Software Security

Hardware is easy to protect: lock it in a room, chain it to a desk, or buy a spare. Information poses more of a problem. It can exist in more than one place; be transported halfway across the planet in seconds; and be stolen without your knowledge.

—Bruce Schneier

Tamer ABUHMED
School of Computer and Information Engineering
INHA University

Why Software?

- Why is software as important to security as crypto, access control, protocols?
- Virtually all of information security is implemented in software
- If your software is subject to attack, your security can be broken
  - Regardless of strength of crypto, access control or protocols
- Software is a poor foundation for security

Bad Software is Ubiquitous

- NASA Mars Lander (cost $165 million)
  - Crashed into Mars due to…
  - …error in converting English and metric units of measure
  - Believe it or not
- Denver airport
  - Baggage handling system --- very buggy software
  - Delayed airport opening by 11 months
  - Cost of delay exceeded $1 million/day
  - What happened to person responsible for this fiasco?
- MV-22 Osprey
  - Advanced military aircraft
  - Undiscovered error in the aircraft's control software caused it to decelerate in response to the pilot's attempts to reset the software.
  - Faulty software can be fatal.

Software Issues

Alice and Bob

- Find bugs and flaws by accident
- Hate bad software…
- …but must learn to live with it
- Must make bad software work

Trudy

- Actively looks for bugs and flaws
- Likes bad software…
- …and tries to make it misbehave
- Attacks systems via bad software

Complexity

- “Complexity is the enemy of security”, Paul Kocher, Cryptography Research, Inc.

System	Lines of Code (LOC)
Netscape	17 million
Space Shuttle	10 million
Linux kernel 2.6.0	5 million
Windows XP	40 million
Mac OS X 10.4	86 million
Boeing 777	7 million

- A new car contains more LOC than was required to land the Apollo astronauts on the moon

Lines of Code and Bugs

- Conservative estimate: 5 bugs/10,000 LOC
- **Do the math**
  - Typical computer: 3k exe's of 100k LOC each
  - Conservative estimate: 50 bugs/exe
  - So, about 150k bugs per computer
  - So, 30,000-node network has 4.5 billion bugs
  - Maybe only 10% of bugs security-critical and only 10% of those remotely exploitable
  - Then "only" 45 million critical security flaws!

Software Security

- Program flaws (unintentional)
  - Buffer overflow - covered
  - Incomplete mediation - covered
  - Race conditions
- Malicious software (intentional)
  - Viruses - covered
  - Worms - covered
  - Other breeds of malware

Program Flaws

- An **error** is a programming mistake
  - To err is human
- An error may lead to incorrect state: **fault**
  - A fault is internal to the program
- A fault may lead to a **failure**, where a system departs from its expected behavior
  - A failure is externally observable

Example

```
char array[10];
for(i = 0; i < 10; ++i)
     array[i] = `A`;
array[10] = `B`;
```

- This program has an **error**
- This error might cause a **fault**
  - Incorrect internal state
- If a fault occurs, it might lead to a **failure**
  - Program behaves incorrectly (external)
- We use the term **flaw** for all of the above

Secure Software

- In software engineering, try to ensure that a program does what is intended
- ***Secure*** software engineering requires that software **does what is intended…**
- **…and nothing more**
- Absolutely secure software is impossible
  - But, absolute security ***anywhere*** is impossible
- How can we manage software risks?

Program Flaws

- Program flaws are **unintentional**
  - But can still create security risks
- We'll consider 3 types of flaws
  - Buffer overflow (smashing the stack)
  - Incomplete mediation
  - Race conditions
- These are the most common problems

Buffer Overflow

Possible Attack Scenario

- Users enter data into a Web form
- Web form is sent to server
- Server writes data to array called buffer, without checking length of input data
- Data "overflows" buffer
  - Such overflow might enable an attack
  - If so, attack could be carried out by anyone with Internet access

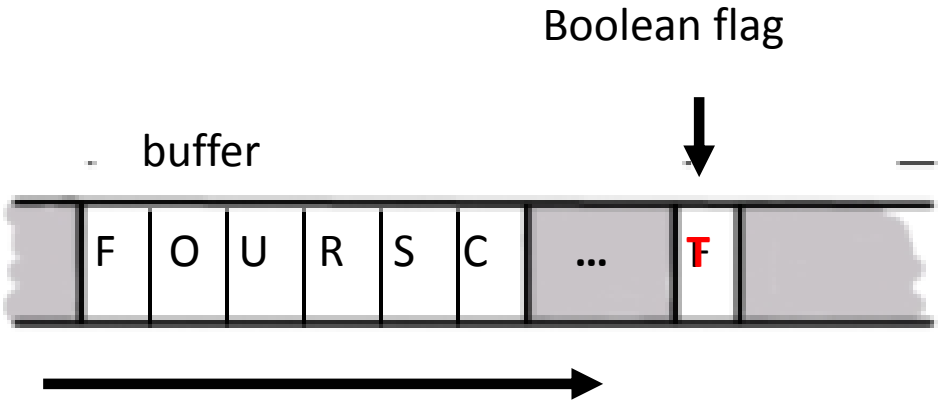
Buffer Overflow

```
int main(){
    int buffer[10];
    buffer[20] = 37;}
```

- **Q:** What happens when code is executed?
- **A:** Depending on what resides in memory at location "buffer[20]"
  - Might overwrite **user** data or code
  - Might overwrite **system** data or code
  - Or program could work just fine

Simple Buffer Overflow

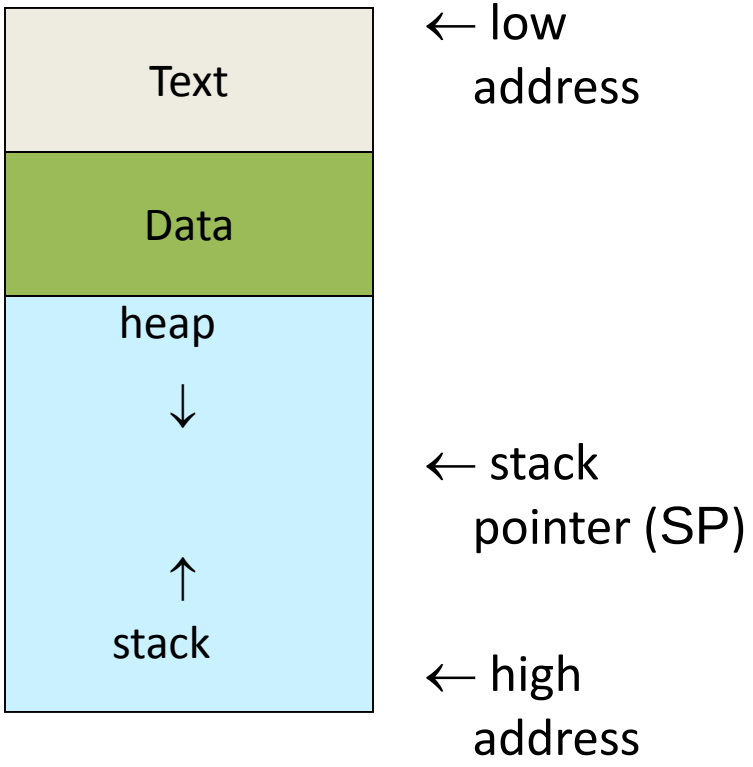
- Consider boolean flag for authentication
- Buffer overflow could overwrite flag allowing anyone to authenticate



