Network Security

<CH 10>

Youn Kyu Lee
Hongik University

Why Software?

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

Bad Software

- Bad software is everywhere!
  - NASA Mars Lander (cost $165 million)
    - Crashed into Mars
    - Error in converting English and metric units of measure
  - Denver airport
    - Buggy baggage handling system
    - Delayed airport opening by 11 months
    - Cost of delay exceeded $1 million/day
  - MV-22 Osprey: Advanced military aircraft
    - Lives have been lost due to faulty software

Software Issues

“Normal” users

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

Attackers

- Actively look for bugs and flaws
- Like bad software…
- …and try to make it misbehave
- Attack systems thru bad software

Complexity

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

System	Lines of code (LOC)
Netscape	17,000,000
Space shuttle	10,000,000
Linux	1,500,000
Windows XP	40,000,000
Boeing 777	7,000,000

- 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/1000 LOC
- **Do the math**
  - Typical computer: 3,000 exe's of 10K each
  - Conservative estimate of 50 bugs/exe
  - About 3K X 50 = 150K bugs per computer
  - 30,000 node network has 4.5 billion bugs
  - Suppose that only 10% of bugs security-critical and only 10% of those remotely exploitable
  - Then "only" 45 million critical security flaws!

Software Security Topics

- Program flaws (unintentional)
  - Buffer overflow
  - Incomplete mediation
  - Race conditions

- Malicious software (intentional)
  - Viruses
  - Worms
  - 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

Error → **Fault** → **Failure**

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 insure that a program does what is intended
- **Secure software engineering** requires that **the software does what is intended…**
  …and nothing more
- Absolutely secure software is impossible
  - Absolute security is almost never possible!

Program Flaws

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

Buffer Overflow

Buffer Overflow: Typical Attack Scenario

1. Users enter data into a Web form
2. Web form is sent to server
3. Server writes data to buffer, without checking length of input data
4. Data overflows from buffer

- Sometimes, overflow can enable an attack
- Web form attack could be carried out by anyone with an Internet connection

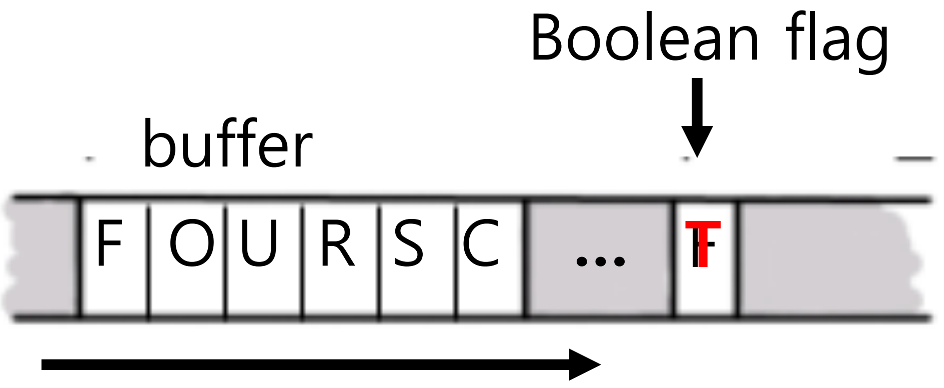
Buffer Overflow: Typical Attack Scenario

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

- **Q:** What happens when this 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

Simple Buffer Overflow

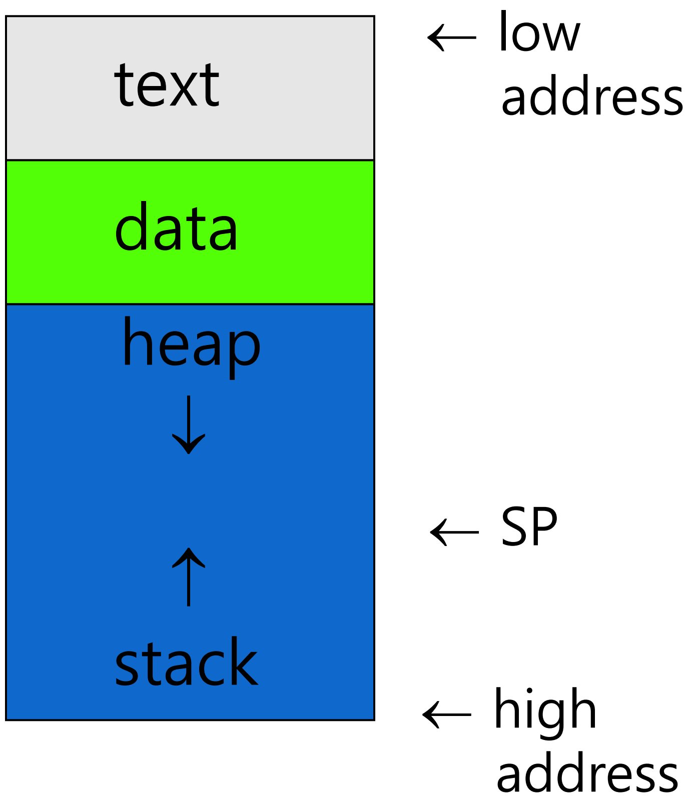
- Consider boolean flag for authentication
- Buffer overflow could overwrite flag allowing anyone to authenticate!



