COMPUTER SECURITY

Software Security

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

Malware Payload

- 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

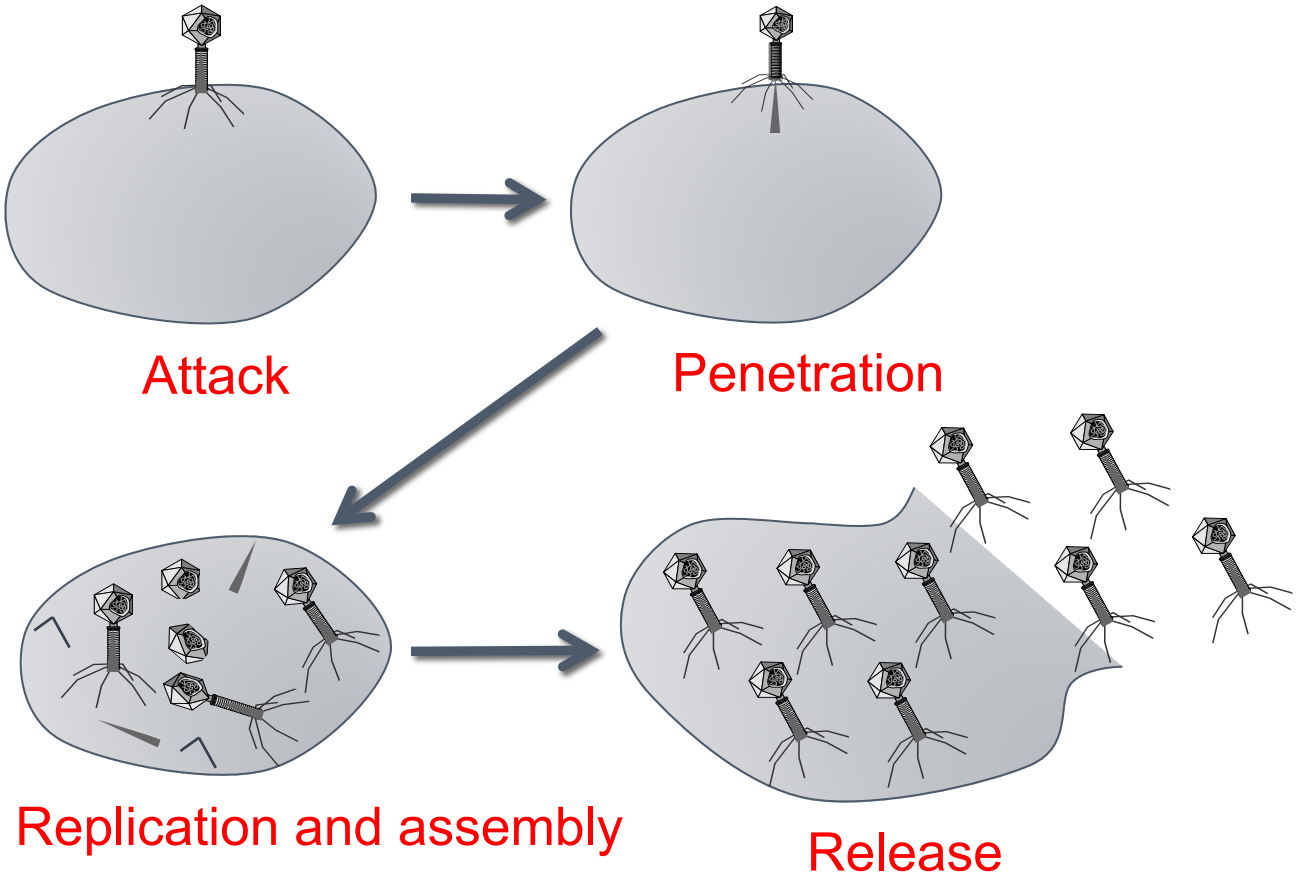
Computer Viruses

- 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.

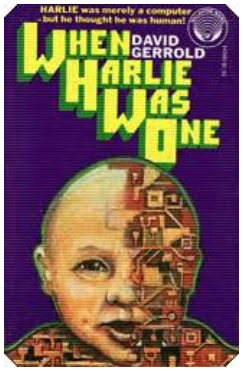
https://www.yelp.com/biz/40-computer-virus-removal-reseda

Biological Analogy

- Computer viruses share some properties with Biological viruses

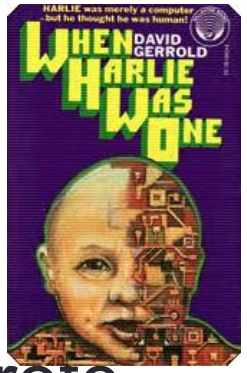


Early History



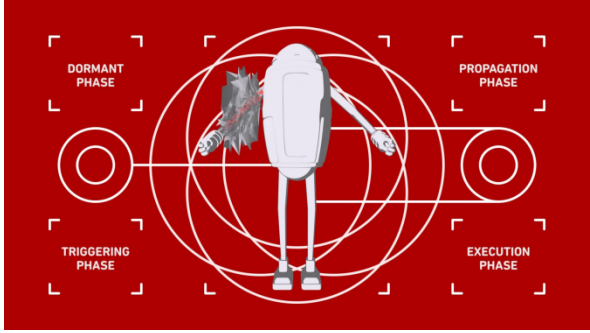
- 1972 sci-fi novel “When HARLIE Was One” features a program called VIRUS that reproduces itself
- First academic use of term ‘virus’ by PhD student Fred Cohen in 1984
  - credits advisor Len Adleman with coining it

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

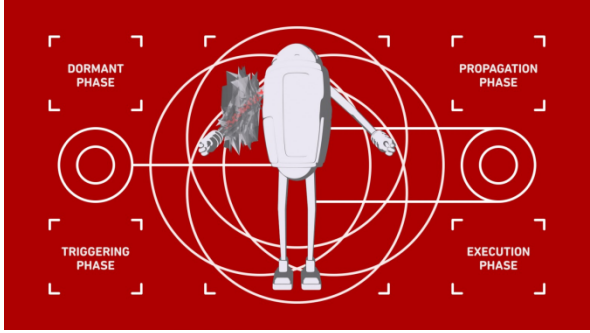
Virus Phases



- **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.