- In some cases, Trudy need not be so lucky as in this example

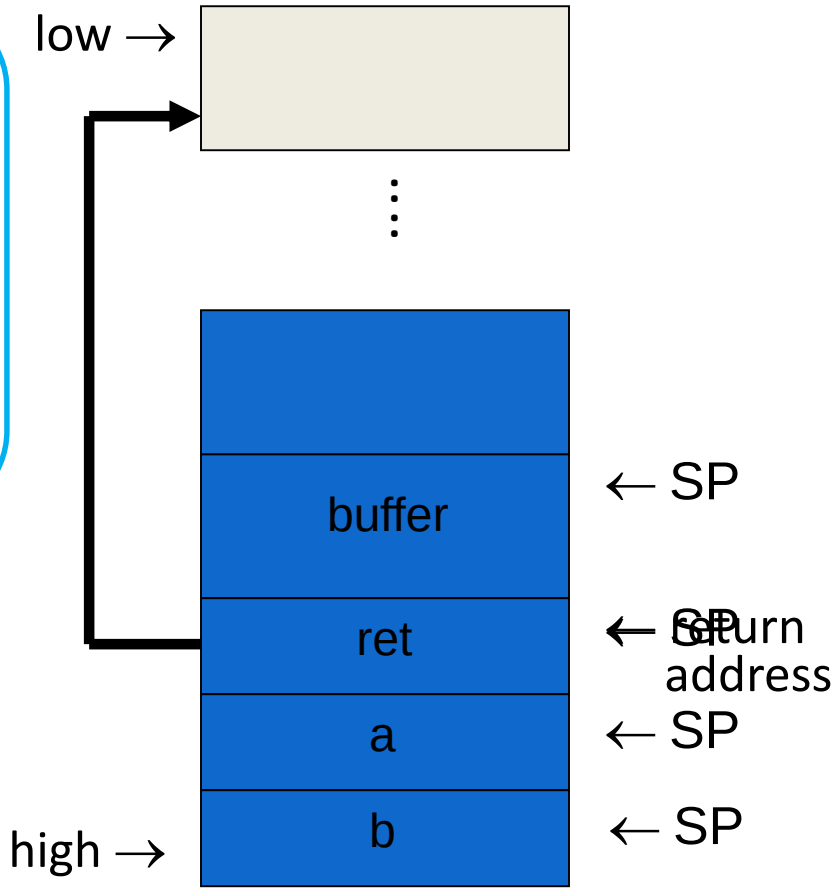
Memory Organization

- **Text** == code
- **Data** == static variables
- **Heap** == dynamic data
- **Stack** == "scratch paper"
  - Dynamic local variables
  - Parameters to functions
  - Return address



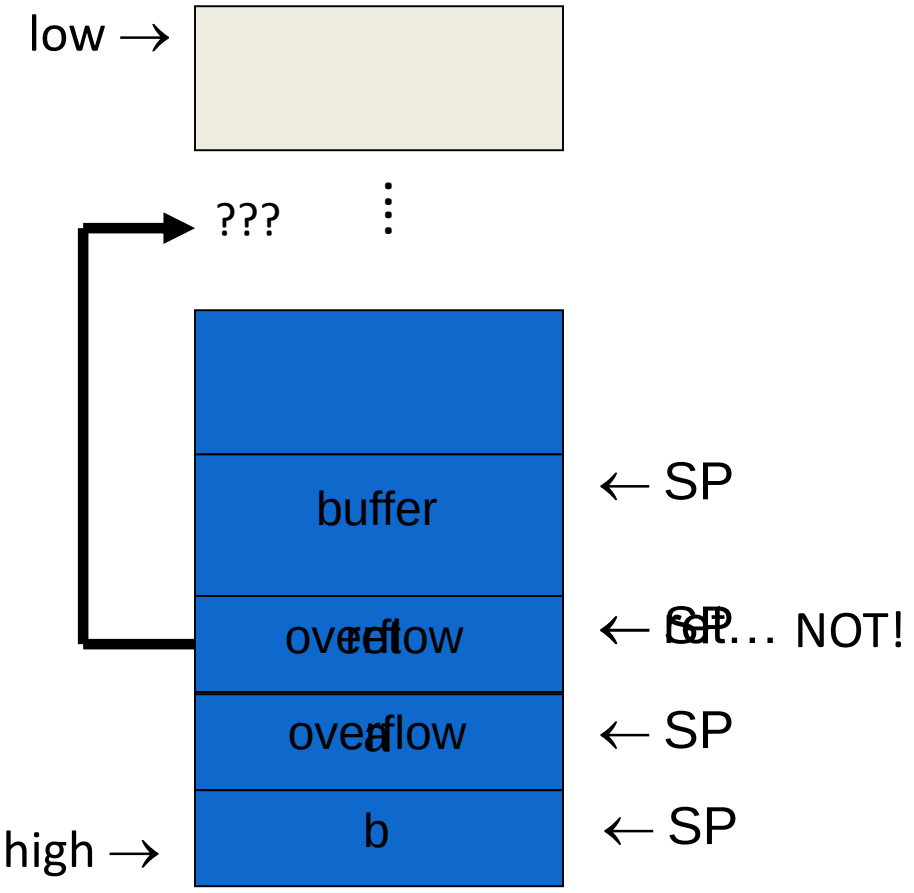
Simplified Stack Example

```
void func(int a, int b){
   char buffer[10];
}
void main(){
   func(1, 2);
}
```



