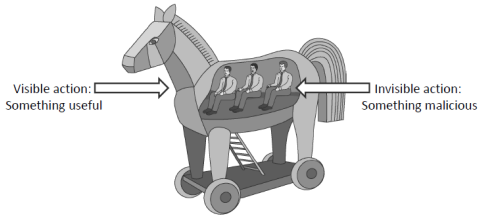
- Cumulative total of unique IP addresses infected by the first outbreak of Code-RedI v2 on July 19-20, 2001

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



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

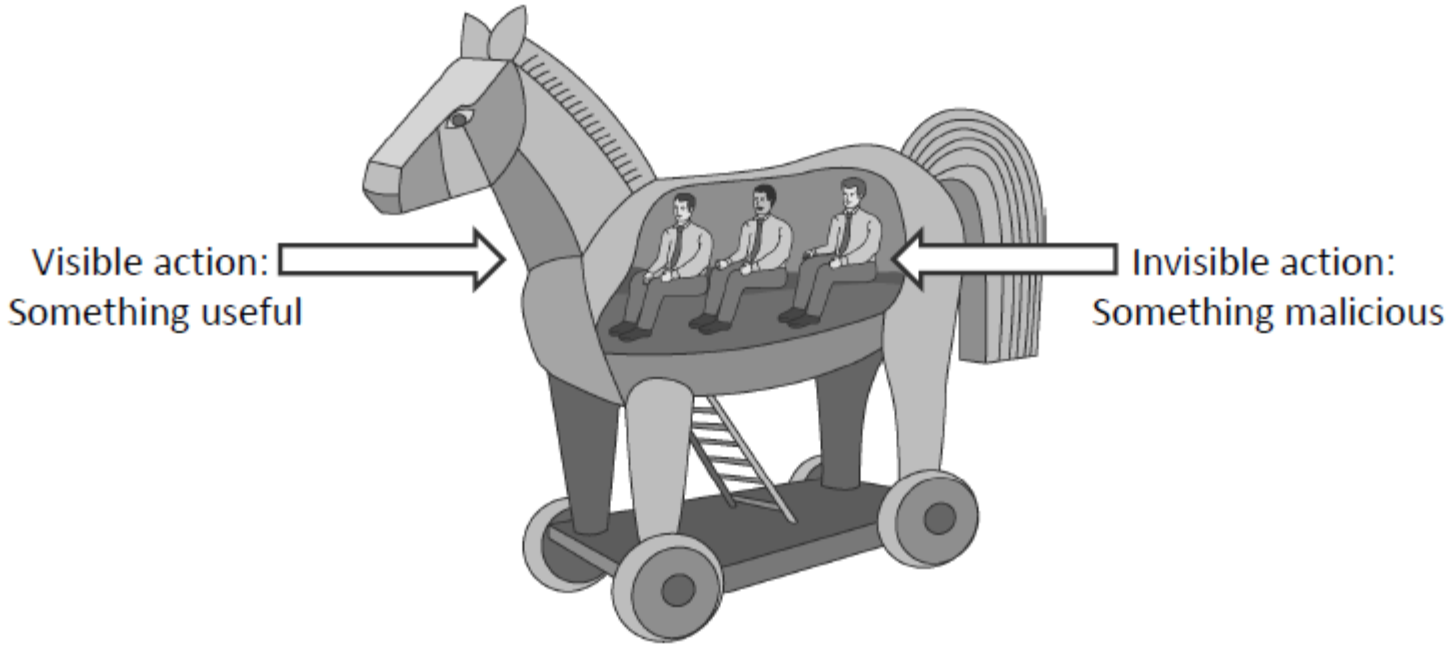
Trojan Horses



- 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.