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

Virus Phases

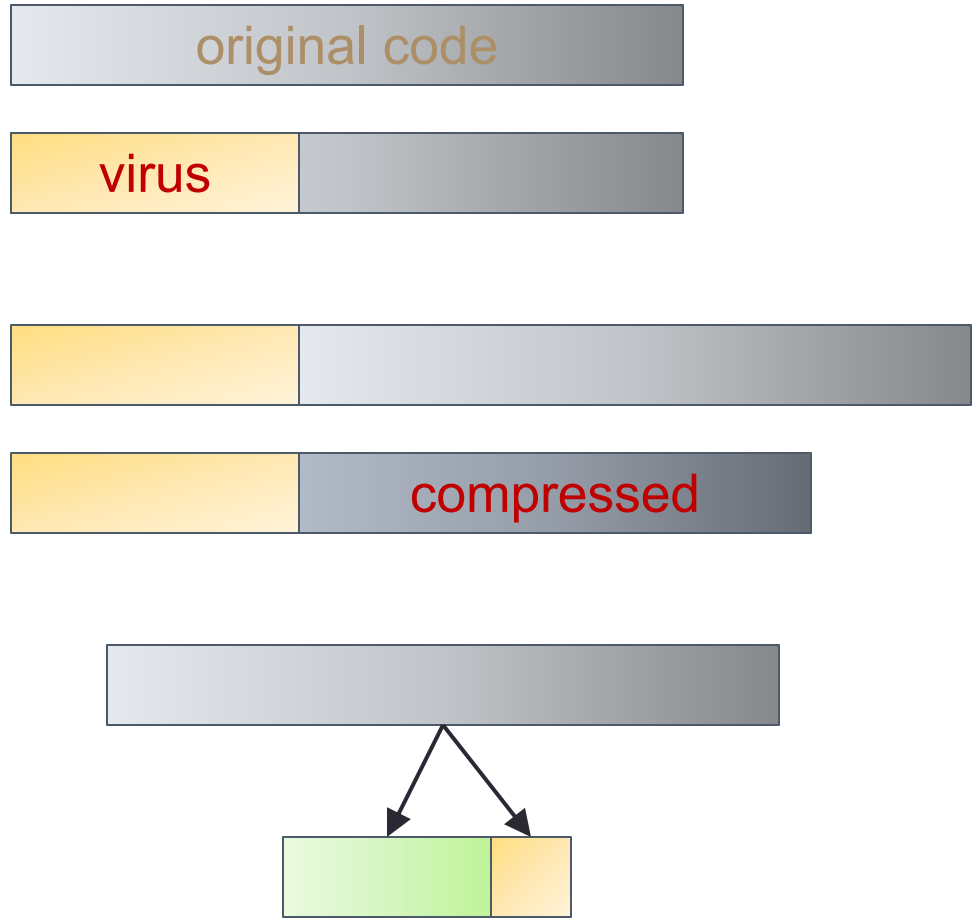


- **Triggering phase:**
  - Some logical condition causes the virus to move from a dormant or propagation phase to perform its intended action.
- **Action phase:**
  - The virus performs the malicious action that it was designed to perform, called **payload.**
    - This action could include something 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

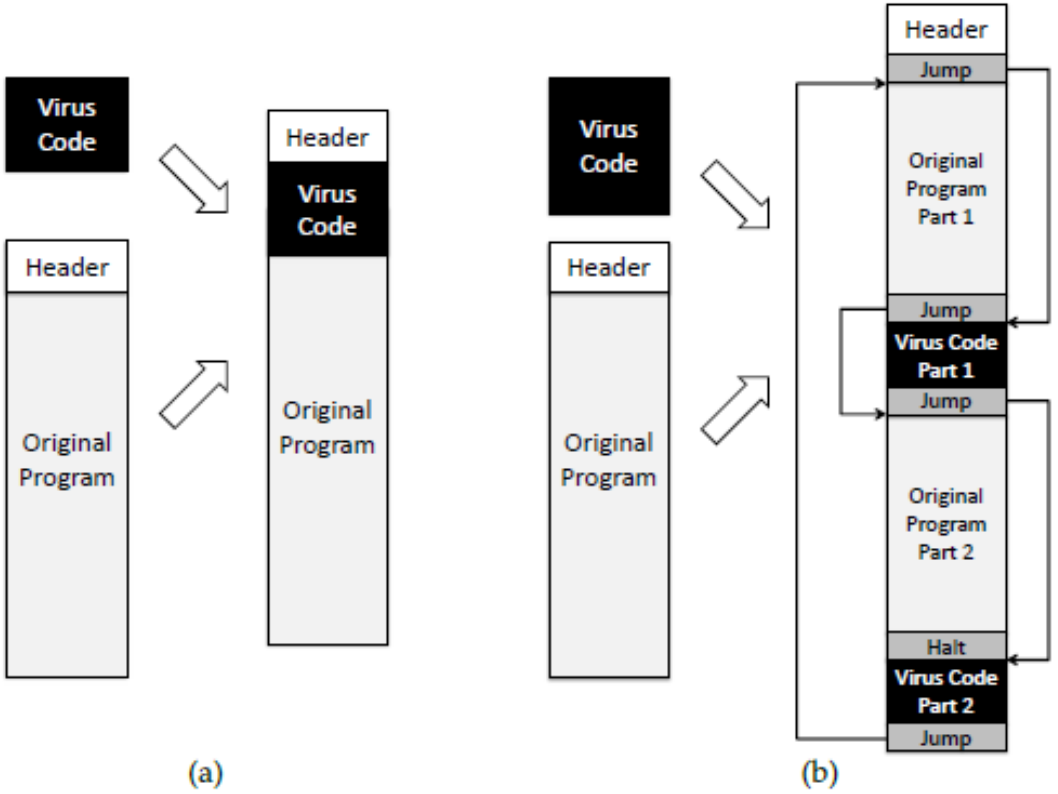
- Example: kernel32.dll is a collection of core Windows API calls (system calls)
  - imported by most applications
- Most viruses relying on patching DLLs usually attack kernel32.dll