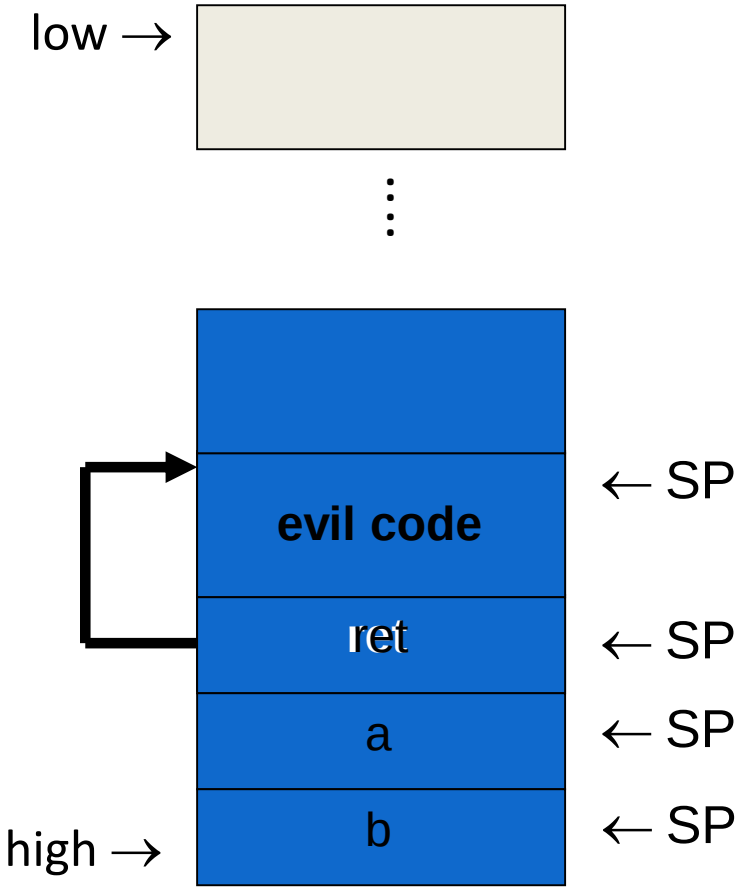
Smashing the Stack

- What happens if buffer overflows?
- Program "returns" to wrong location
- A crash is likely



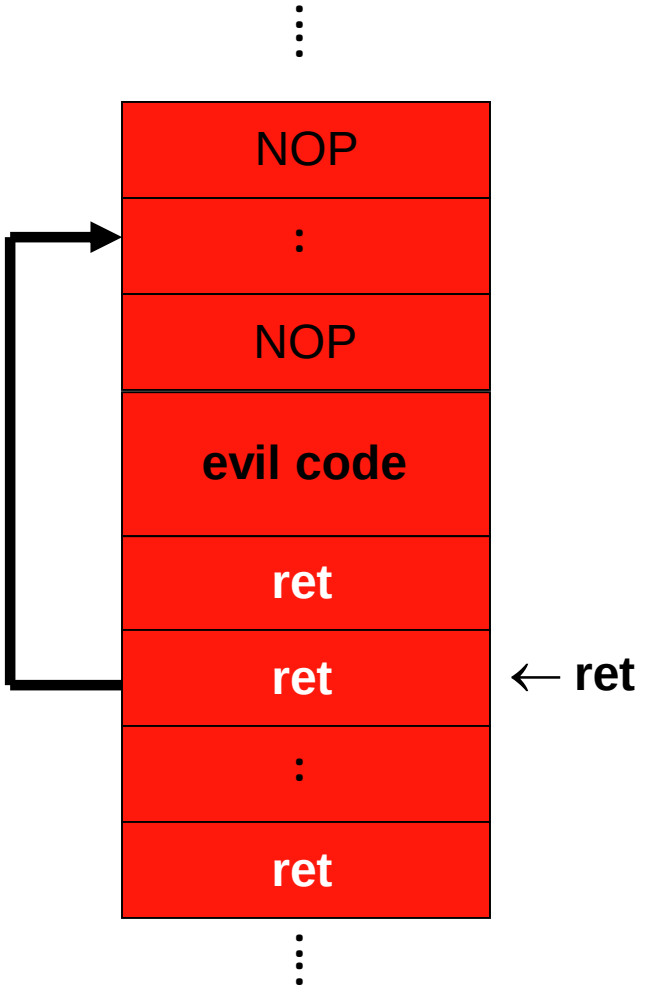
Smashing the Stack

- Trudy has a better idea…
- **Code injection**
- Trudy can run code of her choosing…
  - …on your machine



Smashing the Stack

- Trudy may not know…
  1) Address of evil code
  2) Location of **ret** on stack
- Solutions
  1) Precede evil code with NOP "landing pad"
  2) Insert **ret** many times

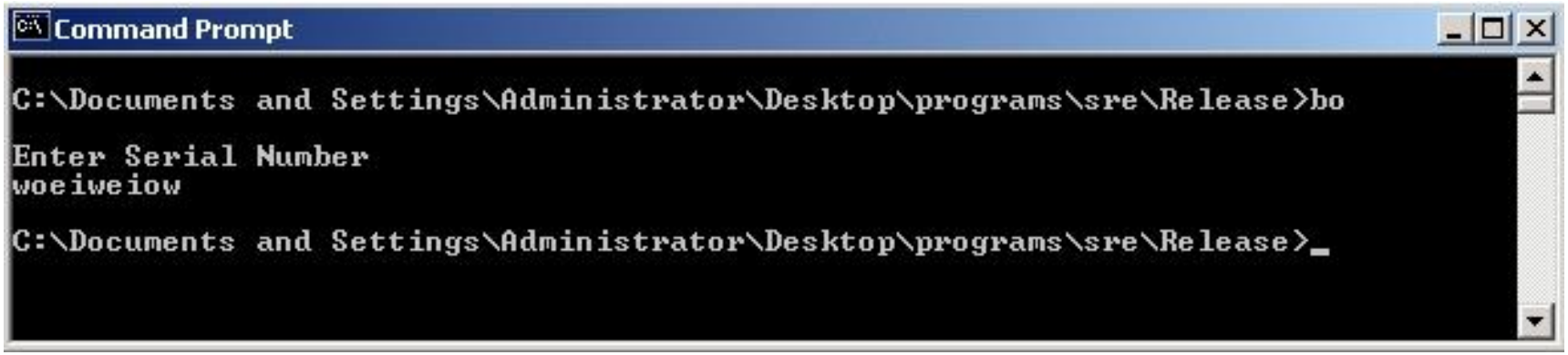


Stack Smashing Summary

- A buffer overflow must exist in the code
- Not all buffer overflows are exploitable
  - Things must align properly
- If exploitable, attacker can **inject code**
- Trial and error is likely required
  - Fear not, lots of help is available online
  - Smashing the Stack for Fun and Profit, Aleph One
- Stack smashing is "attack of the decade"
  - Regardless of the current decade
  - Also heap overflow, integer overflow, …