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

Trojan Horses



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

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

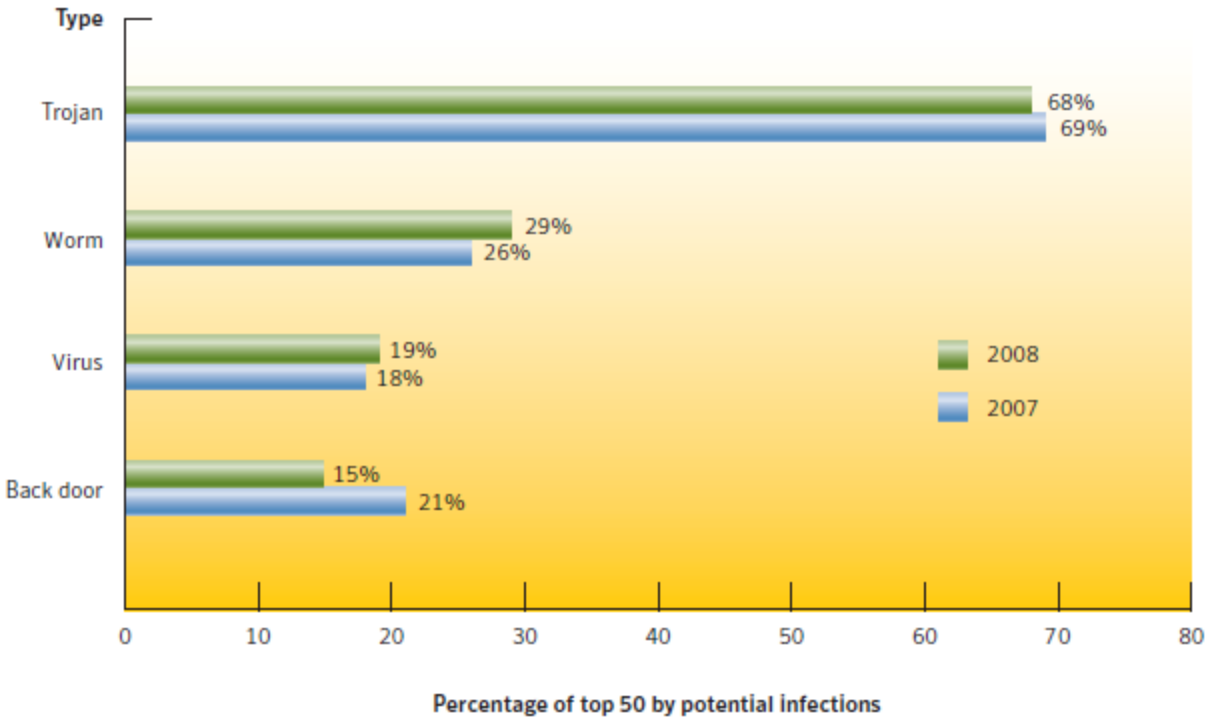
- Sometimes passed on using commonly passed executables, things like jokes forwarded by e-mail
- Sometimes marketed/distributed as “remote administration tool”
- Often combined with rootkits to disguise activity and remote access
- Popularized to an extent by software like Cult of the Dead Cow’s Back 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