- In some cases, attacker need not be so lucky as to have overflow overwrite flag

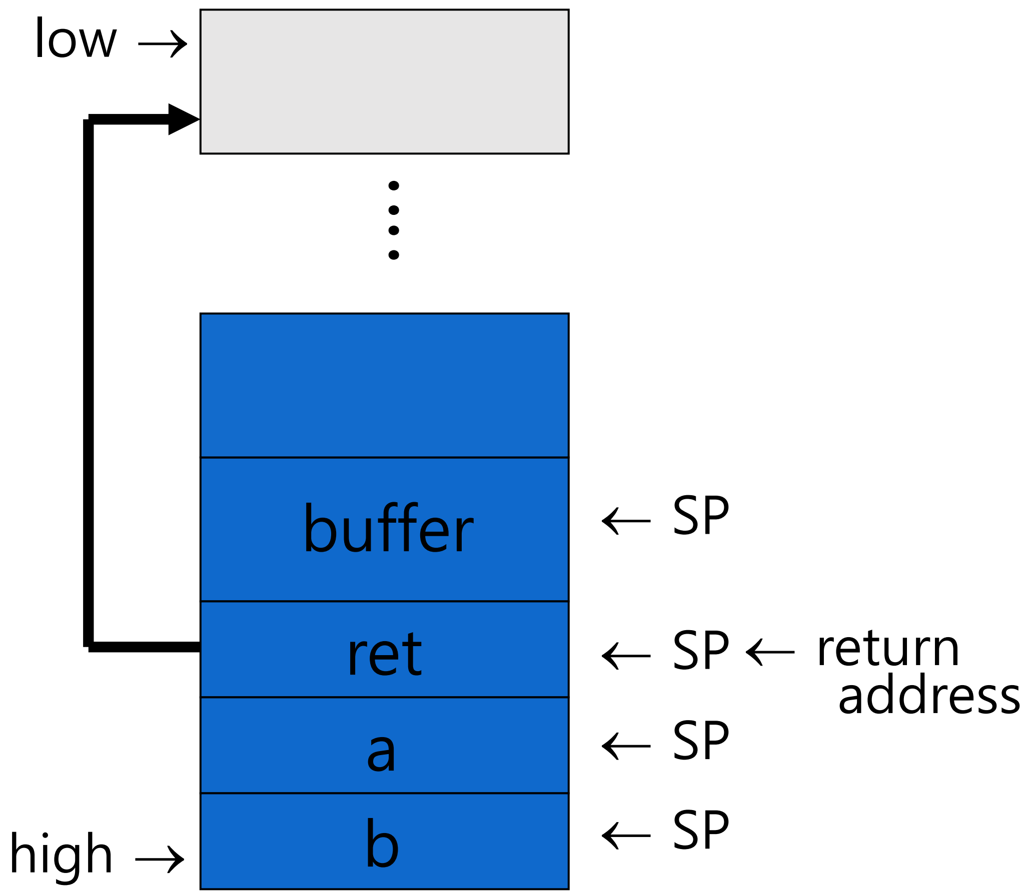
Memory Organization

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



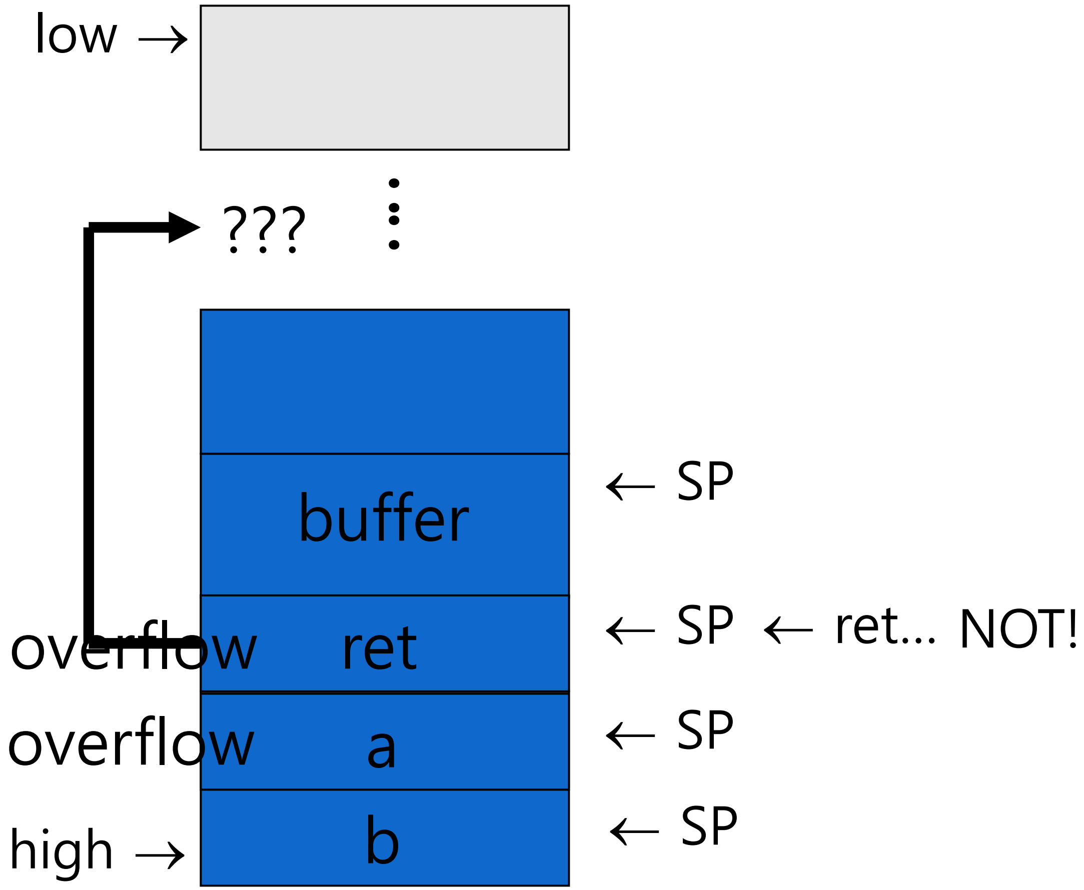
Simplified Stack Example

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



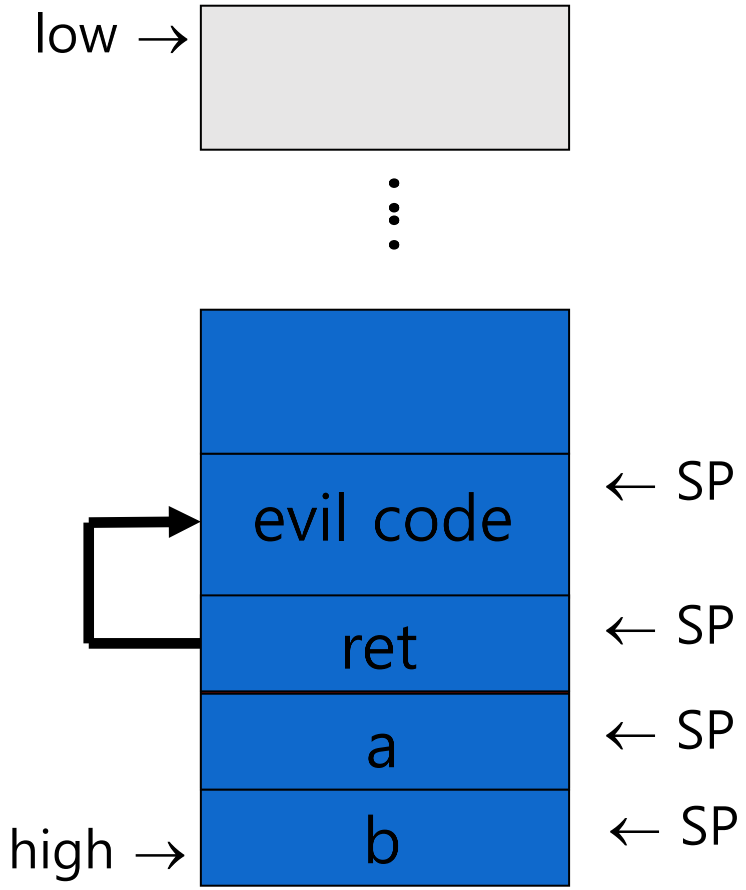
Smashing the Stack

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



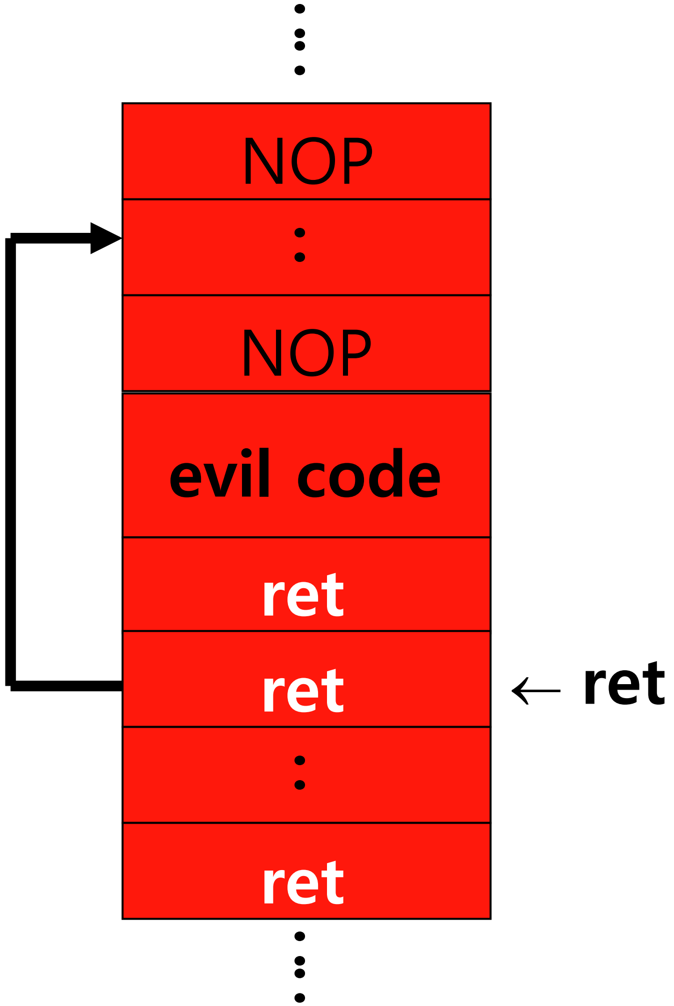
Smashing the Stack

- Attacker has a better idea…
- Code injection
- Attacker can run any code on affected system!