Stack Smashing Example

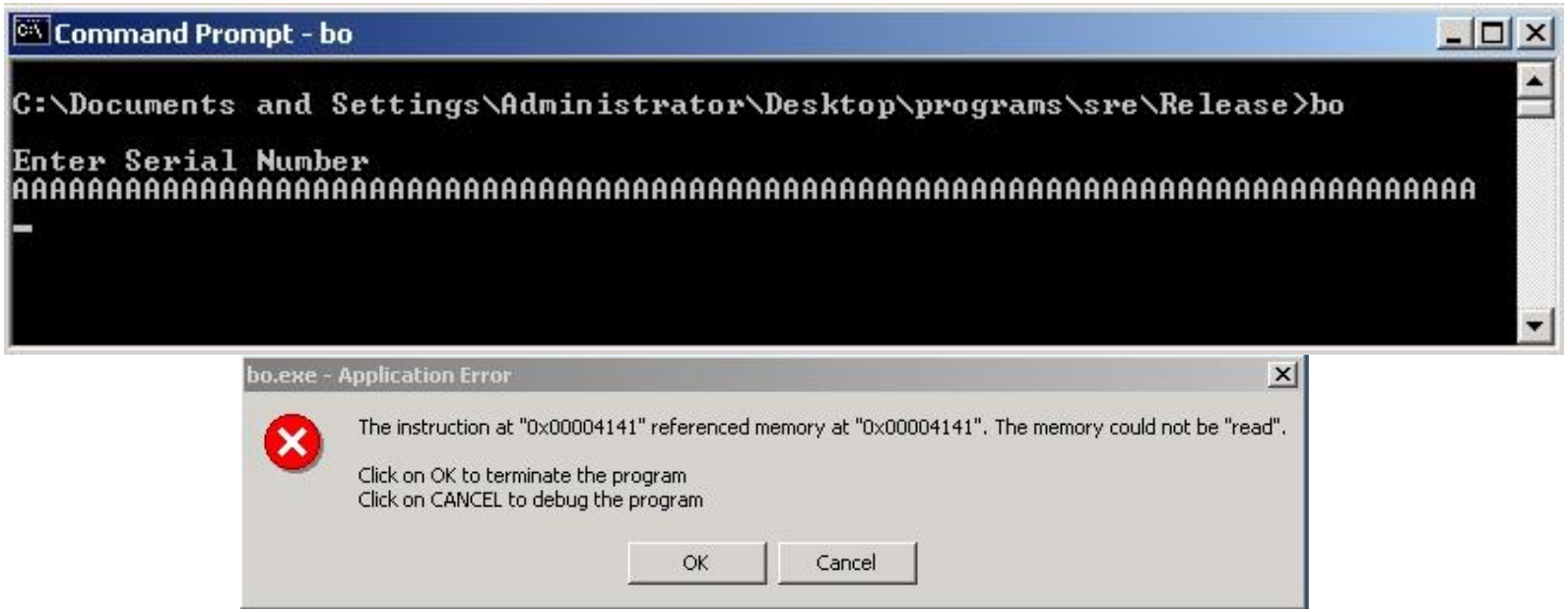
- Program asks for a serial number that the attacker does not know
- Attacker does **not** have source code
- Attacker does have the executable (exe)



- Program quits on incorrect serial number

Buffer Overflow Present?

- By trial and error, attacker discovers apparent buffer overflow



- Note that 0x41 is ASCII for “A”
- Looks like **ret** overwritten by 2 bytes!

Disassemble Code

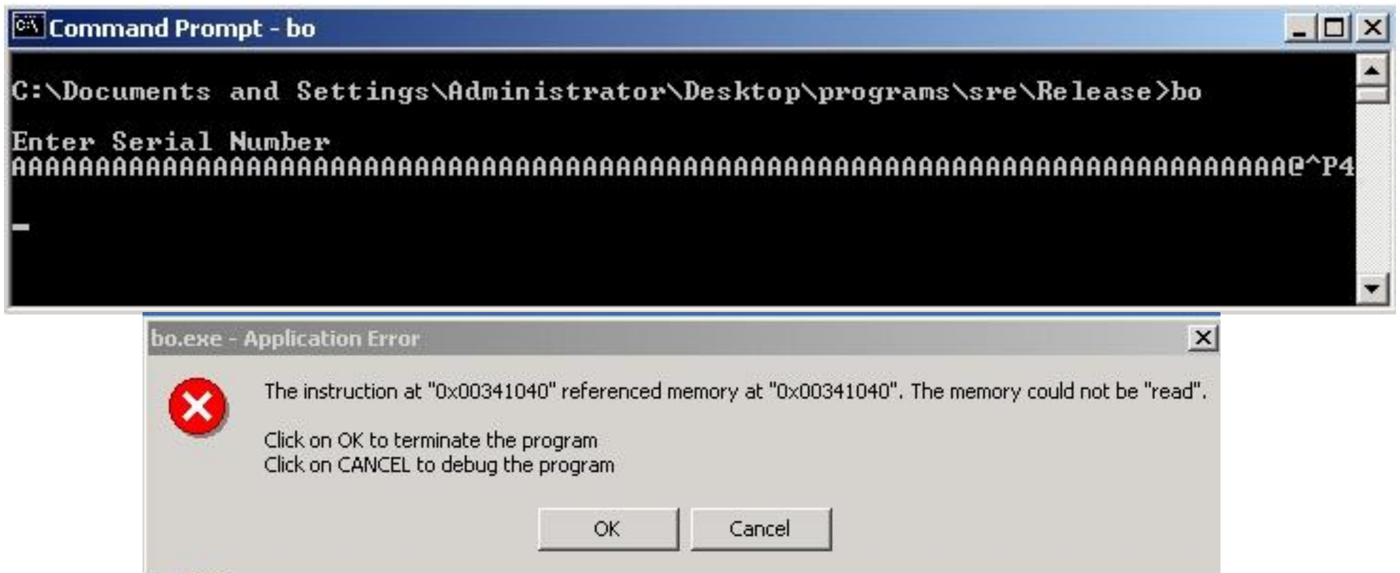
- Next, disassemble bo.exe to find

```
.text:00401000
.text:00401000                sub     esp, 1Ch
.text:00401003                push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008                call    sub_40109F
.text:0040100D                lea     eax, [esp+20h+var_1C]
.text:00401011                push    eax
.text:00401012                push    offset aS       ; "%s"
.text:00401017                call    sub_401088
.text:0040101C                push    8
.text:0040101E                lea     ecx, [esp+2Ch+var_1C]
.text:00401022                push    offset aS123n456 ; "S123N456"
.text:00401027                push    ecx
.text:00401028                call    sub_401050
.text:0040102D                add     esp, 18h
.text:00401030                test    eax, eax
.text:00401032                jnz     short loc_401041
.text:00401034                push    offset aSerialNumberIs ; "Serial number is correct.\n"
.text:00401039                call    sub_40109F
.text:0040103E                add     esp, 4
```