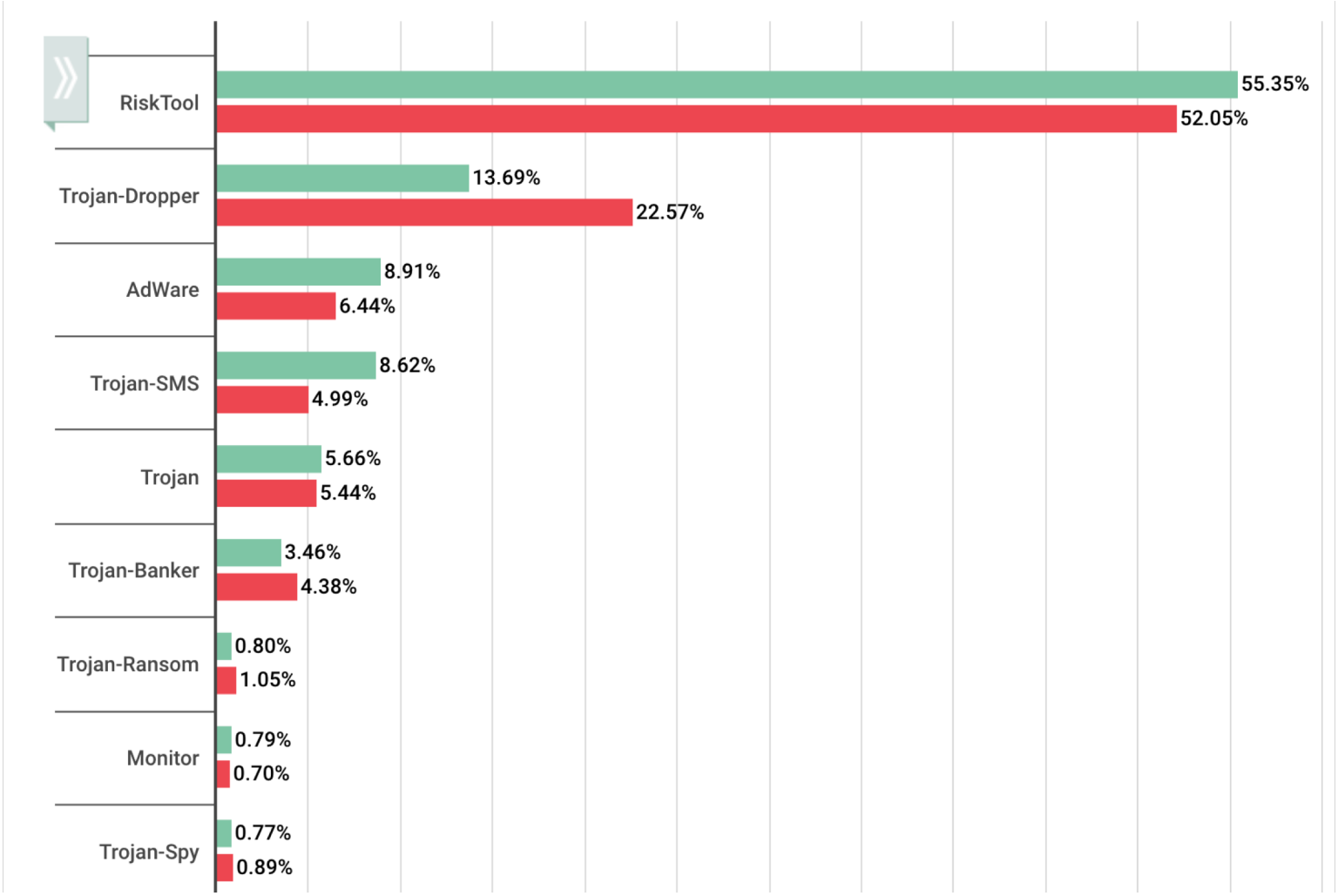
- Trojans currently have largest infection potential
  - Often exploit browser vulnerabilities
  - Typically used to download other malware in multi-stage attacks



Source: Symantec Internet Security Threat Report, April 2009

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

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/

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

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

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

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

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

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

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

Malware Activation

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

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

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

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

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

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

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

Questions?

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

Rise of the Hackers

- Rise of the Hackers

Questions?

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

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

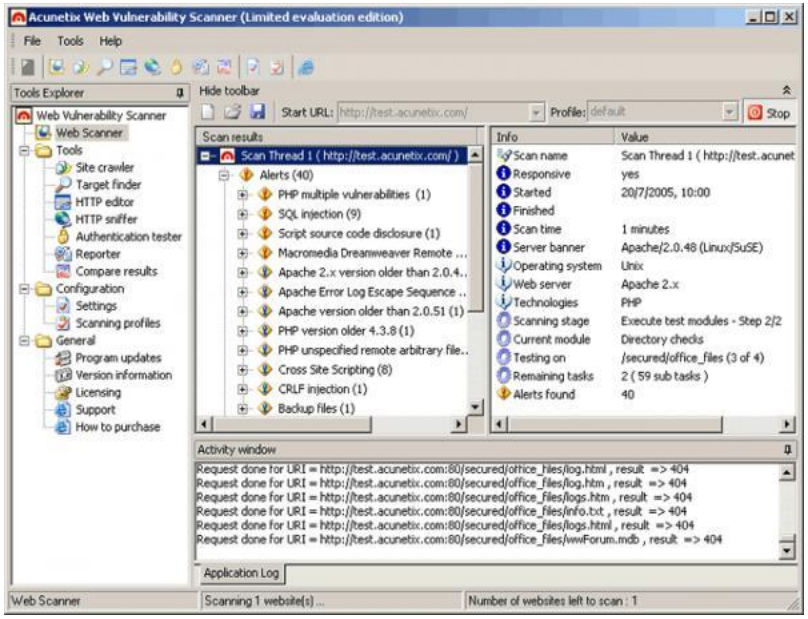
- A computer program designed to assess computers, computer systems, networks or applications for known weaknesses
  - probe your systems/networks for known flaws
- Typically used to detect vulnerabilities in software that runs on a network component
  - firewall, router, web server, application server,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