Smashing the Stack

- Attacker may not know
  - Address of evil code
  - Location of **ret** on stack
- Solutions
  - Precede evil code with NOP "**landing pad**"
  - Insert lots of new **ret**

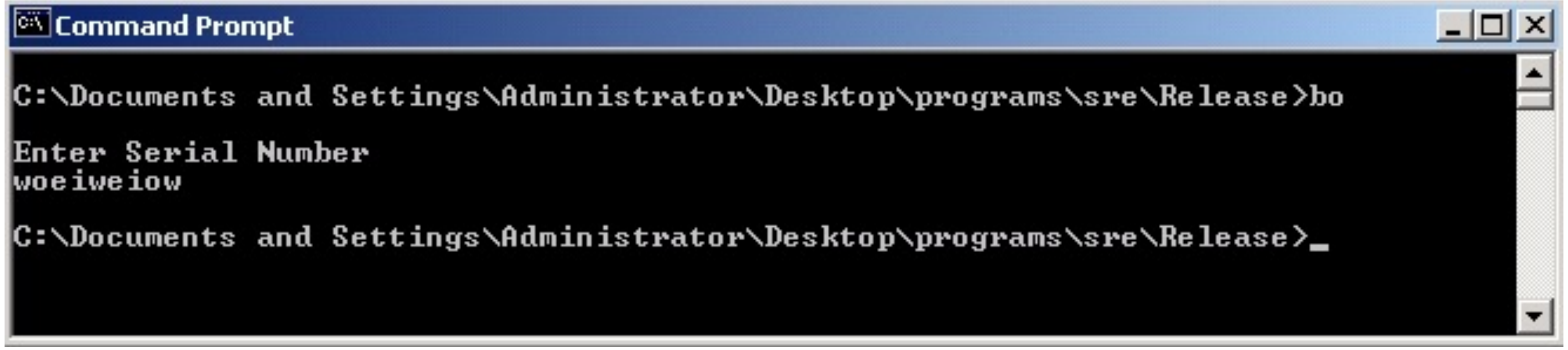


Stack Smashing Summary

- A buffer overflow must exist in the code
- Not all buffer overflows are exploitable
  - Things must line up correctly
- If exploitable, attacker can **inject code**
- Trial and error likely required
  - Lots of help available online
  - Reference: Smashing the Stack for Fun and Profit, Aleph One
- Also possible to overflow the heap