  - The goal is to exploit buffer overflow to jump to address 0x401034

Buffer Overflow Attack

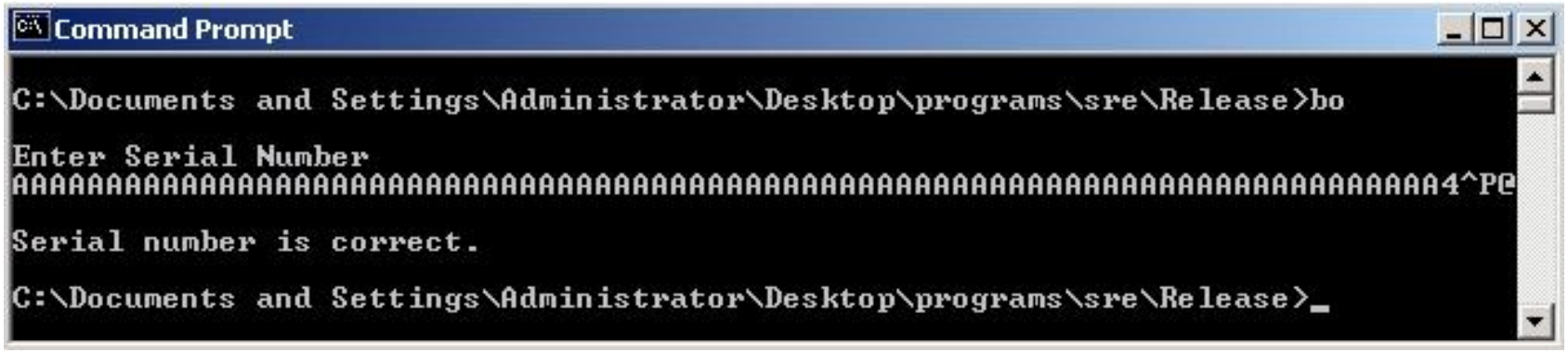
- Find that, in ASCII, 0x401034 is “@^P4”



- Byte order is reversed? Why?
- X86 processors are “little-endian”

Overflow Attack, Take 2

- Reverse the byte order to “4^P@” and…



  - Success! We’ve bypassed serial number check by exploiting a buffer overflow
  - What just happened?
    - Overwrote return address on the stack

Buffer Overflow

- Attacker did **not** require access to the source code
- Only tool used was a disassembler to determine address to jump to
- Find desired address by trial and error?
  - Necessary if attacker does not have exe
  - For example, a remote attack

Source Code

- Source code for buffer overflow example

- Flaw easily found by attacker…
- **…without access to source code!**

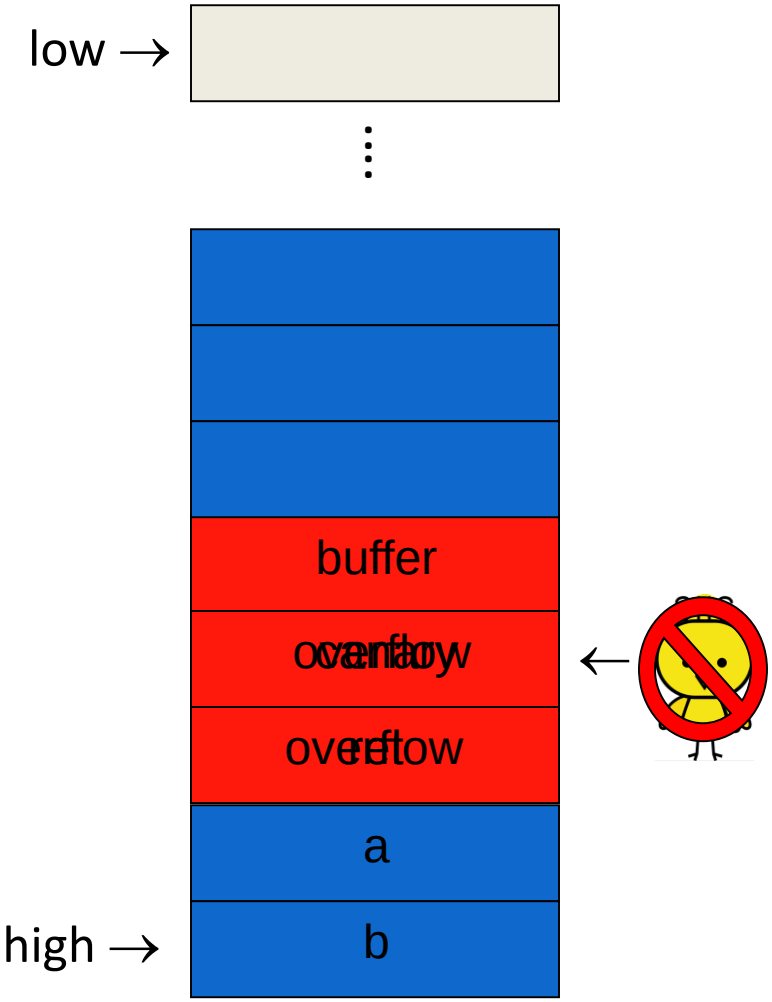
```
#include <stdio.h>
#include <string.h>

main()
{
    char in[75];

    printf("\nEnter Serial Number\n");

    scanf("%s", in);

    if(!strncmp(in, "S123N456", 8))
    {
       printf("Serial number is correct.\n");
    }
}
```

Stack Smashing Defenses

- Employ **non-executable stack**
  - "No execute" **NX bit** (if available)
  - Seems like the logical thing to do, but some real code executes on the stack (Java, for example)
- Use a **canary**
- Address space layout randomization (**ASLR**)
- Use **safe languages** (Java, C#)
- Use **safer C functions**
  - For unsafe functions, safer versions exist
  - For example, strncpy instead of strcpy

Stack Smashing Defenses

- **Canary**
  - Run-time stack check
  - Push canary onto stack
  - Canary value:
    - Constant 0x000aff0d
    - Or may depends on **ret**



Microsoft's Canary

- Microsoft added **buffer security check** feature to C++ with /GS compiler flag
  - Based on canary (or "security cookie")

Q: What to do when canary dies?

A: Check for user-supplied "handler"

- Handler shown to be subject to attack
  - Claim that attacker can specify handler code
  - If so, formerly "safe" buffer overflows become exploitable when /GS is used!

ASLR

- Address Space Layout Randomization
  - Randomize place where code loaded in memory
- Makes most buffer overflow attacks probabilistic
- Windows Vista uses 256 random layouts
  - So about 1/256 chance buffer overflow works?
- Similar thing in Mac OS X and other OSs
- Attacks against Microsoft's ASLR do exist
  - Possible to "de-randomize"

Buffer Overflow

- A major security threat yesterday, today, and tomorrow
- The good news?
- It is possible to reduced overflow attacks
  - Safe languages, NX bit, ASLR, education, etc.
- The bad news?
- Buffer overflows will exist for a long time
  - Legacy code, bad development practices, etc.

Input Validation

- Consider: `strcpy(buffer, argv[1])`
- A buffer overflow occurs if

  `len(buffer) < len(argv[1])`
- Software must **validate** the input by checking the length of `argv[1]`
- Failure to do so is an example of a more general problem: **incomplete mediation**

Input Validation

- Consider web form data
- Suppose input is validated on client
- For example, the following is valid

```
http://www.things.com/orders/final&custID=112&num
  =55A&qty=20&price=10&shipping=5&total=205
```

- Suppose input is not checked on server
  - Why bother since input checked on client?
  - Then attacker could send http message

```
http://www.things.com/orders/final&custID=112&num
  =55A&qty=20&price=10&shipping=5&total=25
```

Incomplete Mediation

- Linux kernel
  - Research has revealed many buffer overflows
  - Many of these are due to incomplete mediation
- Linux kernel is “good” software since
  - Open-source
  - Kernel — written by coding gurus
- Tools exist to help find such problems
  - But incomplete mediation errors can be subtle
  - And tools useful to attackers too!