Viruses and Windows DLL's

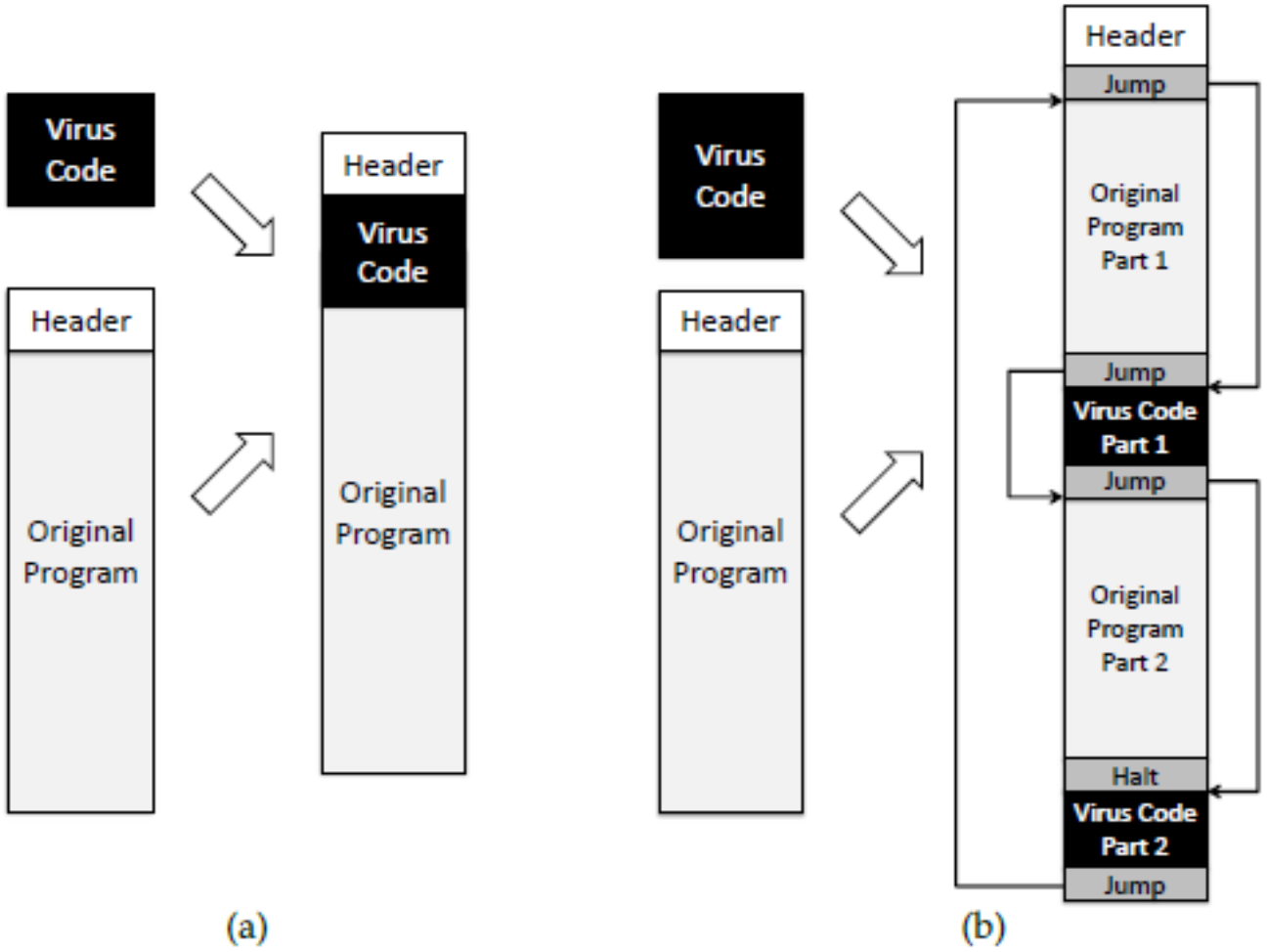
- Example: W32.Kriz will attack any PE executable, and kernel32.dll
  - to get a hook on system calls
- Hooking system calls may be done by legitimate programs
  - such as Regmon (a registry monitoring utility)
- Viruses hook onto DLLs by either:
  - changing their exported symbol table, so as to call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

Degrees of Complication

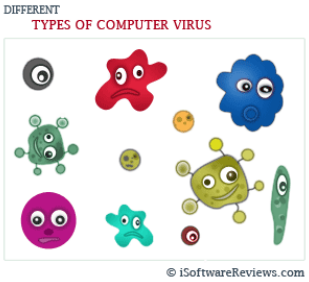
- Viruses have various degrees of complication in how they can insert themselves in computer code.



Degrees of Complication



Concealment

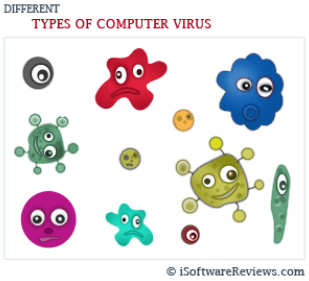


- Encrypted virus
  - Decryption engine + encrypted body
  - Randomly generate encryption key
  - Detection looks for decryption engine