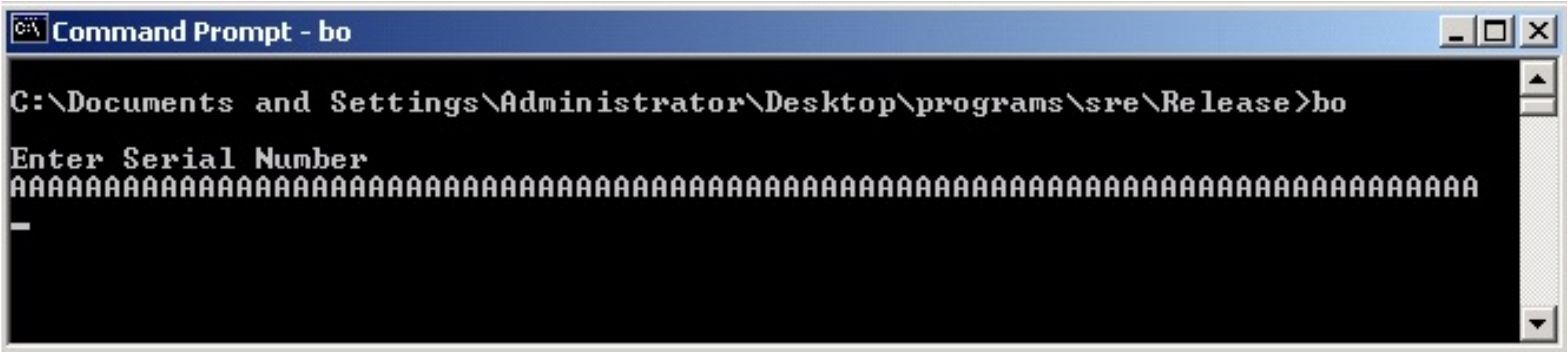
Stack Smashing Example

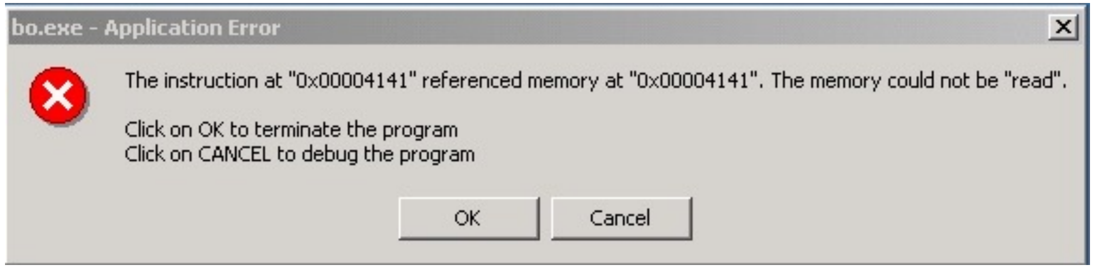
- Program asks for a serial number that the attacker does not know
- Attacker also does not have source code
- Attacker does have the executable (exe)
- Program quits on incorrect serial number



Stack Smashing Example

- By trial and error, attacker discovers an apparent buffer overflow
- Note that 0x41 is "A"
- Looks like ret overwritten by 2 bytes!





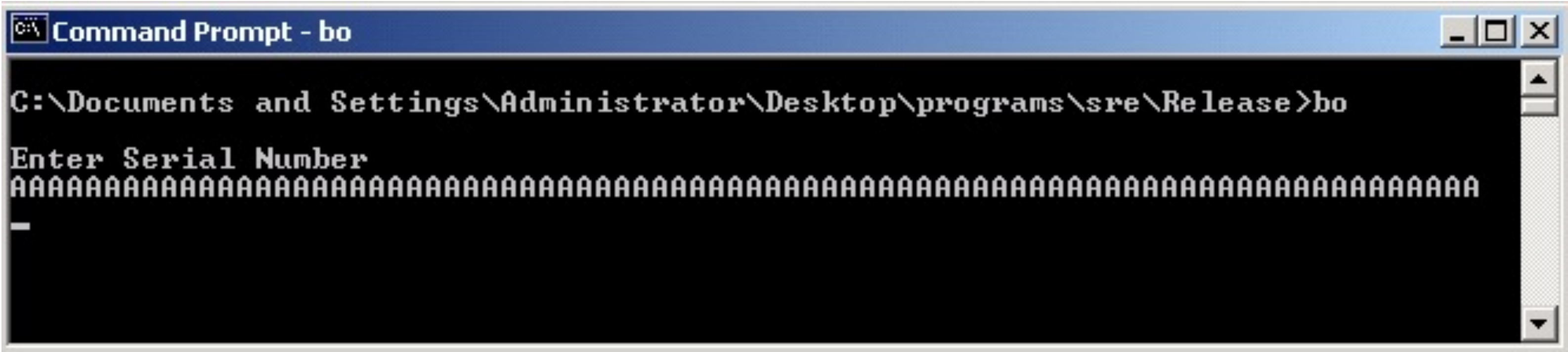
Stack Smashing Example

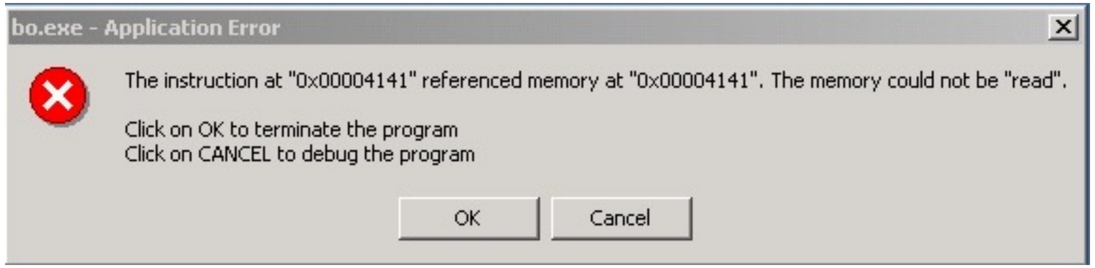
- Next, disassemble bo.exe to find
- The goal is to exploit buffer overflow to jump to address 0x401034

```
.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
```

