Introduction to Program Security

Bo Luo
bluo@ku.edu

Introduction

- What is "secure" program?
  – Means different things to different people
- Is it secure if ?
  – takes too long to break through security controls
  – runs for a long time without failure
  – it conforms to specification
  – free from all faults

Faults in Programs

- which is better:
  - finding and fixing 20 faults in a module?
  - finding and fixing 100 faults?
- Finding 100 could mean
  - you have better testing methods

  OR

  - code is really bad; 100 were just the tip of the iceberg
- Software testing literature:
  - finding many errors early → probably find many more

Faults in Programs

- Fixing Faults: penetrate and patch
  - hire tiger team to try to break software
  - for each fault:
    - release a patch
  - bad idea since late 60s.
  - why bad?

Faults in Programs

- Penetrate and patch: why is this bad?
  - product was broken in the first place
  - developers can only fix problems that they know about
  - patches often only fix symptom. they're not cure
  - people don't bother applying the patches
  - patches can have holes
  - patches might cause bad side effect
  - patches tell the bad guys where the problems are
  - might affect program performance or limit functionality
  - more expensive than making it secure from the beginning

Complete Program Security

- Can we make programs completely secure?
  - Not easy
- Why? Software testing:
  - makes sure that code does what it's supposed to do
  - for security: must also verify that it **doesn't do anything it isn't supposed to do**. **much harder**
  - programming techniques often change more quickly than security techniques

Program Security

- IEEE Terminology
  - *error* – human action that causes an incorrect result
  - *fault* – incorrect step, process or data definition in a program
  - *failure* – system doesn't behave according to requirements

  - a fault is an *inside view* - seen by developers
  - a failure is an *outside view* - seen by users

Program Security

- Types of flaws
  - validation error
  - domain error
  - serialization and aliasing
  - inadequate authentication
  - boundary condition violation
  - other exploitable logic errors

- from Landwehr: *Taxonomy of Security Flaws*

Validation Errors

- Not checking validity of
  - function arguments
  - function return values
- Examples:
  - type of variable
  - length of a buffer
  - permissions of a file
  - other variable properties
  - A DNS crash story: “,” in a domain name
- Should validation include checking user input?

Validation Errors

- “King” of validation errors
  - Order at Burger King costs $4.33
  - Cashier enters '4' '3' '3‘; does something else; enters '4' '3' '3' again
  - Result: bill is $4334.33
  - Customer paid with debit card
  - Refund didn't clear for several days

Validation Errors

- "Fat Finger Syndrome"
  - Japanese bank trader sale of a telecom stock
  - intention to sell:
  - 1 share at 600,000 yen
  - actually sold:
  - 600,000 shares at 1 yen
  - cost company about $256 million

- several other similar examples

Validation Errors

- Quantas Airplane 10/7/2008
  - "Spike" of bad data sent to flight computer
  - Sent plane into nose dive

Domain Errors

- "holes in the fences"
  - insufficient protection of boundaries
  - example: ability to read another user's files

Bo Luo

Serialization, Aliasing

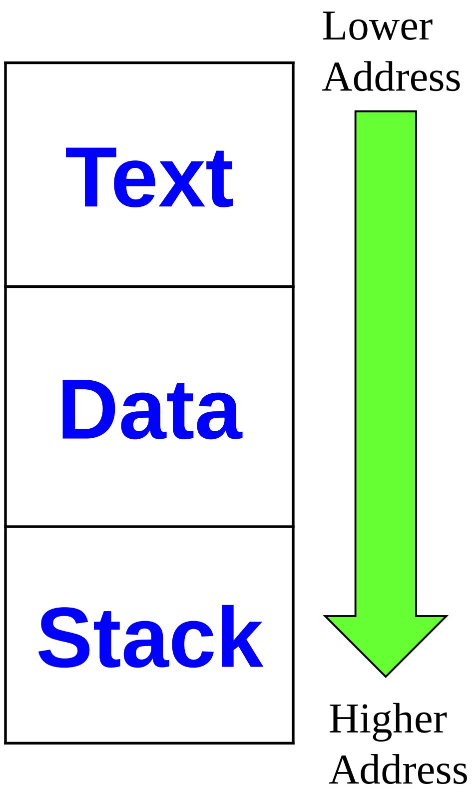
- serialization
  - vulnerability offered by asynchronous system behavior
  - example: TOCTTOU flaws
- Aliasing
  - when two or more objects may have the same name

Non-malicious Program Errors

- Buffer Overflow
  - Simple problem
  - Known about for decades
  - Still very common!
  - Account for 50% of all major CERT/CC in 1999
    - The CERT Coordination Center (CERT/CC): Coordination Center of Computer Emergency Response Team (CERT)
    - Created in response to the Morris worm (interesting story, we will talk about it later)
    - CERT/CC publishes security alerts