- 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

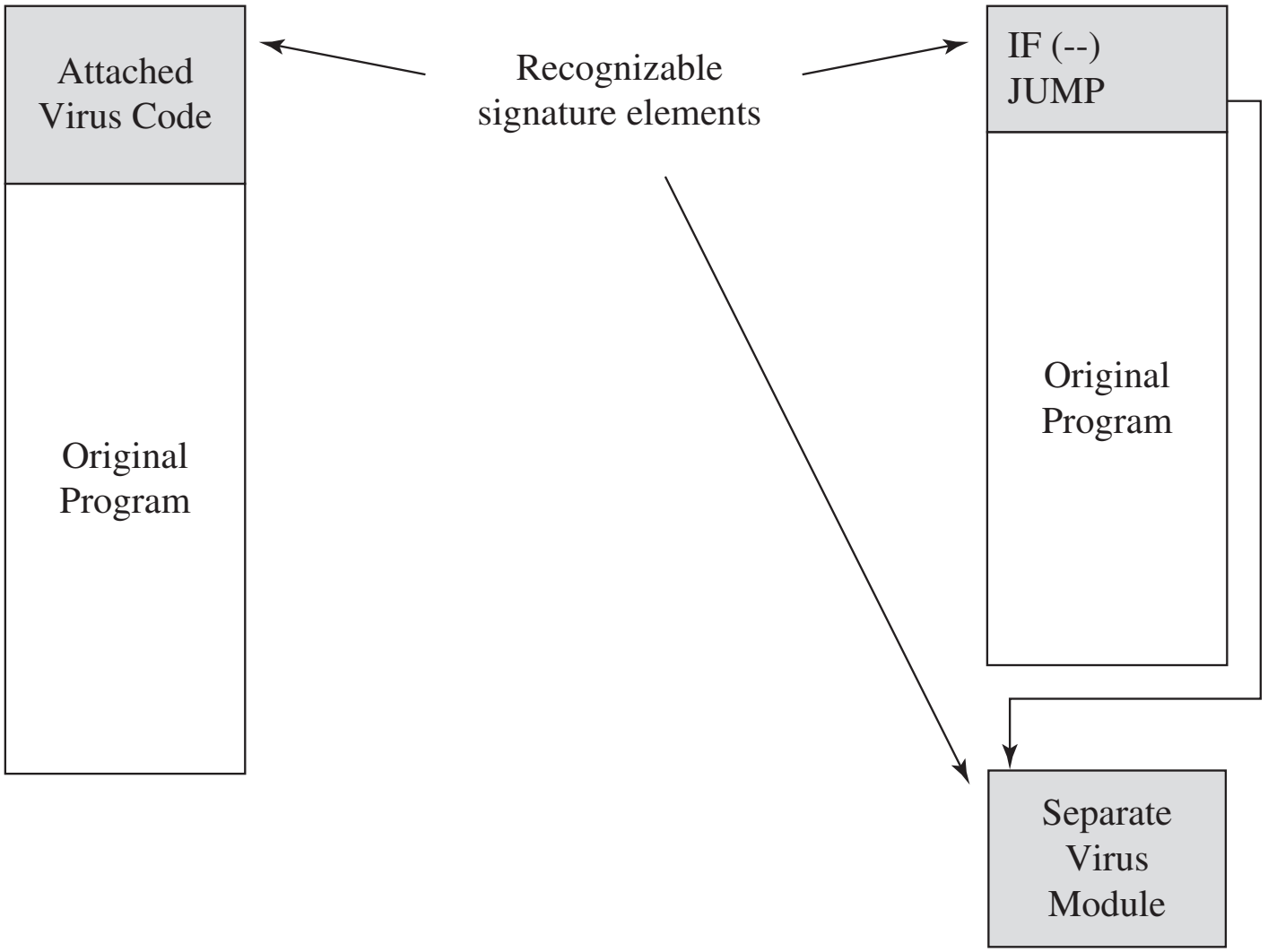
Virus Detection

- 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

Virus Detection

- 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 was the first worm to use e-mail service
  - released in December 1987
  - targeting IBM VM/CMS systems

Early History (cont.)