Stack Smashing Example

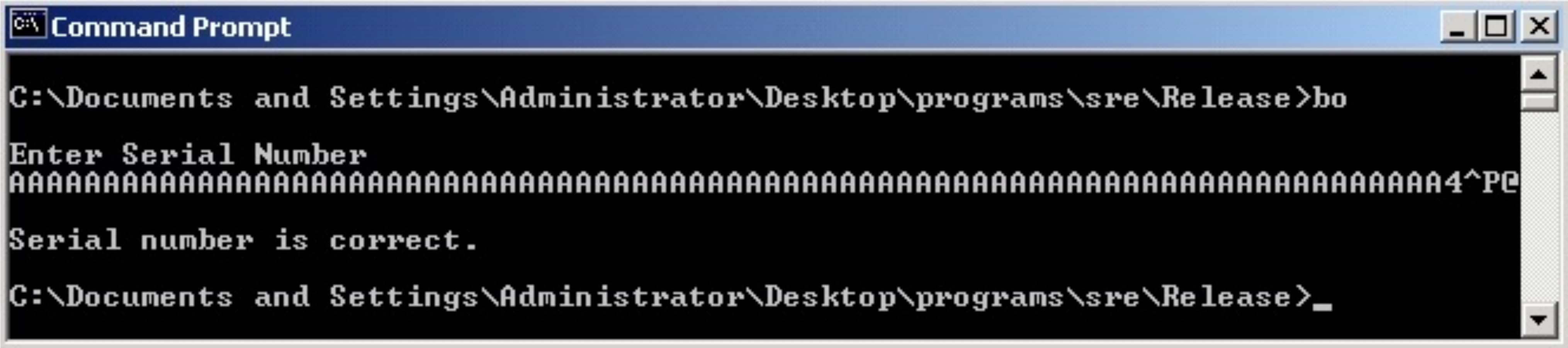
- Find that 0x401034 is “@^P4” in ASCII
- Byte order is reversed? Why?
- X86 processors are “little-endian”





Stack Smashing Example

- Reverse the byte order to "4^P@" and…
- Success! We've bypassed serial number check by exploiting a buffer overflow
- Overwrote the return address on the stack



Stack Smashing Example

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

Stack Smashing Example

- Source code of the buffer overflow

- Flaw easily found by attacker
- Even without the source code!