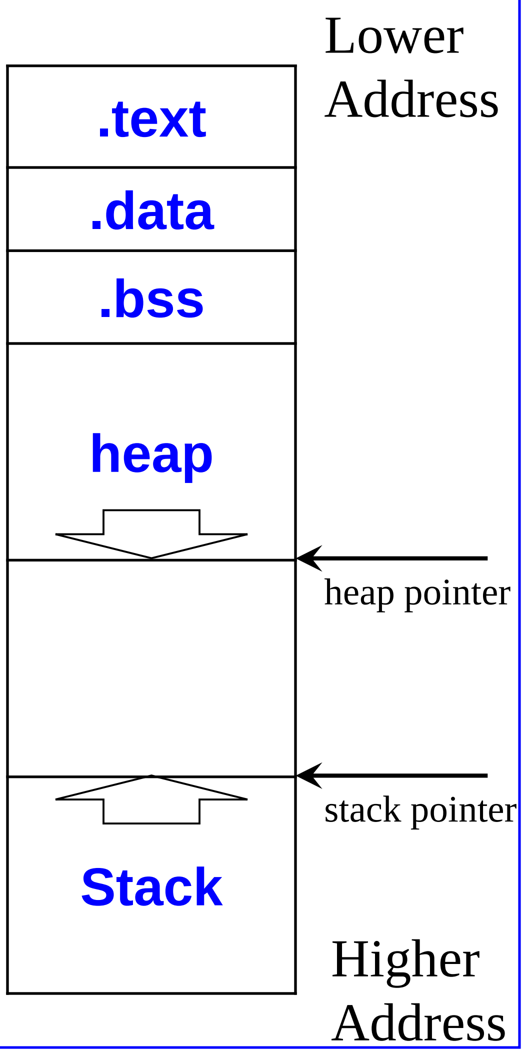
Buffer Overflow

- Memory organization
  - Process's memory
  - Text: code segment
    - program instructions
    - Read only
    - segmentation fault if you try to write to it
  - Data segment
    - Initialized data: global and static variables
    - Uninitialized data: BSS
    - Heap



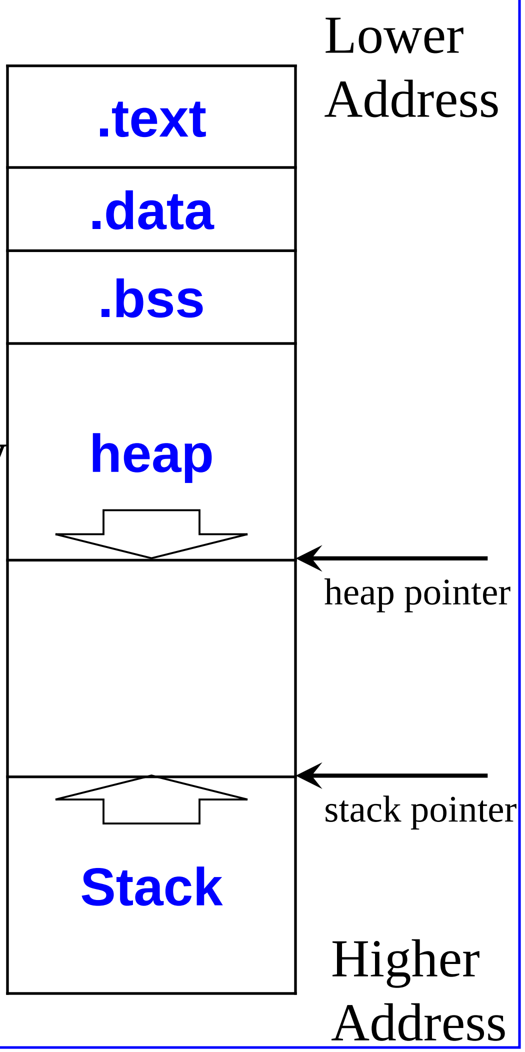
Buffer Overflow

- Memory organization
  - Process's memory
  - Text: code segment
    - program instructions
    - Read only
    - segmentation fault if you try to write to it
  - Data segment
    - Initialized data: global and static variables
    - Uninitialized data: BSS
    - Heap



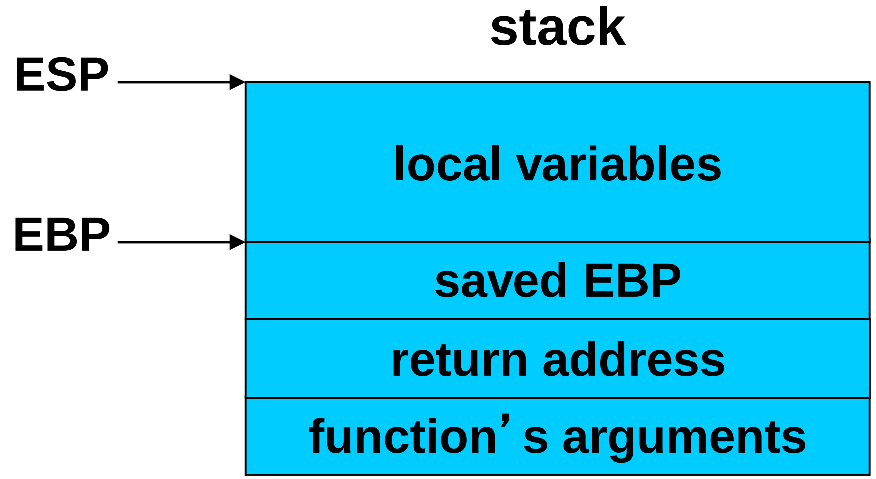
Buffer Overflow

- Memory organization
  - Process's memory
  - Stack
    - Activation records (stack frames) for sub-programs
    - Stack variables
  - contiguous block of memory
    - top of the stack: pointed to by the stack pointer (SP)
    - bottom of the stack: fixed address
  - CPU instructions to PUSH and POP