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

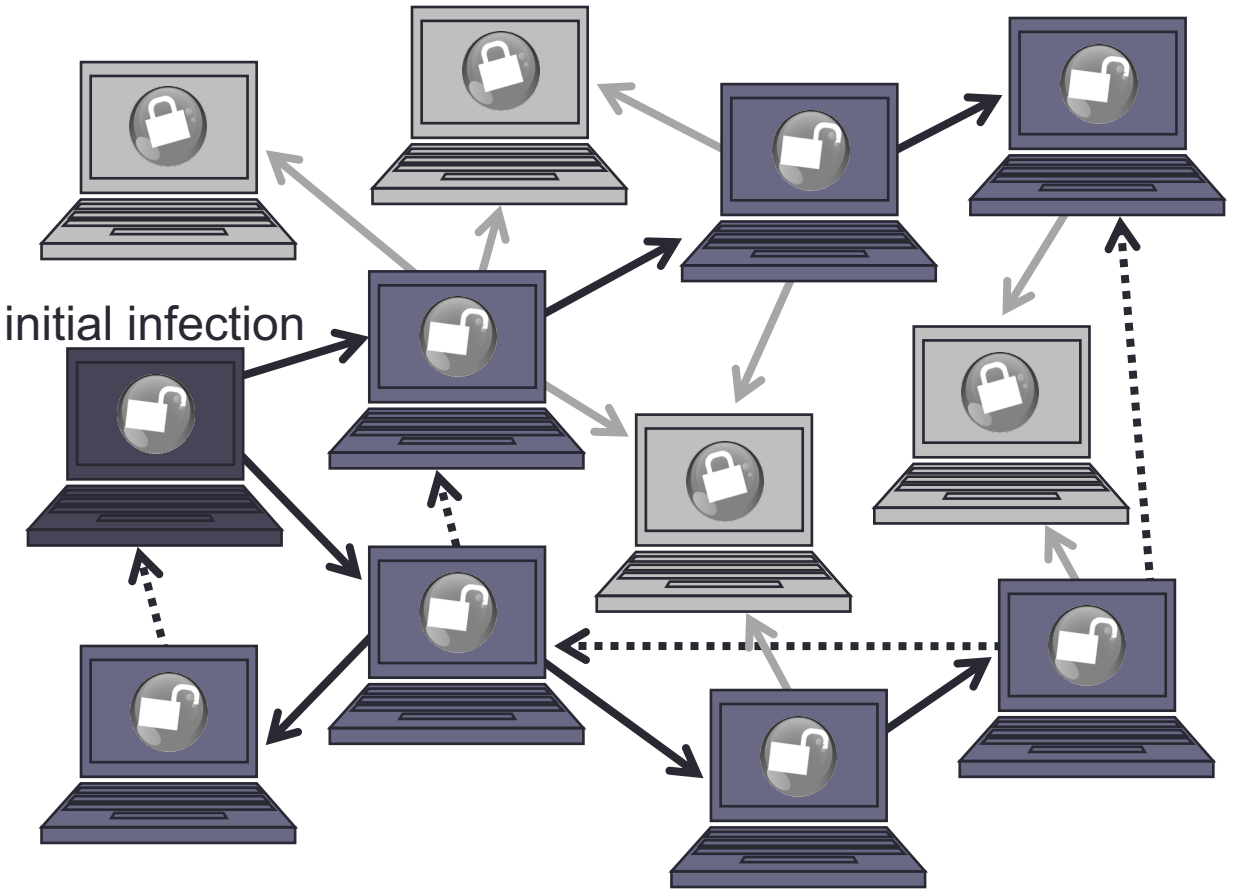
I love you Worm

- 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

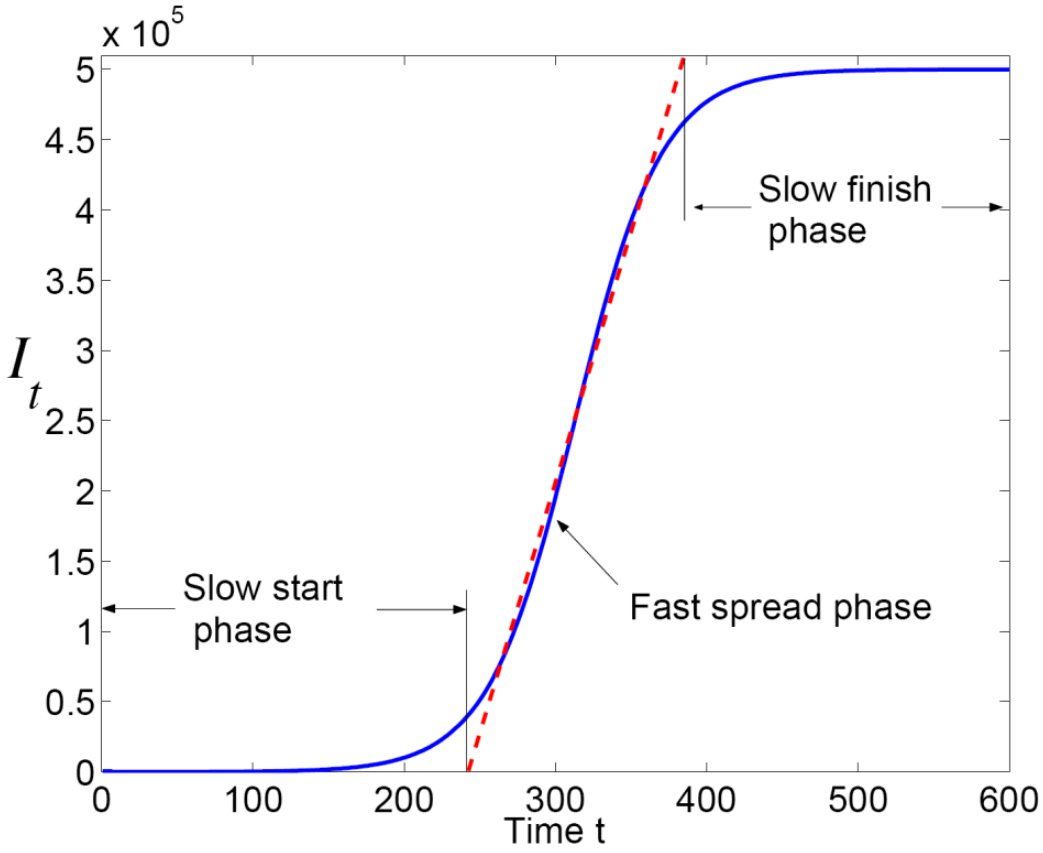
- I love you worm

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

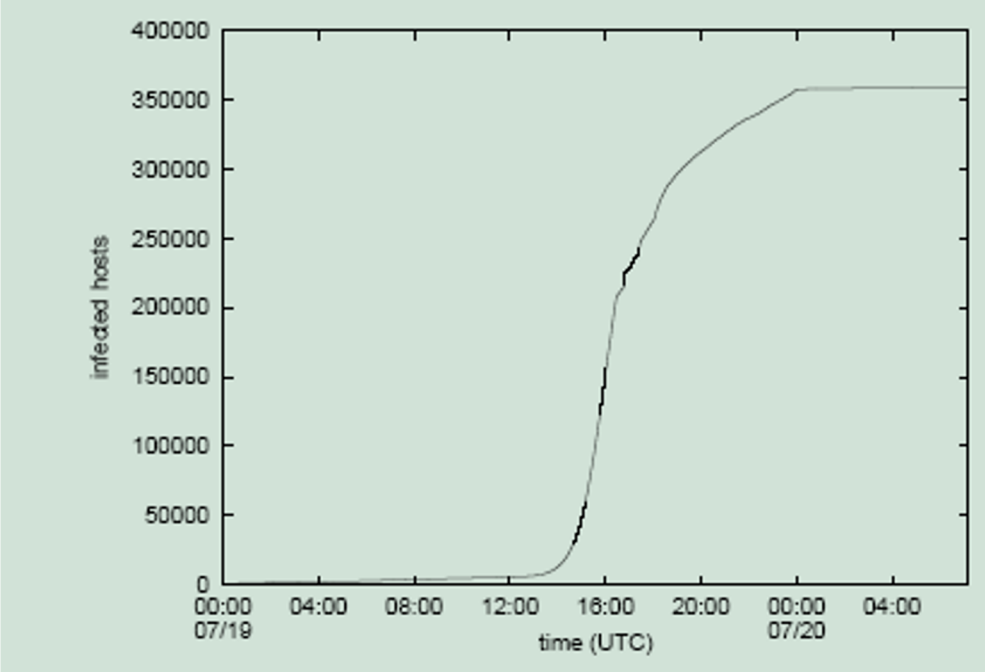


Source:
Cliff C. Zou, Weibo Gong, Don Towsley, and Lixin Gao. The Monitoring and Early Detection of Internet Worms, IEEE/ACM Transactions on Networking, 2005.