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

Security through Obscurity

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-more-crypto-primitives-and-obscurity-pocket-as-security/

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

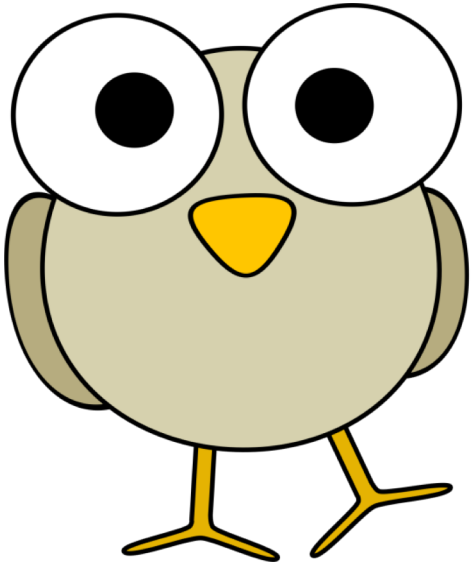
http://ithare.com/advocating-obscurity-pockets-as-a-complement-to-security-part-ii-deployment-scenarios-more-crypto-primitives-and-obscurity-pocket-as-security/

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

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

- 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

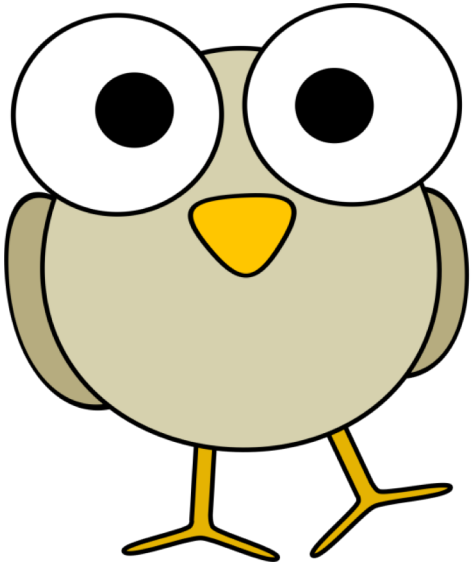
Which of the following are benefits of penetration testing?

- 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/

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

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

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

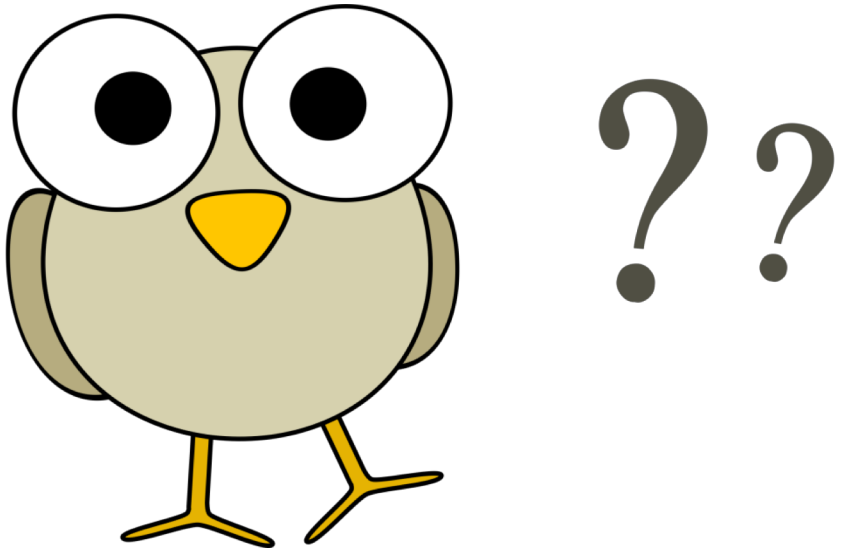
- This principle states that user interfaces should be **well designed and intuitive**
- All security-related settings should adhere to what an ordinary user might expect.