Buffer Overflow

- Look into the stack
  - The typical activation record for a function
    - Arguments
    - Return address
    - Old EBP (Extended Base Pointer): caller's EBP
    - Local variables



Buffer Overflow

- Function call example:

```
void func(int a, int b, int c){
    char buf[10];
    char cuf[20];
}


void main(){
    func(10,20,30);
}
```

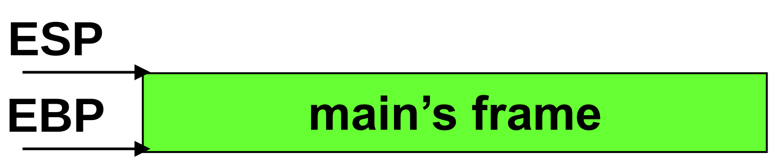
- push $30
- push $20
- push $10
- push return addr
- push (old) base ptr
- old stack ptr becomes new base ptr.
- push buf
- push cuf

Buffer Overflow

- Function call example:

- At main()

```
void func(int a, int b, int c){
    char buf[10];
    char cuf[20];
}


void main(){
    func(10,20,30);
}
```

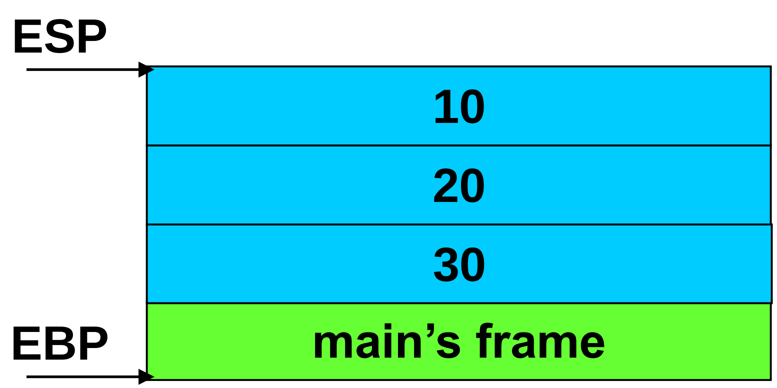


Buffer Overflow

- Function call example:

```
void func(int a, int b, int c){
     char buf[10];
     char cuf[20];
}


void main(){
     func(10,20,30);
}
```

- push $30
- push $20
- push $10

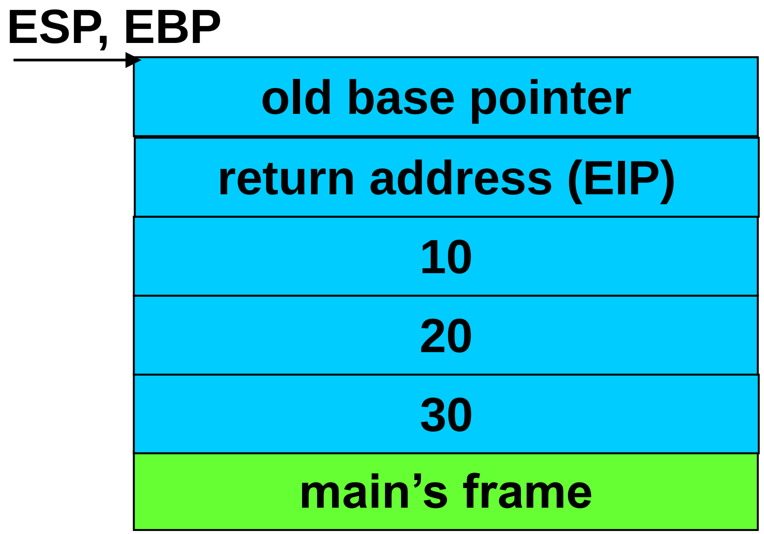


Buffer Overflow

- Function call example:

```
void func(int a, int b, int c){
     char buf[10];
     char cuf[20];
}


void main(){
     func(10,20,30);
}
```

- push return addr
- push (old) base ptr
- old stack ptr becomes new base ptr.