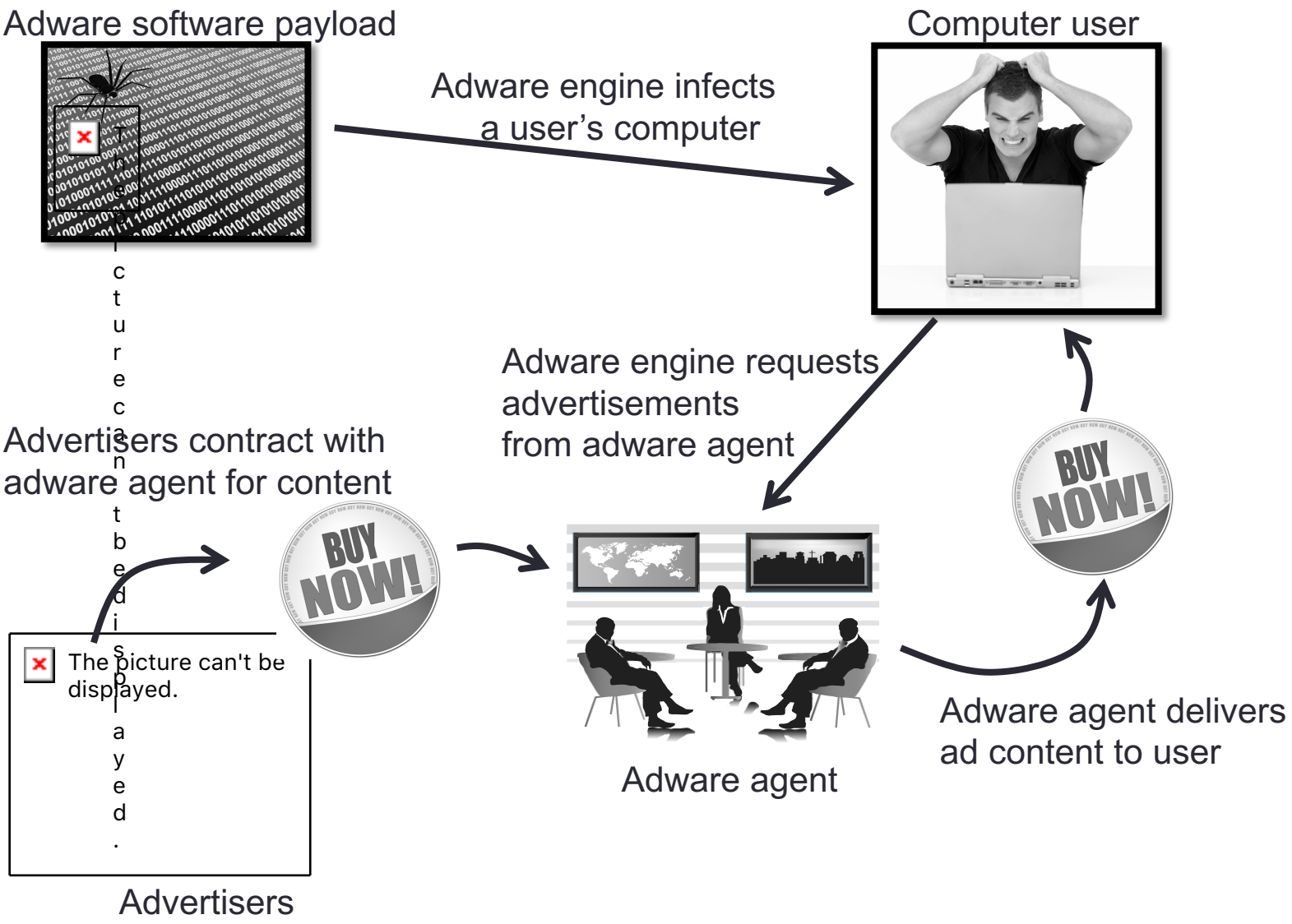
Propagation: Practice

- Cumulative total of unique IP addresses infected by the first outbreak of Code-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