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

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

- 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

https://www.slideshare.net/LosaniWhat-is-the-internet-of-things-iot

IoT Devices

https://www.pentasecurity.com/blog/10-smartest-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

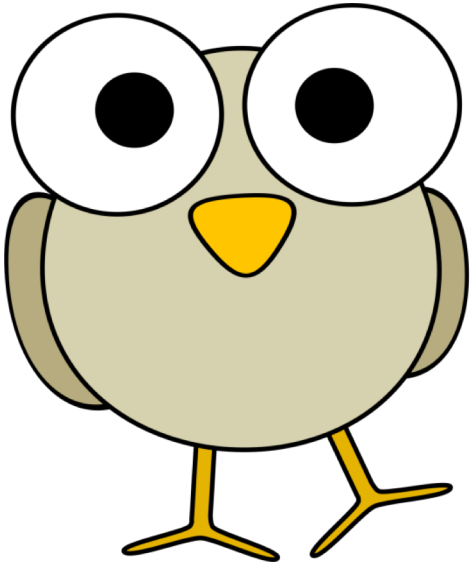
Risks in using IoT devices

- 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:

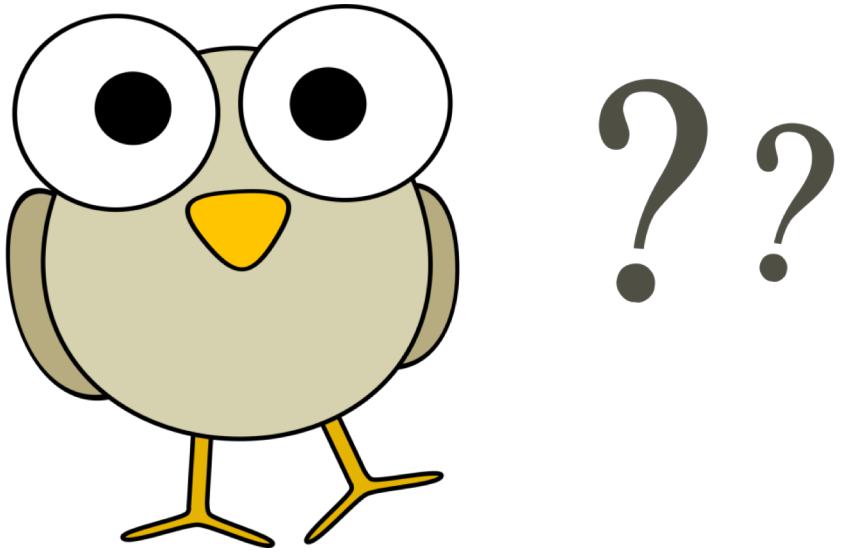
A. Only governments and universities owned computers.

B. Spamware caused some computers to break down

C. The number of personal computers greatly increased.

D. The hacktivists started using the internet

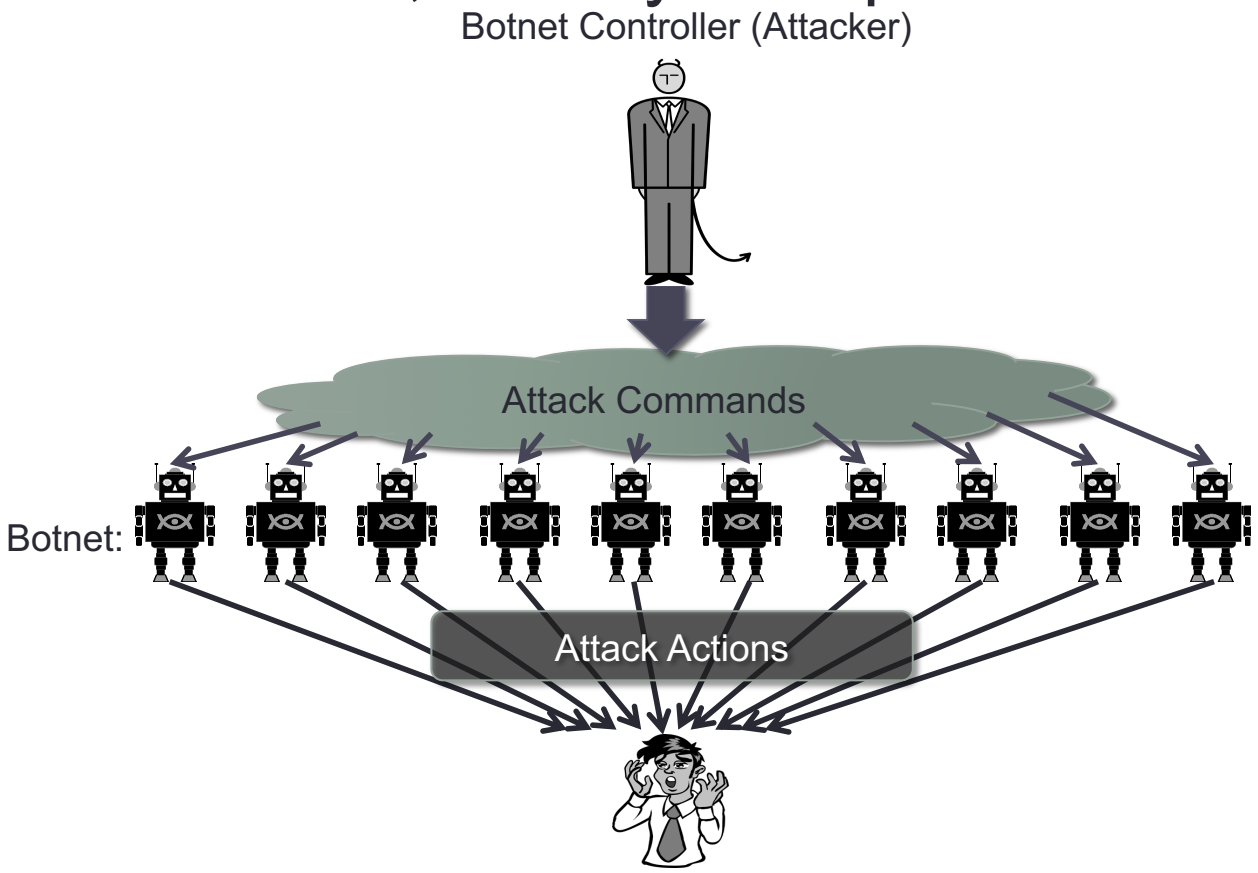
- Questions?