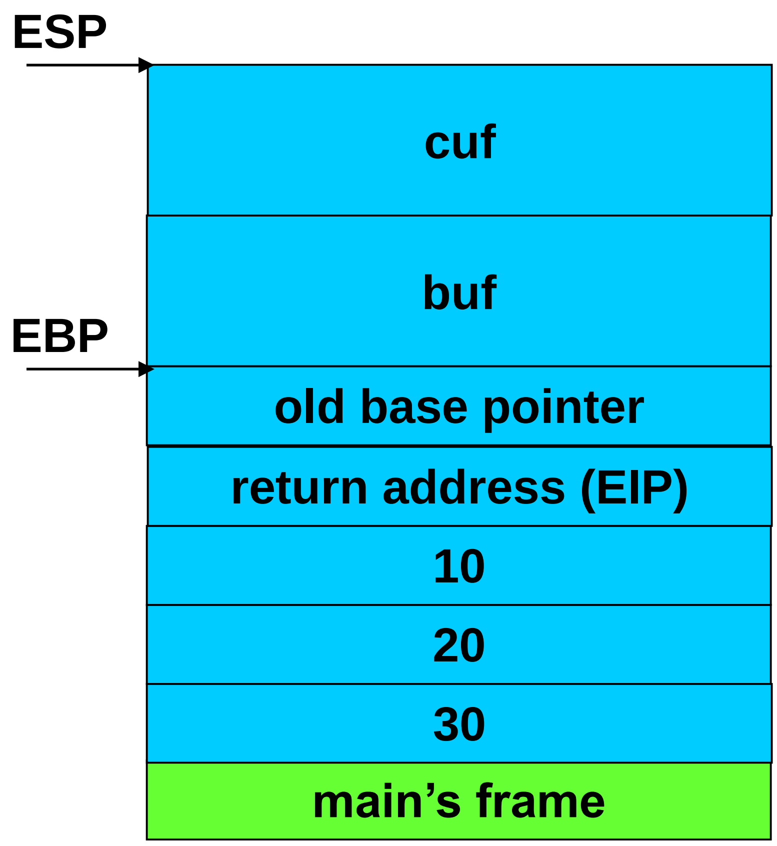


Buffer Overflow

- Function call example:

- push buf
- push cuf

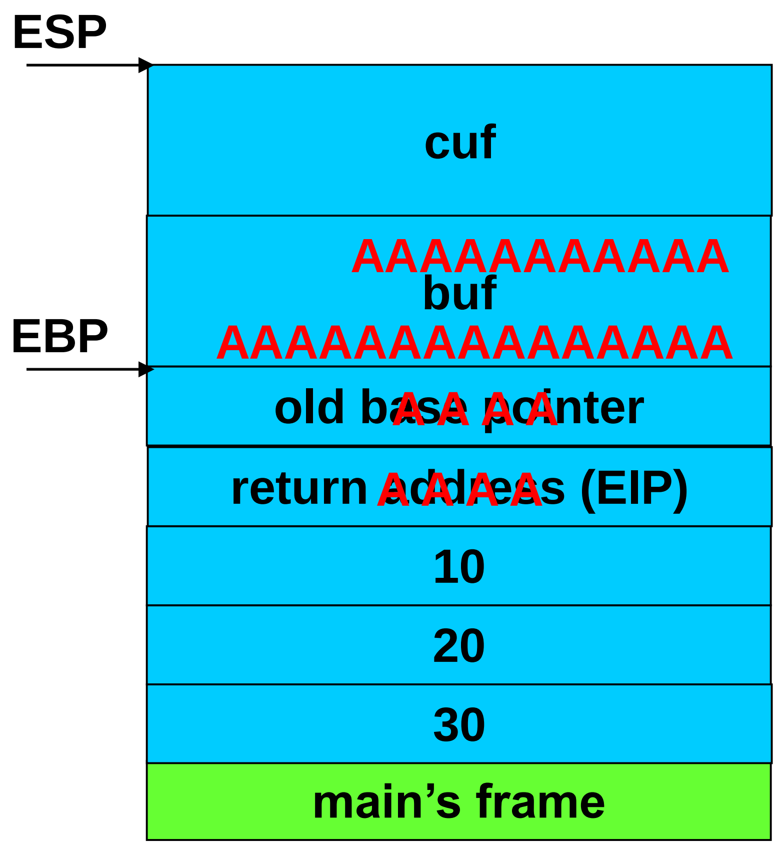
```
void func(int a, int b, int c){
    char buf[10];
    char cuf[20];
}

void main(){
    func(10,20,30);
}
```



Bo Luo

Buffer Overflow

- What we can do?
  - Overflow **`buf`**:
    - with malicious input data?
  - Rewrite the return address
  - Now you can run any program
    - As long as you know where it is
    - To be executed after function finishes
    - Most of attacks: exec a shell



Buffer Overflow

- So what?
  - Shell runs with same permissions as program we overflowed!
- Background: more on Unix

Buffer Overflow

- Background: more on Unix
- Unix set-uid mechanism
  - A user can execute a program if the program file has "x" bit set for the user
  - Typically the program process will have the invoker's privilege
  - If the program file also has the set-uid bit set for the owner ("s" is shown for the owner), then the program will also have the program owner's privilege. We call such programs "set-uid programs".

Buffer Overflow

- Background: more on Unix
- Unix set-uid mechanism
  - Provides a path for *privilege elevation*
  - There are legitimate needs for elevating a process' privilege to perform its jobs, *e.g.* passwd command.