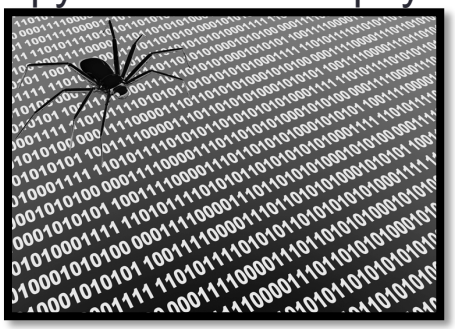


Adware



Spyware

Spyware software payload



1. Spyware engine infects a user's computer.

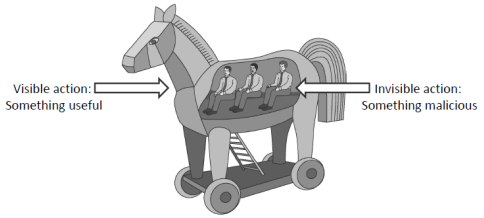
Computer user

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

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

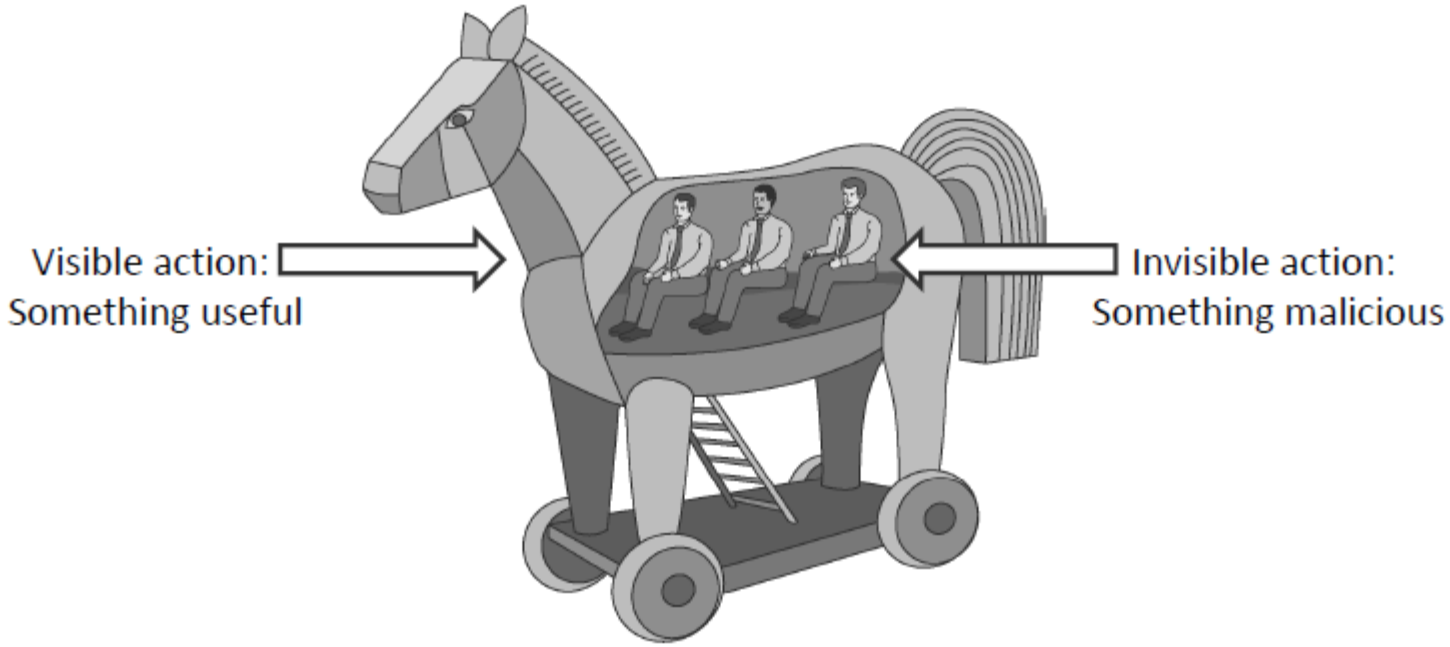
Spyware data collection agent

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
  - May allow taking control of systems
- Trojan horse programs do not replicate themselves

Trojan Horses

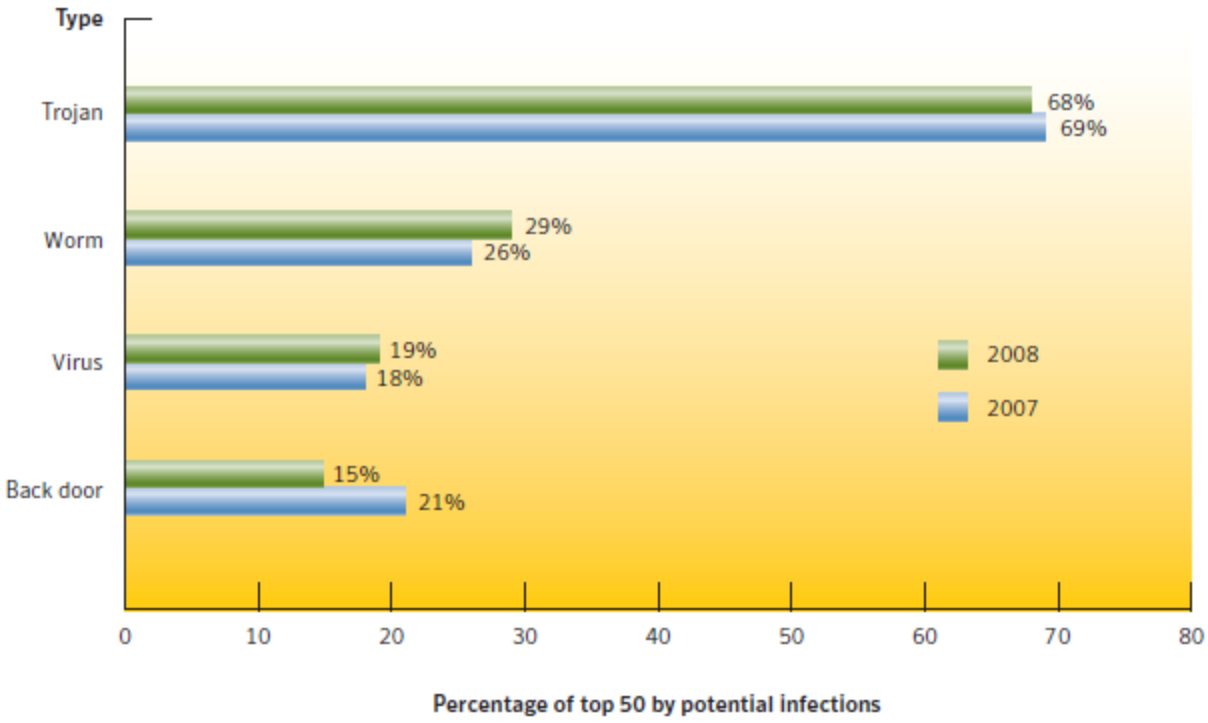
- Sometimes passed on using commonly passed executables, things like jokes forwarded by e-mail
- Sometimes marketed/distributed as “remote administration tool”
- Often combined with rootkits to disguise activity and remote access
- Popularized to an extent by software like Cult of the Dead Cow’s Back 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 symptom of being infected by a Trojan.
- Have lately begun using much of the same defense mechanisms used by viruses