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/

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

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

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

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

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

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.

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

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.

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

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

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

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

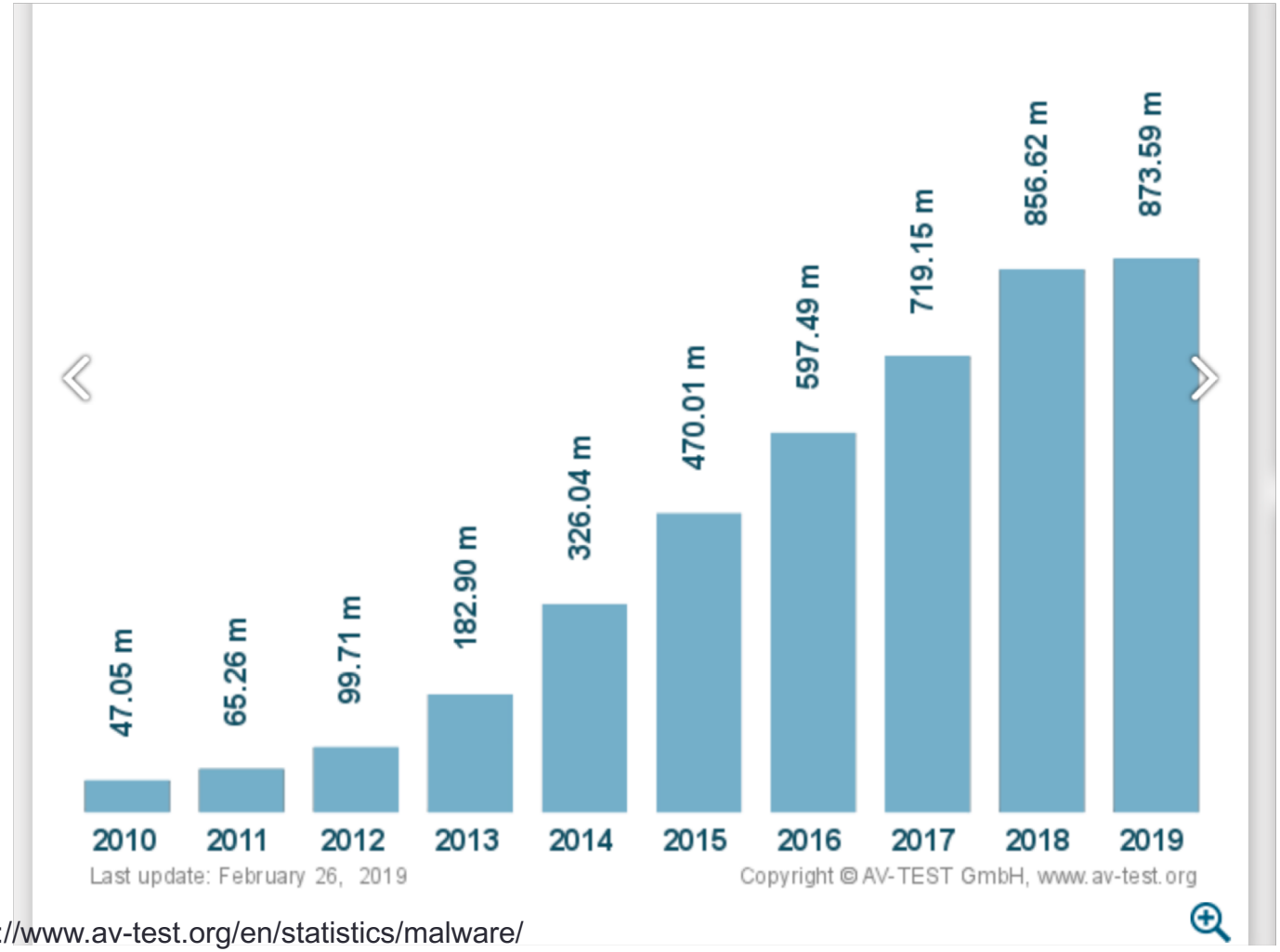
From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