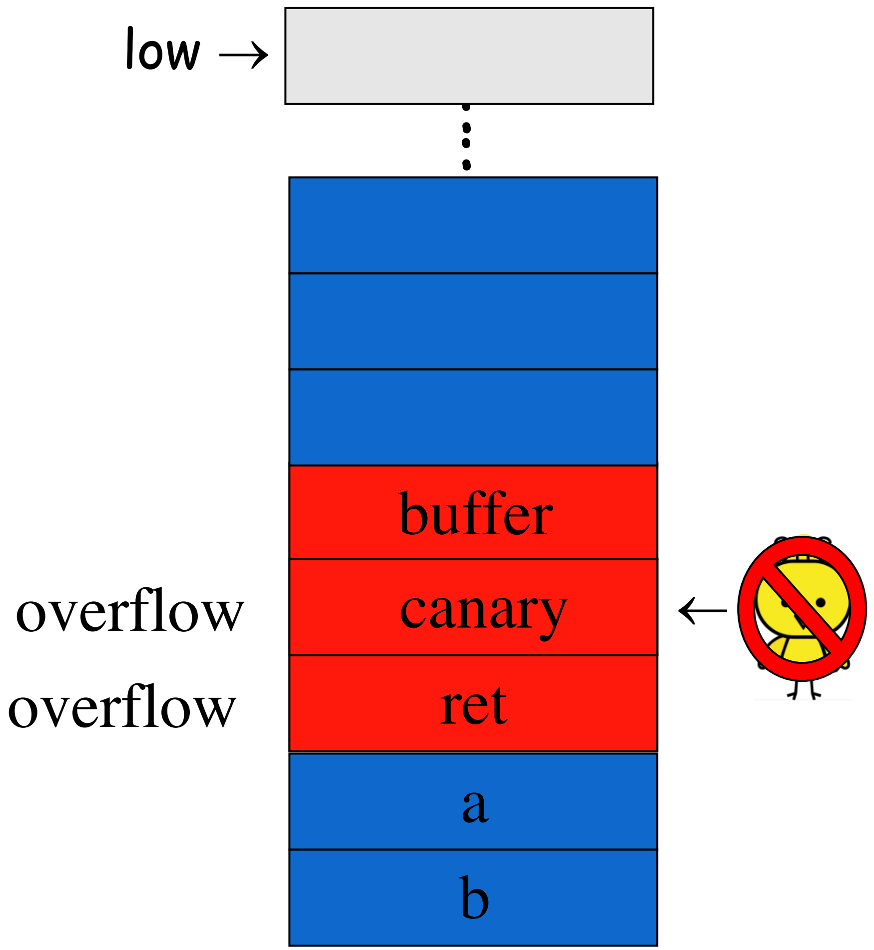
```c
#include <stdio.h>
#include <string.h>

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 Prevention

- 1st choice: employ **non-executable stack**
  - "No execute" **NX bit** (if available)
- 2nd choice: use **safe languages** (Java, C#)
- 3rd choice: use **safer C functions**
  - For unsafe functions, there are safer versions
  - For example, strncpy instead of strcpy

Stack Smashing Prevention

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



Microsoft's Canary

- Microsoft added **buffer security check** feature to C++ with /GS compiler flag
- Uses canary (or "security cookie")
- **Q:** What to do when canary dies?
- **A:** Check for user-supplied handler
- Handler may be subject to attack
  - Claimed that attacker can specify handler code
  - If so, formerly safe buffer overflows become exploitable when /GS is used!

Buffer Overflow

- The “attack of the decade” for 90’s
- Will be the attack of the decade for 00’s
- Can be prevented
  - Use safe languages/safe functions
  - Educate developers, use tools, etc.
- Buffer overflows will exist for a long time
  - Legacy code
  - Bad software development

Incomplete Mediation

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
- Tools exist to help find such problems
  - But incomplete mediation errors can be subtle
  - And tools useful to attackers too!

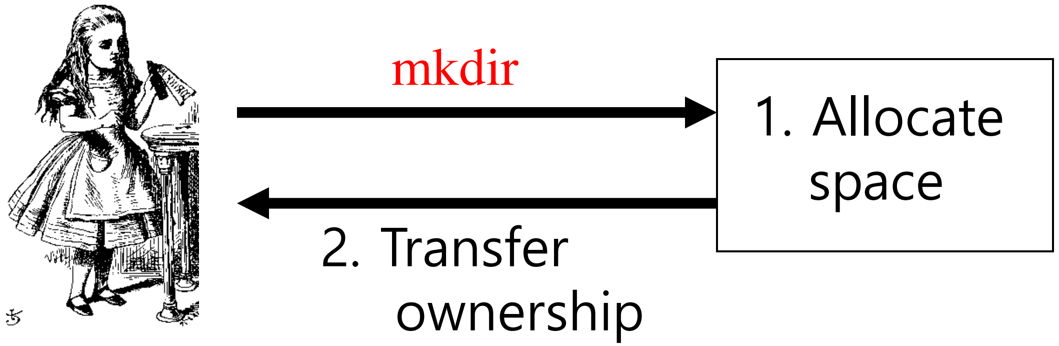
Race Conditions

Race Condition

- Security processes should be **atomic**
- Race conditions can arise when security-critical process occurs in stages
- Attacker makes change between stages
  - Often, between stage that gives authorization, but before stage that transfers ownership

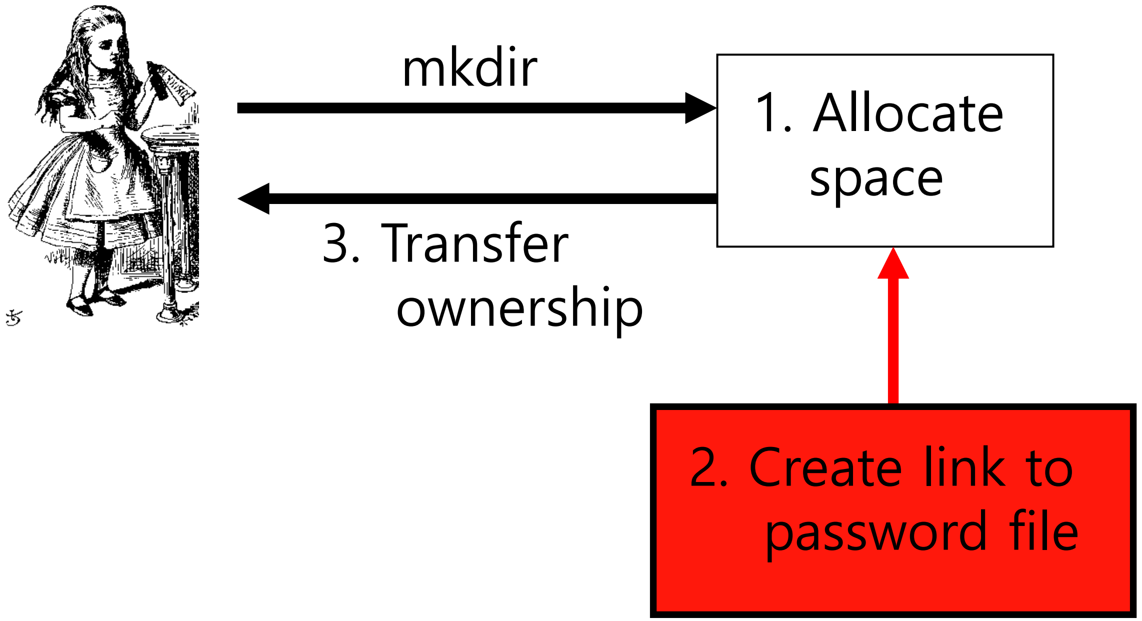
mkdir Race Condition

- mkdir creates new directory
- How mkdir is supposed to work



mkdir Attack

- The mkdir **race condition**
- Not really a “race”
  - But attacker’s timing is critical



Race Conditions

- Race conditions are common
- Race conditions may be more prevalent than buffer overflows
- But race conditions harder to exploit
- To prevent race conditions, make security-critical processes atomic
  - Occur all at once, not in stages
  - Not always easy to accomplish in practice

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 can implemented by virus, warm …

Malicious Software

- Preliminary work by Cohen (early 80’s)
- Brain virus (1986)
- Morris worm (1988)
- Code Red worm (2001)
- SQL Slammer worm (2004)
- Future of malware?

Where do viruses live?

- Boot sector
  - Take control before anything else
- Memory resident
  - Stays in memory – Rebooting system can remove the virus out
- Applications, macros, data, etc.
- Library routines
- Compilers, debuggers, virus checker, etc.
  - These are particularly nasty!

Brain virus

- 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 malicious

Morris Worm – 1/5

- First appeared in 1988
- What it tried to do
  - Determine where it could spread
  - Spread its infection
  - Remain undiscovered
- Morris claimed it was a test gone bad
- "Flaw" in worm code — it tried to re-infect already-infected systems
  - Led to resource exhaustion
  - Adverse effect was like a so-called rabbit

Morris Worm – 2/5

How to spread its infection?

- Tried to obtain access to machine by
  - User account password guessing
  - Exploited buffer overflow in fingerd
  - Exploited trapdoor in sendmail
- Flaws in fingerd and sendmail were well-known at the time, but not widely patched

Morris Worm – 3/5

① Once access had been obtained to machine

② "Bootstrap loader" sent to victim

- Consisted of 99 lines of C code

③ Victim machine compiled and executed code

④ Bootstrap loader then fetched the rest of the worm

⑤ Victim even **authenticated** the sender!

Morris Worm – 4/5

How to remain undetected?

- If transmission of the worm was interrupted, all code was deleted