Buffer Overflow

- So what?
  - Shell runs with same permissions as program we overflowed!
  - If you successfully overflow a program
    - Owned by the root
    - Has the set-uid bit set for the owner
    - Program fails to ensure that a write to a buffer is always within its bound.
      - How do you know?
      - You get “Segmentation fault” with malicious input
    - Invoke a “shellcode”
  - You can invoke a shell as root…

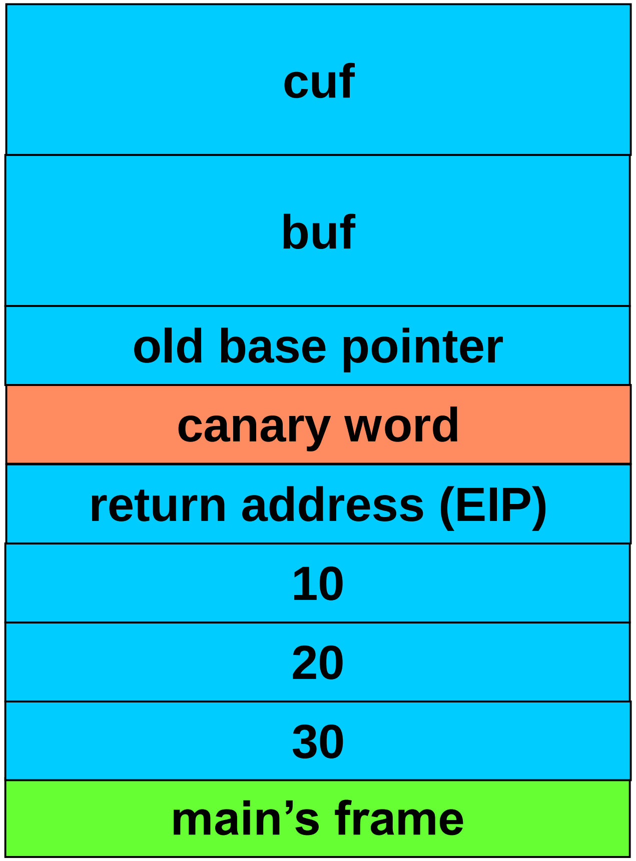
Buffer Overflow

- When buffer overflow happens, data structures in memory will be corrupted, potentially changing the program's behavior.
  - In many cases it can lead to the execution of arbitrary code by attackers
- A common problem for unsafe programming languages such as C and C++.
- *Local privilege escalation vulnerability*, i.e. an attacker who already obtained local access on the system can escalate his privilege.
  - If the setuid program is owned by root, an attacker who has user account privilege may gain root privilege on the system.

Bo Luo

Buffer Overflow

- Buffer overflow controls
  - Tools: ProPolice, Stackguard
  - Idea: use a "canary" before return addr
    - Canary = random number
    - Put there before func call
    - Check after function finishes
    - If canary isn't dead, continue



TOCTTOU

- Time of Check to Time of Use
- Real world example, purchase at a store:

Time of check

  - Costs $100
  - You count out the money on the counter

  - Cashier turns around, you take $20 back

Time of use

  - Cashier doesn't notice
  - Still get the $100 item

TOCTTOU

- Software security example: pseudocode for opening file stuff.txt:

Time of check

Time of use

```
if (permission(user, stuff.txt))
    open(stuff.txt)
else
    return failure
```

TOCTTOU

- Software security example: pseudocode for opening file stuff.txt:

Time of check

Time of use

```
if (permission(user, stuff.txt))
   open(stuff.txt)
else
   return failure
```

- Suppose that stuff.txt is a symlink
- What would happen if we switched the link to a different file?

Bo Luo

TOCTTOU

- TOCTTOU is unlikely?
  - Timing would have to be perfect.
- But:
  - can run program over and over
  - only have to get it right once
  - can run many other programs to lengthen time between check and open

OpenSSL “Heartbleed” Bug

- Announced April, 2014. (But bad code checked in December 31, 2011!)
- Exploits a programming mistake in the OpenSSL implementation of the TLS “heartbeat hello’’ extension.
  - Heartbeat protocol is used to keep a TLS connection alive without continuously transferring data.
  - One endpoint (e.g., a Web browser) sends a HeartbeatRequest message containing a payload to the other endpoint (e.g. a Web server).
  - The server then sends back a HeartbeatReply message containing the same payload.
  - “Buffer over-read” error caused by a failure to check for an invalid read-length parameter.

OpenSSL "Heartbleed" Bug

- Heartbeat Request and Response Messages

```
struct {
HeartbeatMessageType type;
uint16 payload_length;
opaque payload[HeartbeatMessage.payload_length];
opaque padding[padding_length];
} HeartbeatMessage;
```

Problem: no check that payload_length matches the actual length of the payload

- The total length of a HeartbeatMessage MUST NOT exceed 2^14 or max_fragment_length.

```
type: heartbeat_request or heartbeat_response
payload_length: The length of the payload
payload: The payload consists of arbitrary content
padding: The padding is random content that MUST
be ignored by the receiver.
```

OpenSSL “Heartbleed” Bug

Heartbeat sent to victim

SSLv3 record:

Length
4 bytes

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_REQUEST	65535 bytes	1 byte	

Victim’s response

SSLv3 record:

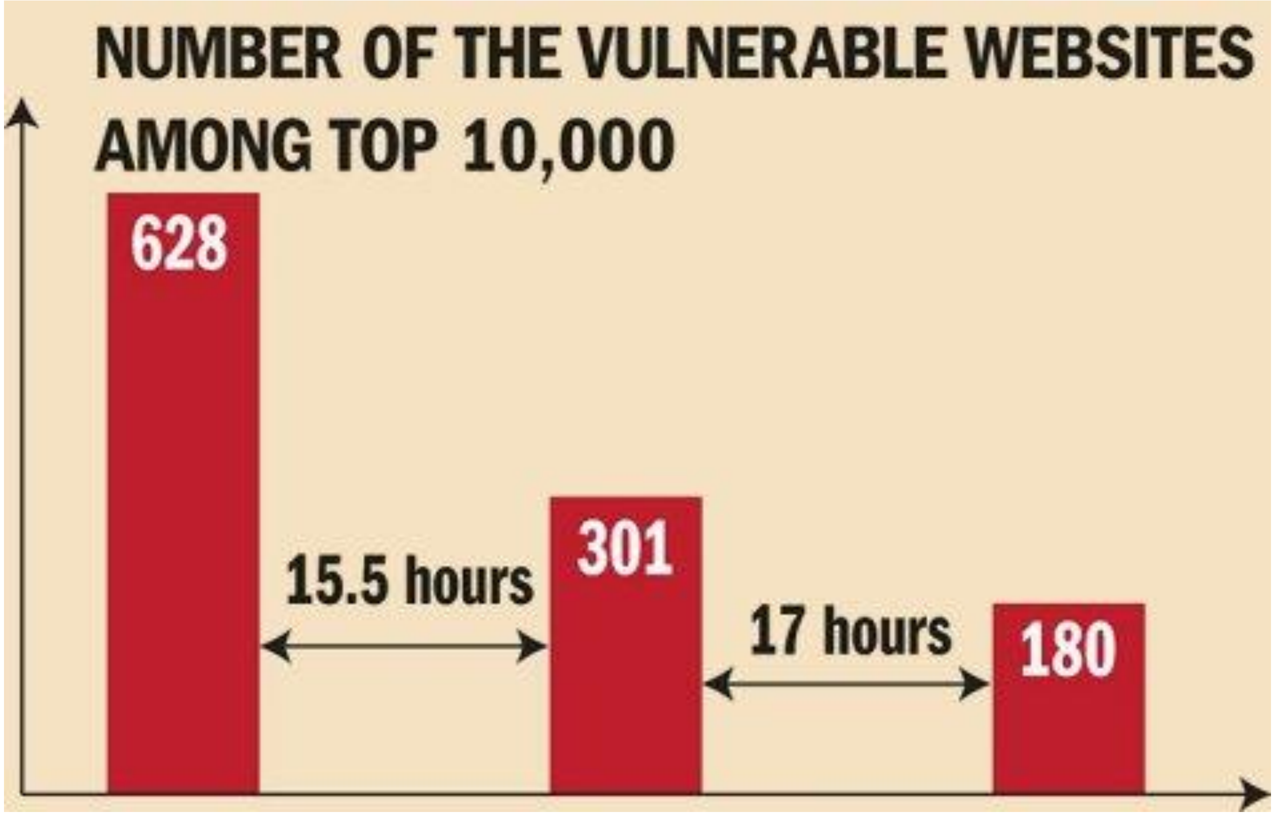
Length
65538 bytes

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_RESPONSE	65535 bytes	65535 bytes	

From http://www.theregister.co.uk/2014/04/09/heartbleed_explained/

OpenSSL “Heartbleed” Bug



Non-malicious Program Errors

- Recap:
  - Software testing: code does what it's supposed to do.
  - Software security: code doesn’t do anything it isn’t supposed to do.
    - Much harder
  - Program errors could be exploited by adversaries to: gain control of the system, deploy Trojan horses, etc.

Open Source vs. Closed Source

- Discussion: Which is better: Open source or closed?
  - *Argument*: Closed source more secure because it's harder to find flaws to exploit.
  - *Argument*: Open source more secure because more eyes on code.
  - What's your take?

Open Source vs. Closed Source

- Discussion: Which is better: Open source or closed?
  - *Argument*: Closed source more secure because it’s harder to find flaws to exploit.

  - but there are tools for finding flaws
  - patches tell you where to look
  - fewer people looking at code
  - often longer to release fixes

Open Source vs. Closed Source

- Discussion: Which is better: Open source or closed?
  - *Argument*: Open source more secure because more eyes on code.
  - Do you look at the code?
  - Code authors can be temporary, weekend warriors
  - Often not very strict quality standards
    - Kernels usually good, but drivers, other software packages can be shoddy
  - Code might make job easier on hacker
    - Can just do a grep on source for vulnerable functions

Patching

- Patching OS and applications
  - Importance of patching
  - Timing: vulnerability window
  - 0-day vulnerabilities
  - Vulnerability scanning

Bo Luo

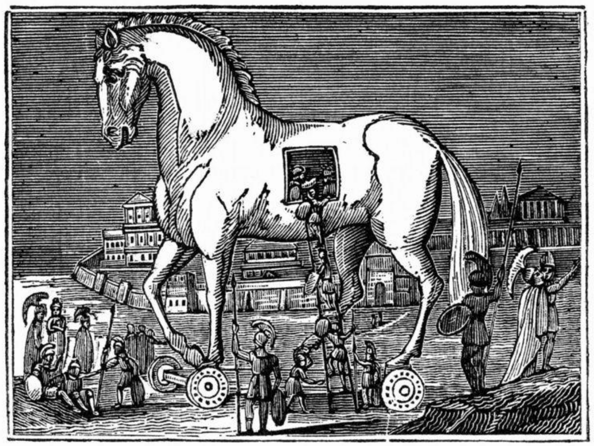
Malicious Software

- Malicious code: designed to do things “it isn’t supposed to do”
  - virus
  - trojan horse
  - logic bomb
  - time bomb (special case of logic bomb)
  - trapdoor
  - worm
  - rabbit

Bo Luo

Trojan Horses

- Trojan horses: program with
  - Open, known effect
  - And a secret effect
- Example: game that searches hard drive for passwords
- Propagating Trojans: Trojans which make copies of themselves



Trojans Deceived.