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.

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

Malware Growth



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

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*

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

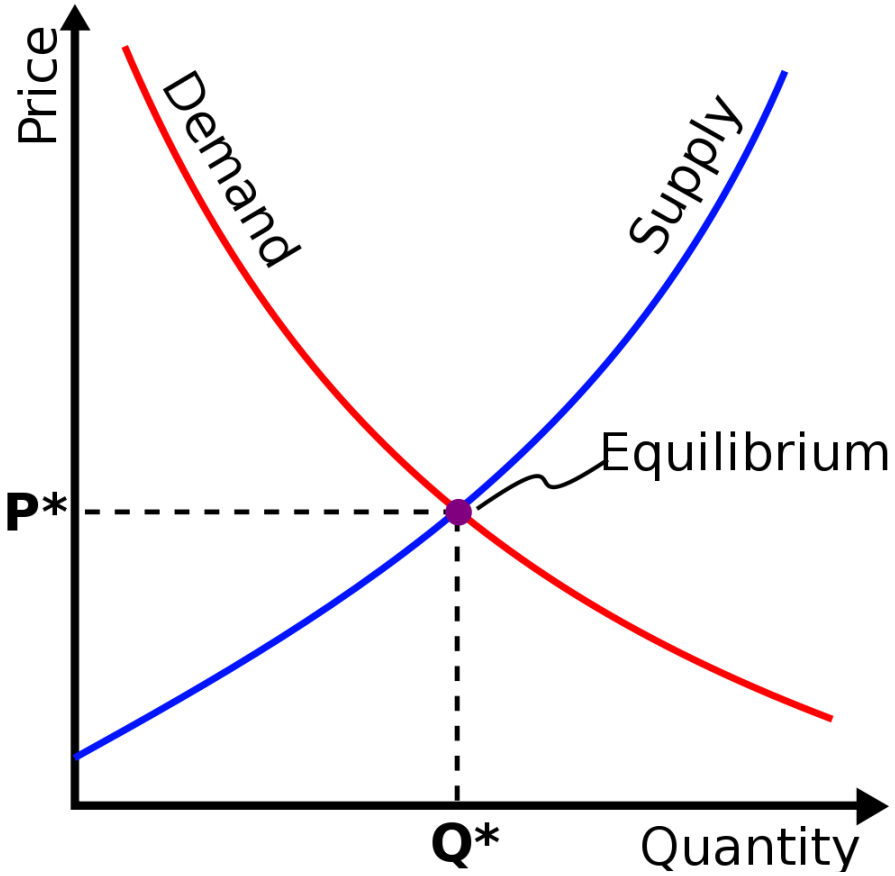
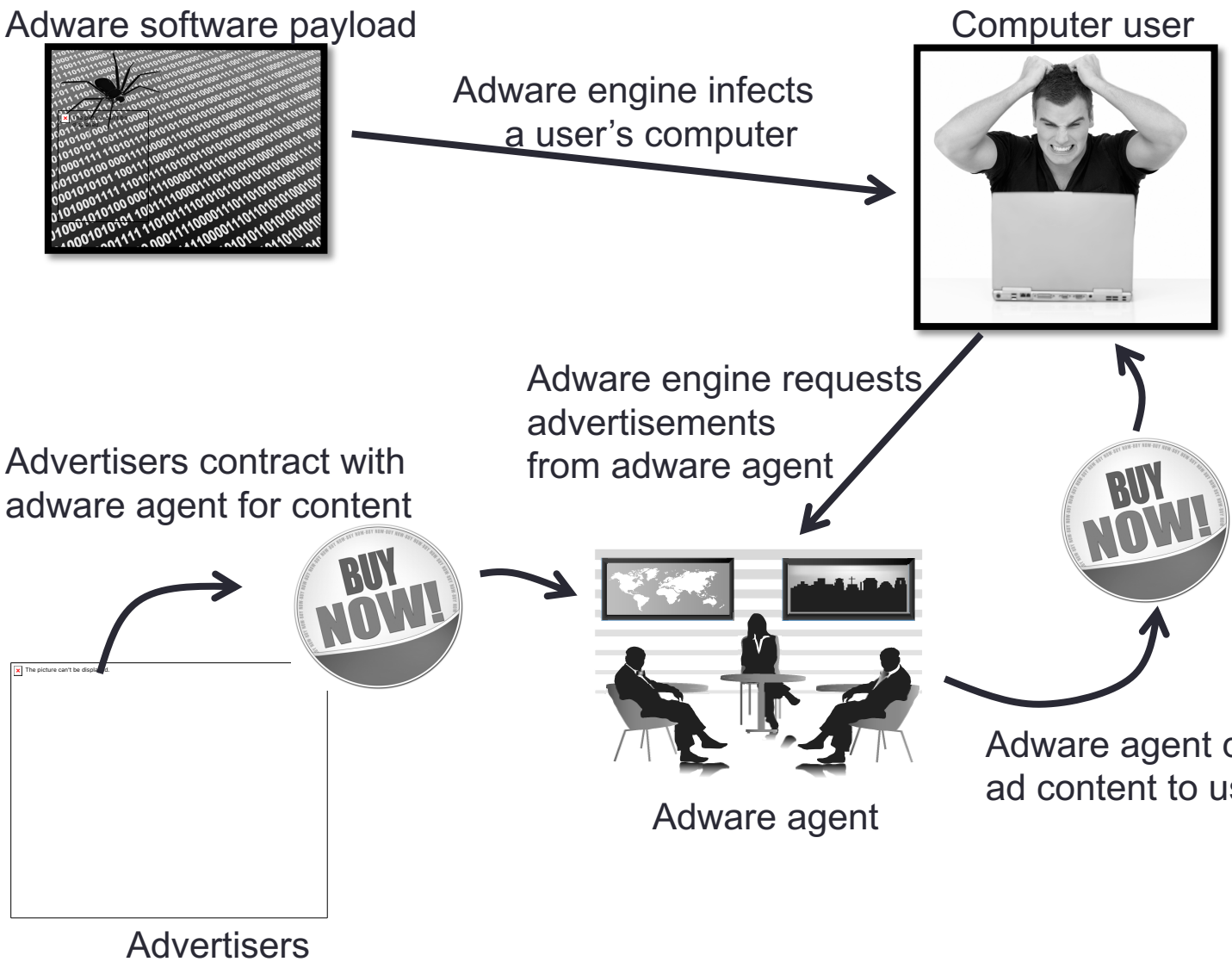


Image by User:SilverStar from http://commons.wikimedia.org/wiki/File:Supply-demand-equilibrium.svg used by permission under the *Creative Commons Attribution ShareAlike 3.0 License*

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

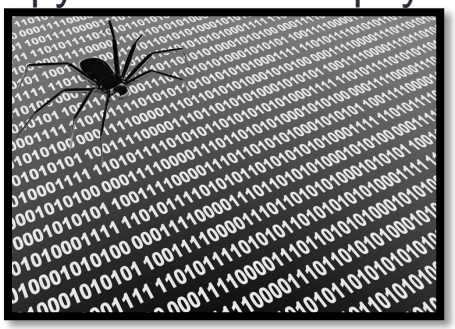
Adware



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

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

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

- Questions?