- Code was encrypted when downloaded
- Downloaded code deleted after decrypting and compiling
- When running, the worm regularly changed its name and process identifier (PID)

Result of Morris Worm – 5/5

- Shocked the Internet community of 1988
- Internet designed to withstand nuclear war
  - Yet it was brought down by a graduate student!
  - At the time, Morris' father worked at NSA…
- Could have been much worse — not malicious
  - Users who did not panic recovered quickest
- CERT began, increased security awareness
  - Though limited actions to improve security

Code Red Worm – 1/2

- Appeared in July 2001
- Infected more than **250,000 systems in about 10 ~ 15 hours**
- In total, infected 750,000 out of 6,000,000 susceptible systems
- To gain access to a system, exploited buffer overflow in Microsoft IIS server software
- Then monitored traffic on port 80 looking for other susceptible servers

Code Red Worm – 2/2

- What it did
  - Day 1 to 19 of month: tried to spread infection
  - Day 20 to 27: distributed denial of service (DDOS) attack on `www.whitehouse.gov`
- Later versions (several variants)
  - Included trapdoor for remote access
  - Rebooted to flush worm, leaving only trapdoor
- Has been claimed that Code Red may have been "beta test for information warfare"

SQL Slammer worm – 1/2

- Infected **250,000 systems in 10 minutes!**
- Code Red took 15 hours to do what Slammer did in 10 minutes
- At its peak, Slammer infections doubled every 8.5 seconds
- Slammer spread too fast
- “Burned out” available bandwidth

SQL Slammer worm – 2/2

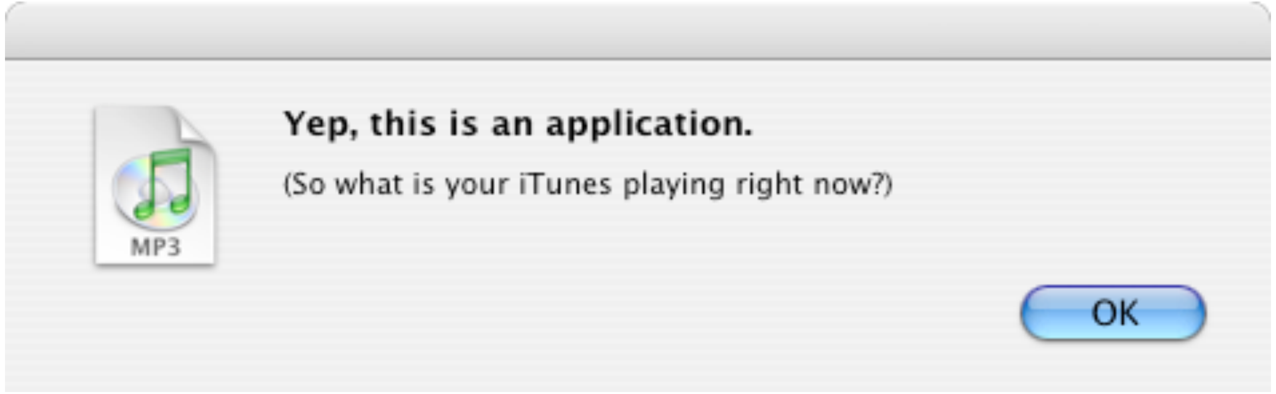
- Why was Slammer so successful?
  - Worm fit in **one 376 byte UDP packet**
  - Firewalls monitor the connection
    - Expectation was that much more data would be required for an attack
  - Firewalls often let small packet thru, assuming it could do no harm by itself
- Slammer defied assumptions of “experts”

Trojan Horse – 1/3

- A trojan has unexpected function
- Prototype of 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…

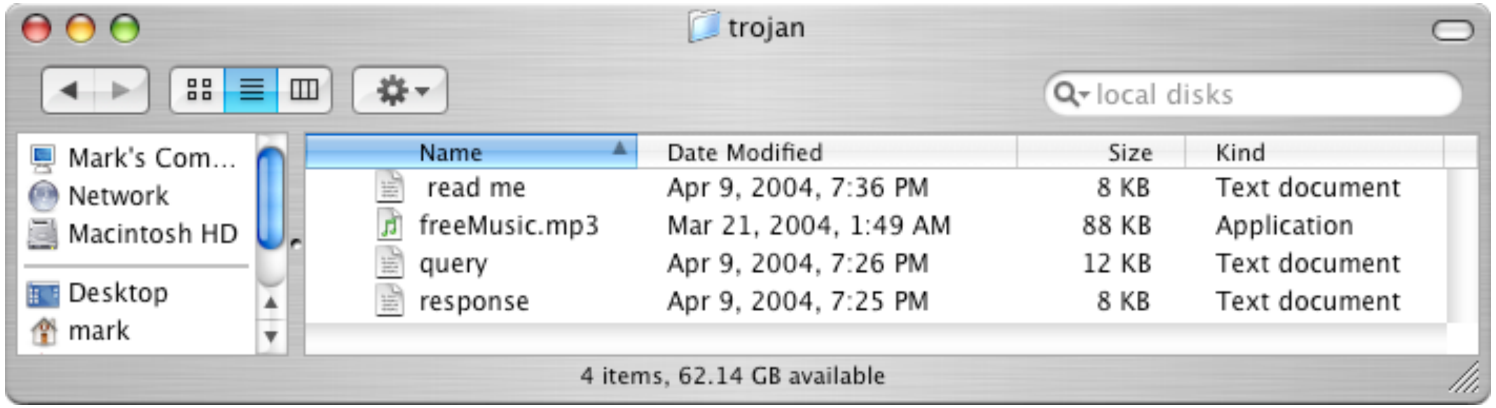
Trojan Horse – 2/3

- Double click on freeMusic.mp3
  - iTunes opens (expected)
  - “Wild Laugh” (probably not expected)
  - Message box (unexpected)
- A wolf in sheep’s clothing



Trojan Horse – 3/3

- How does freeMusic.mp3 trojan work?
- This "mp3" is an application, not data!
- This trojan is harmless,
- but… trojan could have done anything user can do delete files, download files, launch apps, etc.



Malware Detection

Malware Detection

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

Signature Detection - 1/2

- A **signature** is a string of bits found in software (or could be a hash value)
- Suppose that a virus has signature 0x23956a58bd910345
- We can search for this signature in all files
- If we find the signature, are we sure we've found the virus?
  - No, same signature could appear in other files
  - But at random, chance is very small: $1/2^{64}$
  - Software is not random, so probability is higher

Signature Detection - 2/2

- Advantages
  - Effective on "traditional" malware
  - Minimal burden for users/administrators
- Disadvantages
  - Signature file can be large (10,000's)…
  - …making scanning slow
  - Signature files must be kept up to date
  - Cannot detect unknown viruses
  - Cannot detect some new types of malware
- By far the most popular detection method!

Change Detection - 1/2

- Viruses must live somewhere on system
- If we detect that a file has changed, it may be infected
- How to detect changes?
  - Hash files and (securely) store hash values
  - Recompute hashes and compare
  - If hash value changes, file **might** be infected

Change Detection - 2/2

- 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 still need signature-based system

Anomaly Detection - 1/2

- Monitor system for anything "unusual" or "virus-like" or potentially malicious