Trojan Horses

- The secret effects of Trojan horses
  - Control the computer (Zombie computers)
  - Steal information: passwords, bank accounts, credit card numbers, SSN, etc.
  - Install (malicious) software
  - Monitor and control hardware: key logger, watch screen, view webcam
  - More?

Viruses

- Viruses: program which
  - infects other files (inserts itself into)
  - performs some action
- Many types:
  - boot sector
  - executable file infector
  - multipartite (different targets e.g. either boot sector or exe)
  - encrypted viruses
  - polymorphic viruses
  - macro virus

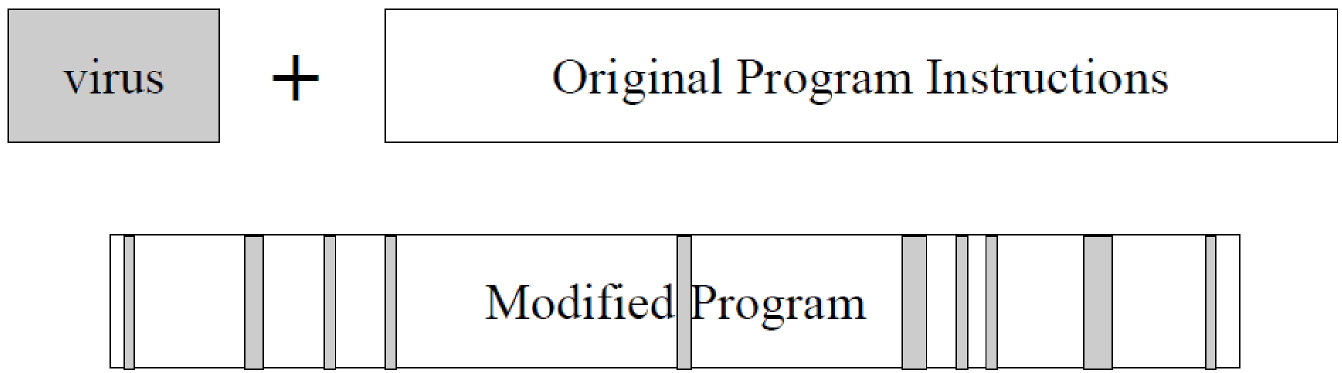
Viruses

- How Viruses Attach?
  - Appended (prepended)
  - Surrounding
  - Replace
  - Example: windows .com precedence over .exe
    - virus is calc.com
    - when you run calc
    - calc.com runs
    - then it calls calc.exe
    - virus renames itself to calc.exe and then moves old calc.exe to different filename or hidden filename ro directory that’s not often accessed

Bo Luo

Viruses

- How virus attach
  - Integrated



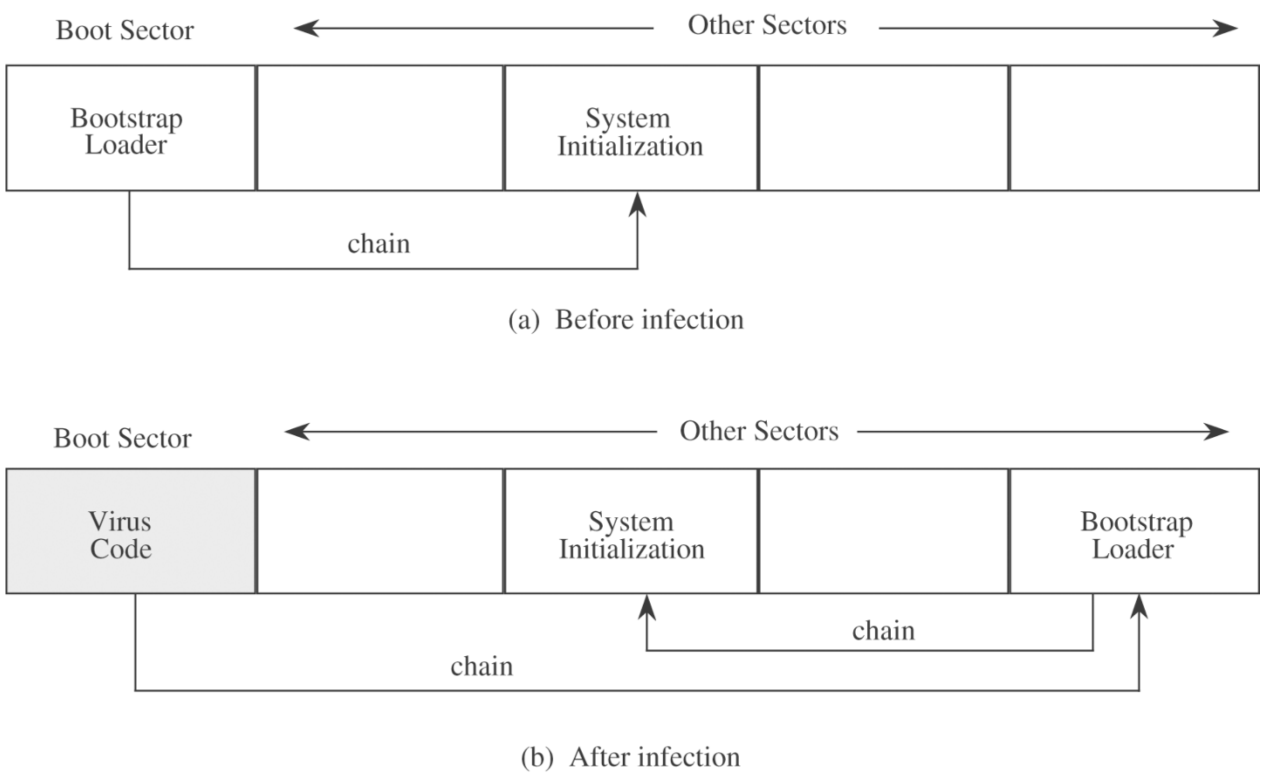
  - Requires more detailed knowledge about program structure