Malware

Malicious Software

- Malware is not new…
  - Fred Cohen's initial virus work in 1980's, used viruses to break MLS systems
- Types of malware (lots of overlap)
  - **Virus** — passive propagation
  - **Worm** — active propagation
  - Trojan horse — unexpected functionality
  - Trapdoor/backdoor — unauthorized access
  - Rabbit — exhaust system resources

Where do Viruses Live?

- They live just about anywhere, such as…
- Boot sector
  - Take control before anything else
- Memory resident
  - Stays in memory
- Applications, macros, data, etc.
- Library routines
- Compilers, debuggers, virus checker, etc.
  - These would be particularly nasty!

Malware Examples

- Brain virus (1986)
- Morris worm (1988)
- Code Red (2001)
- SQL Slammer (2004)
- Botnets (currently fashionable)
- Future of malware?

Brain

- First appeared in 1986
- More annoying than harmful
- A prototype for later viruses
- Not much reaction by users
- What it did
  1. Placed itself in boot sector (and other places)
  2. Screened disk calls to avoid detection
  3. Each disk read, checked boot sector to see if boot sector infected; if not, goto 1
- Brain did nothing really malicious

Morris Worm

- First appeared in 1988
- What it tried to do
  - Determine where it could spread, then…
  - …spread its infection and…
  - …remain undiscovered
- Morris claimed his worm had a bug!
  - It tried to re-infect infected systems
  - Led to resource exhaustion
  - Effect was like a so-called rabbit

How Morris Worm Spread

- Obtained access to machines by…
  - User account **password guessing**
  - Exploit **buffer overflow** in fingerd
  - Exploit **trapdoor** in sendmail
- Flaws in fingerd and sendmail were well-known, but not widely patched

Bootstrap Loader

- Once Morris worm got access…
- "Bootstrap loader" sent to victim
  - 99 lines of C code
- Victim compiled and executed code
- Bootstrap loader fetched the worm
- Victim **authenticated** sender!
  - Don't want user to get a bad worm…

How to Remain Undetected?

- If transmission interrupted, code deleted
- Code encrypted when downloaded
- Code deleted after decrypt/compile
- When running, worm regularly changed name and process identifier (PID)

Morris Worm: Bottom Line

- Shock to Internet community of 1988
  - Internet of 1988 ***much*** different than today
- Internet designed to withstand nuclear war
  - Yet, brought down by one graduate student!
  - At the time, Morris' father worked at NSA…
- Could have been much worse
- Result? CERT, more security awareness
- But should have been a wakeup call

Code Red Worm

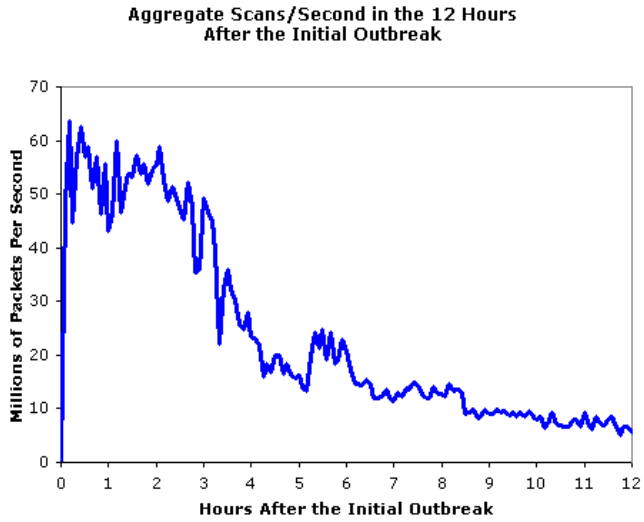
- Appeared in July 2001
- Infected more than **250,000 systems in about 15 hours**
- Eventually infected 750,000 out of about 6,000,000 vulnerable systems
- Exploited buffer overflow in Microsoft IIS server software
  - Then monitor traffic on port 80, looking for other susceptible servers

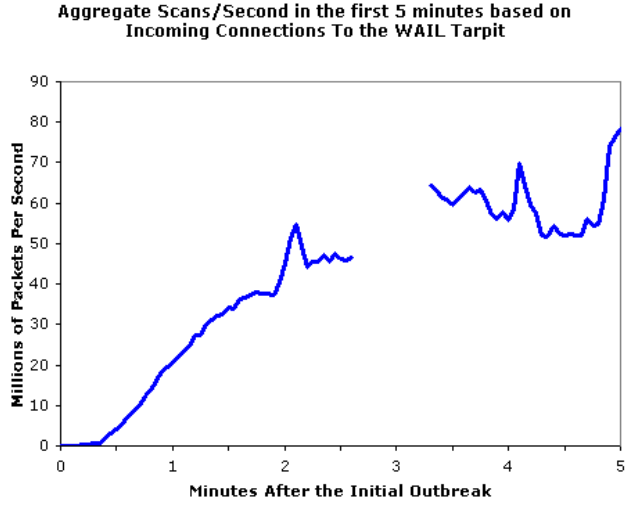
Code Red: What it Did

- Day 1 to 19 of month: spread its infection
- Day 20 to 27: distributed denial of service attack (DDoS) on `www.whitehouse.gov`
- Later version (several variants)
  - Included trapdoor for remote access
  - Rebooted to flush worm, leaving only trapdoor
- Some say it was “beta test for info warfare”
  - But no evidence to support this

SQL Slammer

- Infected **75,000 systems in 10 n**
- At its peak, infections doubled e
- Spread “too fast”…
- …so it “burned out” available bandwidth





Why was Slammer Successful?