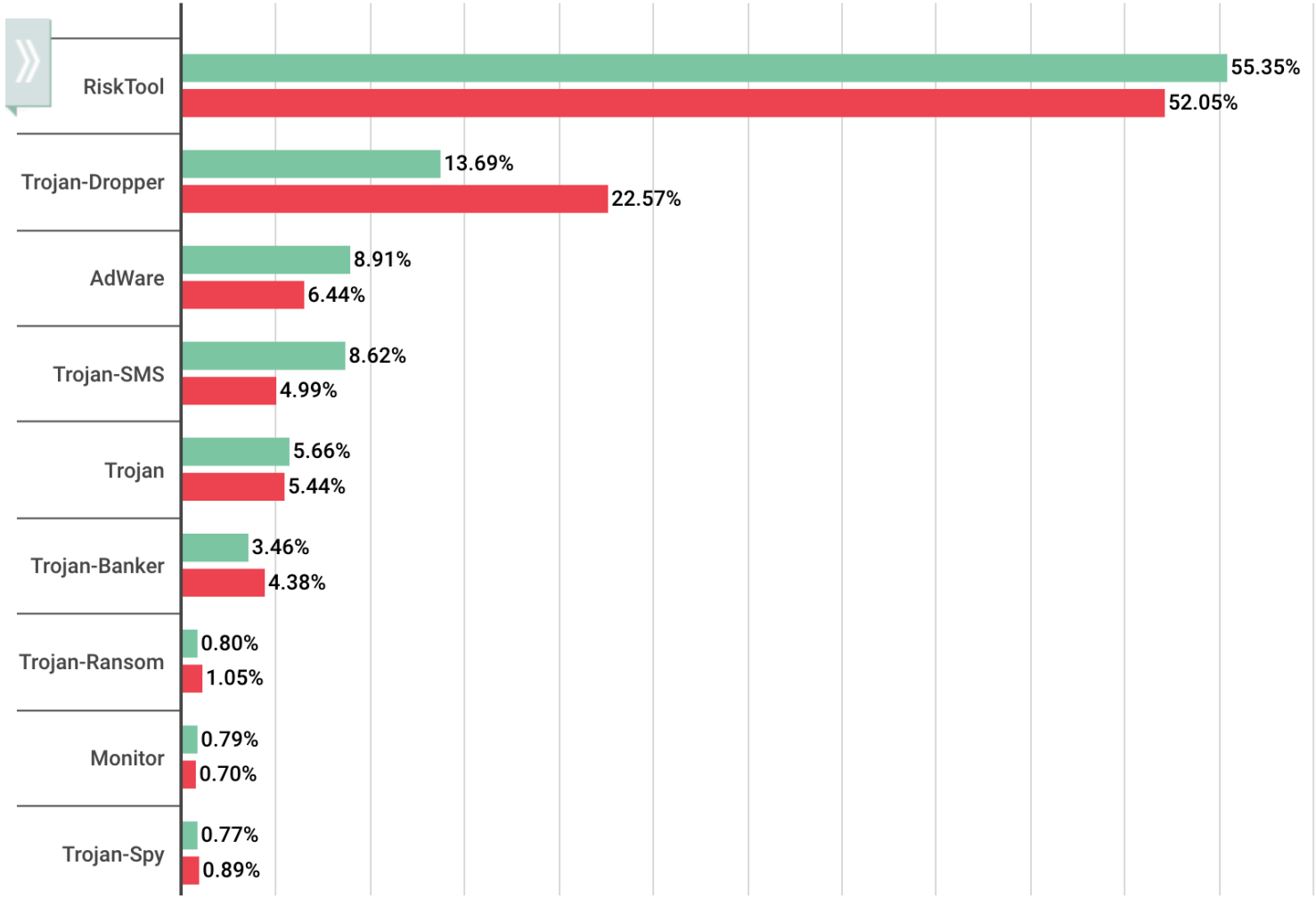
Current Trends

- 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

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

Types of Malware (cont.)

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

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
    - 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

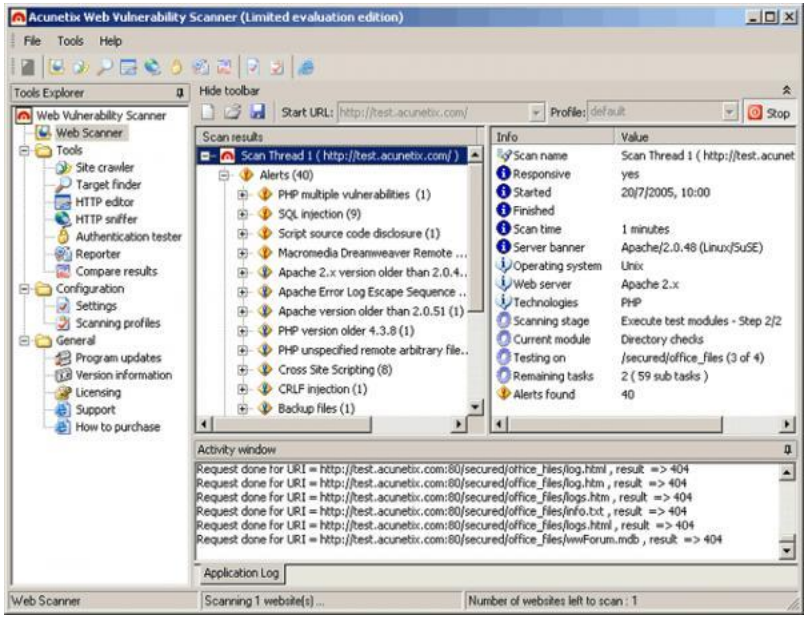
- A computer program designed to assess systems for known weaknesses
  - can be used on computers, computer systems, networks or applications
  - probe your systems/networks for known flaws
- Typically used to detect vulnerabilities in software that runs on a network component
  - firewall, router, web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

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

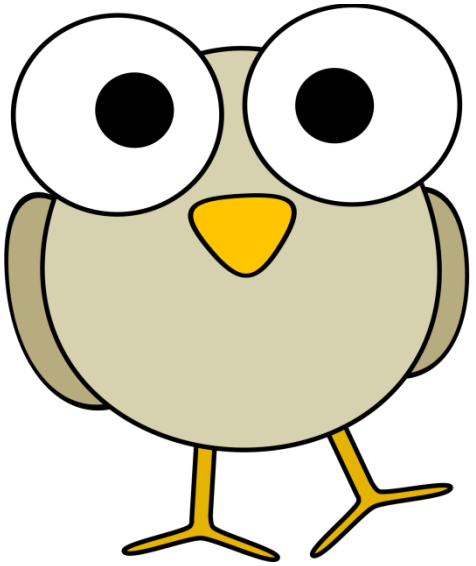
- Reliance on the secrecy of the design or implementation
  - as the main system security method
    - or component of a system
- Rejected by security experts!
  - obscurity should never be the only security mechanism!
  - has been historically used without success by several organizations

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 memory structure
  - in order to maliciously modify executing programs
    - code and data stored in the same memory
- 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?