Viruses

- Boot sector virus
  - Normal boot operation:
    - BIOS
    - Master Boot Record (MBR)
    - Partition boot sector, aka. volume boot sector/record
    - Operating system
  - Virus example: Michelangelo (1991)
    - moved MBR someplace else (last sector of root dir)
    - copied itself into MBR
    - on boot, Michelangelo ran, then normal MBR
    - spread by copying itself to floppies
    - March 6 (artist’s bday), trashes hard disk

Viruses

- Boot Sector Virus: Chaining
  - Chaining allows for larger bootstraps, but eases insertion of virus



Viruses

- Document Virus
  - Macros for app programs (e.g. MS-VBA)
    - in Word, excel, etc.
  - Default template docs popular place:
    - MS-Office: Normal.dot, Personal.xls, Blank.pot

Viruses

- Duration of viruses
  - Transient: only runs when infected host program runs
  - Resident: stays in memory; runs even when host program isn't running

Viruses

- Virus Detection
  - detect change of file size
    - virus writer counter move: remove or compress part of original file
  - look for virus signature
    - virus writer counter-move: polymorphism, encryption, use a kit to write a different virus with similar effect

Discussions

- Differences between viruses and Trojan horses?

  病毒自我复制

  Trojan：远程控制的黑客工具，具有隐蔽性和非授权性的特点。

  https://zhidao.baidu.com/question/830514.html

- True of false:
  - Viruses can only affect MS-Windows.
  - Viruses can modify hidden or read only files.
  - Can't remain in memory after power-off
  - Viruses can't infect hardware false, no destory

Worms

- Worms: propagates from one computer to other using network
- Morris Worm – Nov. 2, 1988
  - First worm in the history
  - Written by Robert Morris, then a student at Cornel
  - Released from MIT
  - “to gauge the size of the Internet”: spreads over the internet
  - Queries the target before infecting it.
  - Re-infects targets at 1/7 rate → DoS attacks
  - Robert Morris was convicted (1990) of violating the Computer Fraud and Abuse Act: three years of probation, 400 hours of community service, a fine of $10,050, and the costs of his supervision.

Rootkits

- Rootkits: set of tools installed secretly
  - To gain root or administrative access to your computer.
  - User level: replaces or modifies user or admin programs.
  - Kernel level: modifies OS kernel to include backdoors.

Rootkits

- Sony XCP Rootkit 版权保护恶意行为，反复制，隐藏性，偷偷摸摸的
  - Intention: copy protection of CDs
  - Affects MS-Windows
  - Installs itself through AutoRun
  - Three components:
    - Anti-copying program
    - Stealth component: hides program’s existence
    - Phone home “feature”: contacts Sony
      - Ostensibly for graphics, ads, etc.
      - Tells Sony when disk is played, and from where
  - CD contains no uninstaller

Sony XCP EULA

- Program loaded with autorun
  - Displays EULA
  - Runs before user agrees
  - Expect program to load when inserting music CD?
- User agree to “a small proprietary software program ... intended to protect the audio files embodied on the CD.”
- Doesn't mention
  - “phone home” feature
  - It protects all XCP CDs, not just this one

Malware classification

- We have covered: virus, trojan horse, worm, rootkit
- Sometimes hard to classify
  - Worm.Win32.GetCodec.a
  - Converts MP3s to WMAs (but doesn't change file .mp3 extension)
  - Adds to WMA link to infected webpage
  - When file is played, IE is opened to infected page
  - User is prompted to download a codec
  - If user agrees, trojan is installed, which gives attacker control of machine

Malware Controls

- Developmental Controls
  - **Good software engineering practice**
  - Modularity
  - Encapsulation, information hiding
  - Separation, isolation
  - Layering
  - Testing
  - Peer reviews
  - Designing good specs
  - Least astonishment
  - Proofs of program correctness
  - Fail safe mechanisms

Malware Controls

- Operating Systems Controls
  - trusted software
  - protection, confinement
  - limited privilege
  - logging