- Worm size: **one 376-byte UDP packet**
- Firewalls often let one packet thru
  - Then monitor ongoing “connections”
- Expectation was that much more data required for an attack
  - So no need to worry about 1 small packet
- Slammer defied “experts”

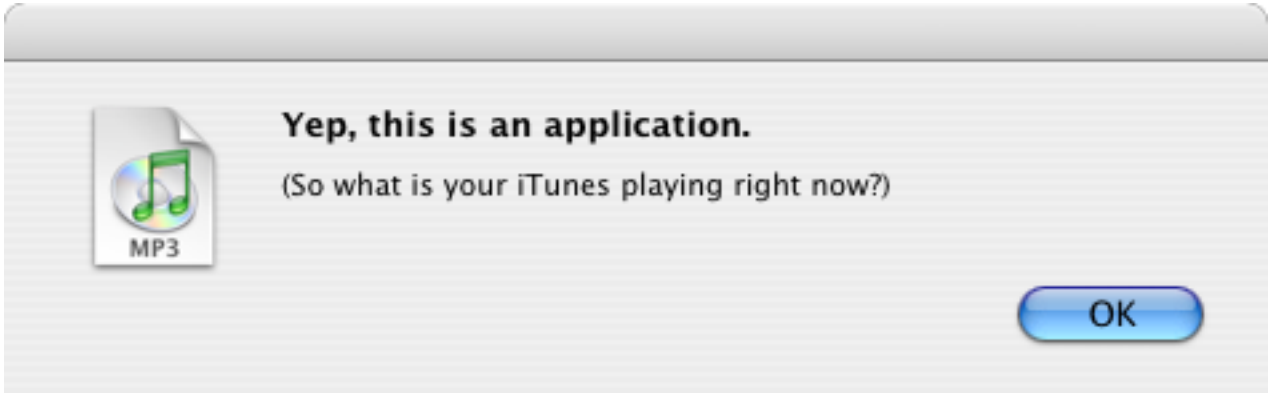
Trojan Horse Example

- Trojan: unexpected functionality
- Prototype trojan for the Mac
- File icon for freeMusic.mp3:

  - For a real mp3, double click on icon
    - iTunes opens
    - Music in mp3 file plays
  - But for freeMusic.mp3, unexpected results…

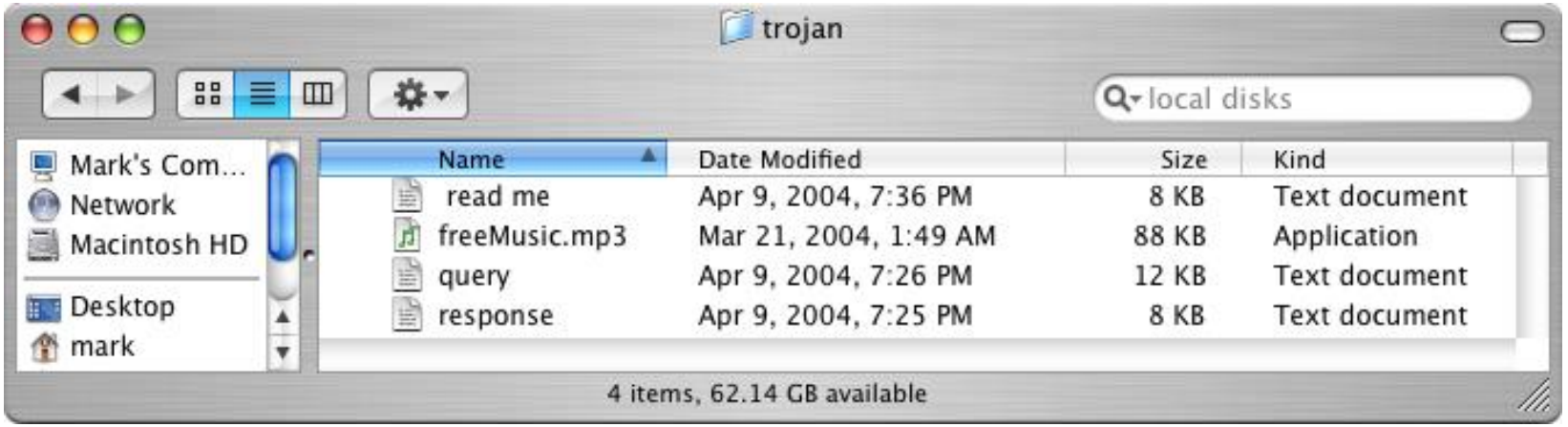
Mac Trojan

- Double click on freeMusic.mp3
  - iTunes opens (expected)
  - “Wild Laugh” (not expected)
  - Message box (not expected)



Trojan Example

- How does freeMusic.mp3 trojan work?
- This "mp3" is an application, not data



- This trojan is harmless, but…
- …could have done anything user could do
  - Delete files, download files, launch apps, etc.

Malware Detection

- Three common detection methods
  - Signature detection
  - Change detection
  - Anomaly detection
- We briefly discuss each of these
  - And consider advantages…
  - …and disadvantages

Signature Detection

- A **signature** may be a string of bits in exe
  - Might also use wildcards, hash values, etc.
- For example, W32/Beast virus has signature

  83EB 0274 EB0E 740A 81EB 0301 0000
  - That is, this string of bits appears in virus
- We can search for this signature in all files
- If string found, have we found W32/Beast?
  - Not necessarily — string could appear elsewhere
  - At random, chance is only $1/2^{112}$
  - But software is not random

Signature Detection

- Advantages
  - Effective on “ordinary” malware
  - Minimal burden for users/administrators
- Disadvantages
  - Signature file can be large (10s of thousands)…
  - …making scanning slow
  - Signature files must be kept up to date
  - ***Cannot detect unknown viruses***
  - Cannot detect some advanced types of malware
- The most popular detection method

Change Detection

- Viruses must live somewhere
- If you detect a file has changed, it might have been infected
- How to detect changes?
  - Hash files and (securely) store hash values
  - Periodically re-compute hashes and compare
  - If hash changes, file **might** be infected

Change Detection

- Advantages
  - Virtually no false negatives
  - Can even detect previously unknown malware
- Disadvantages
  - Many files change — and often
  - Many false alarms (false positives)
  - Heavy burden on users/administrators
  - If suspicious change detected, then what?
  - Might fall back on signature-based system

Anomaly Detection

- Monitor system for anything “unusual” or “virus-like” or potentially malicious or …
- Examples of “unusual”
  - Files change in some unexpected way
  - System misbehaves in some way
  - Unexpected network activity
  - Unexpected file access, etc., etc., etc., etc.
- But, we must first define “normal”
  - Normal can (and must) change over time

Anomaly Detection

- Advantages
  - Chance of detecting unknown malware
- Disadvantages
  - No proven track record
  - Trudy can make abnormal look normal (go slow)
  - Must be combined with another method (e.g., signature detection)
- Also popular in intrusion detection (IDS)
- Difficult unsolved (unsolvable?) problem
  - Reminds me of AI…

Future of Malware

- Recent trends
  - Encrypted, polymorphic, metamorphic malware
  - Fast replication/Warhol worms
  - Flash worms, slow worms
  - Botnets
- The future is bright for malware
  - Good news for the bad guys…
  - …bad news for the good guys
- Future of malware detection?

Encrypted Viruses

- Virus writers know **signature detection** used
- So, how to evade signature detection?
- Encrypting the virus is a good approach
  - Ciphertext looks like random bits
  - Different key, then different “random” bits
  - So, different copies have no common signature
- Encryption often used in viruses today

Encrypted Viruses

- How to detect encrypted viruses?
- Scan for the decryptor code
  - More-or-less standard signature detection
  - But may be more false alarms
- Why not encrypt the decryptor code?
  - Then encrypt the decryptor of the decryptor (and so on…)
- Encryption of limited value to virus writers

Polymorphic Malware

- Polymorphic worm
  - Body of worm is encrypted
  - Decryptor code is “mutated” (or “morphed”)
  - Trying to hide decryptor signature
  - Like an encrypted worm on steroids…

Q: How to detect?

A: Emulation — let the code decrypt itself

  - Slow, and anti-emulation is possible

Metamorphic Malware

- A metamorphic worm mutates before infecting a new system
  - Sometimes called “body polymorphic”
- Such a worm can, in principle, evade signature-based detection
- Mutated worm must function the same
  - And be “different enough” to avoid detection
- Detection is a difficult research problem

Metamorphic Worm

- One approach to metamorphic replication…
  - The worm is disassembled
  - Worm then stripped to a base form
  - Random variations inserted into code (permute the code, insert dead code, etc., etc.)
  - Assemble the resulting code
- Result is a worm with same functionality as original, but different signature

Warhol Worm

- "In the future everybody will be world-famous for 15 minutes" — Andy Warhol
- Warhol Worm is designed to infect the entire Internet in 15 minutes
- Slammer infected 250,000 in 10 minutes
  - "Burned out" bandwidth
  - Could **not** have infected entire Internet in 15 minutes — too bandwidth intensive
- Can rapid worm do "better" than Slammer?

A Possible Warhol Worm

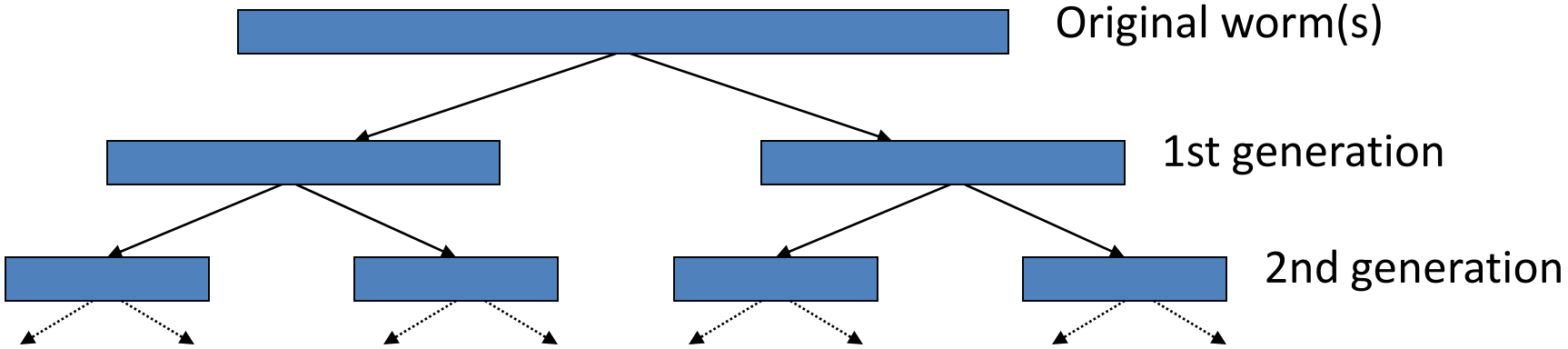
- Seed worm with an initial **hit list** containing a set of vulnerable IP addresses
  - Depends on the particular exploit
  - Tools exist for identifying vulnerable systems
- Each successful initial infection would attack selected part of IP address space
- Could infect entire Internet in 15 minutes!
- No worm this sophisticated has yet been seen in the wild (as of 2011)
  - Slammer generated random IP addresses

Flash Worm

- Can we do “better” than Warhol worm?
- Infect entire Internet in less than 15 minutes?
- Searching for vulnerable IP addresses is the slow part of any worm attack
- Searching might be bandwidth limited
  - Like Slammer
- **Flash worm** designed to infect entire Internet almost instantly

Flash Worm

- Predetermine **all** vulnerable IP addresses
  - Depends on details of the attack
- Embed these addresses in worm(s)
  - Results in huge worm(s)
  - But, the worm replicates, it splits
- No wasted time or bandwidth!



Flash Worm

- Estimated that ideal flash worm could infect the entire Internet in **15 seconds!**
  - Some debate as to actual time it would take
  - Estimates range from 2 seconds to 2 minutes
- In any case…
- …much faster than humans could respond
- So, any defense must be fully automated
- How to defend against such attacks?

Rapid Malware Defenses

- Master IDS watches over network
  - "Infection" proceeds on part of network
  - Determines whether an attack or not
  - If so, IDS saves most of the network
  - If not, only a slight delay
- Beneficial worm
  - Disinfect faster than the worm infects
- Other approaches?

Push vs Pull Malware

- Viruses/worms examples of “push”
- Recently, a lot of “pull” malware
- Scenario
  - A compromised web server
  - Visit a website at compromised server
  - Malware loaded on you machine
- Good paper: Ghost in the Browser

Botnet

- Botnet: a “network” of infected machines
- Infected machines are “bots”
  - Victim is unaware of infection (stealthy)
- Botmaster controls botnet
  - Generally, using IRC
  - P2P botnet architectures exist
- Botnets used for…
  - Spam, DoS attacks, keylogging, ID theft, etc.

Botnet Examples

- XtremBot
  - Similar bots: Agobot, Forbot, Phatbot
  - Highly modular, easily modified
  - Source code readily available (GPL license)
- UrXbot
  - Similar bots: SDBot, UrBot, Rbot
  - Less sophisticated than XtremBot type
- GT-Bots and mIRC-based bots
  - mIRC is common IRC client for Windows

More Botnet Examples

- Mariposa
  - Used to steal credit card info
  - Creator arrested in July 2010
- Conficker
  - Estimated 10M infected hosts (2009)
- Kraken
  - Largest as of 2008 (400,000 infections)
- Srizbi
  - For spam, one of largest as of 2008

Computer Infections

- Analogies are made between computer viruses/worms and biological diseases
- There are differences
  - Computer infections are much quicker
  - Ability to intervene in computer outbreak is more limited (vaccination?)
  - Bio disease models often not applicable
  - “Distance” almost meaningless on Internet
- But there are some similarities…

Computer Infections

- Cyber “diseases” vs biological diseases
- One similarity
  - In nature, too few susceptible individuals and disease will die out
  - In the Internet, too few susceptible systems and worm might fail to take hold
- One difference
  - In nature, diseases attack more-or-less at random
  - Cyber attackers select most “desirable” targets
  - Cyber attacks are more focused and damaging

Future Malware Detection?

- Malware today outnumbers “goodware”
  - Metamorphic copies of existing malware
  - Many virus toolkits available
  - Trudy: recycle old viruses, different signature
- So, may be better to “detect” good code
  - If code not on “good” list, assume it’s bad
  - That is, use **whitelist** instead of **blacklist**