- What is unusual?
  - Files change in some unusual way
  - System misbehaves in some way
  - Unusual network activity
  - Unusual file access, etc., etc.
- But must first define "normal"
  - And normal can change!

Anomaly Detection - 2/2

- Advantages
  - Chance of detecting unknown malware
- Disadvantages
  - Unproven in practice
  - Attacker can make anomaly look normal
  - Must be combined with another method (such as signature detection)
- Also popular in intrusion detection (IDS)
- A difficult unsolved (unsolvable?) problem!
  - AI?

Future of Malware

Future of Malware

- Polymorphic malware and
- metamorphic malware
- Fast replication/Warhol worms
- Flash worms, Slow worms, etc.
- Future is bright for malware
  - Good news for the bad guys…
  - …bad news for the good guys
- Future of malware detection?

Polymorphic Malware - 1/2

- The first responses of virus writers of signature detection success
- Polymorphic worm (usually) encrypted
- New key is used each time worm propagates
    - The purpose of encryption is for masking
    - The encryption is weak (repeated XOR)
    - Worm body has no fixed signature
    - Worm must include code to decrypt itself
    - Signature detection searches for decrypt code

Polymorphic Malware - 2/2

- Detectable by signature-based method
  - Though more challenging than non-polymorphic…

Metamorphic Malware – 1/2

- A step further than polymorphic malware
- A metamorphic worm mutates before infecting a new system
- Such a worm can avoid signature-based detection systems
- The mutated worm must do the same thing as the original
- And it must be “different enough” to avoid detection
- Detection is currently unsolved problem

Metamorphic Malware – 2/2

The way to replicate

- To replicate, the worm is disassembled
- Worm is stripped to a base form
- Random variations inserted into code
  - Rearrange jumps
  - Insert dead code
  - Many other possibilities
- Assemble the resulting code
- Result is a worm with same functionality as original, but very different signature

Warhol Worm - 1/2

- "In the future everybody will be world-famous for 15 minutes" — Andy Warhol
- A Warhol Worm is designed to infect the entire Internet in 15 minutes
- Slammer infected 250,000 systems in 10 minutes
  - "Burned out" bandwidth
  - Slammer could **not** have infected all of Internet in 15 minutes — too bandwidth intensive

Warhol Worm - 2/2

One approach to a Warhol worm…

1. Seed worm with an initial **hit list** containing a set of vulnerable IP addresses
   - Depends on the particular exploit
   - Tools exist for finding vulnerable systems
2. Each successful initial infection would attack **selected part of IP address space**

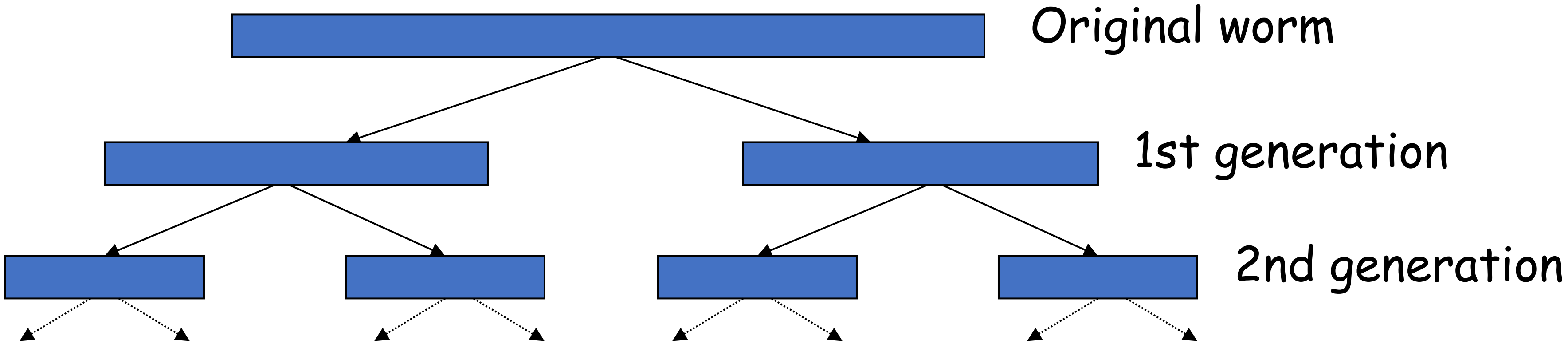
- No worm this sophisticated has yet been seen in the wild (as of 2005)
  - Slammer generated random IP addresses
- Could infect entire Internet in 15 minutes!

Flash Worm – 1/3

- Possible to do "better" than Warhol worm?
- Can entire Internet be attacked in < 15 min?
- **Searching for vulnerable IP addresses is slow part of any worm attack**
- Searching might be bandwidth limited
  - Like Slammer
- A "flash worm" is designed to infect entire Internet almost instantly

Flash Worm – 2/3

- Predetermine **all** vulnerable IP addresses
  - Depends on the particular exploit
- Embed all known vulnerable addresses in worm
- Result is a huge worm (perhaps 400KB)
- Whenever the worm replicates, it splits
- Virtually no wasted time or bandwidth!



Flash Worm – 3/3

- Estimated that ideal flash worm could infect the entire Internet in **15 seconds!**
- Much faster than humans could respond
- A conjectured defense against flash worms
  - Deploy many "personal IDSs"
  - Master IDS watches over the personal IDSs
  - When master IDS detects unusual activity, lets it proceed on a few nodes, blocks it elsewhere
  - If sacrificial nodes adversely affected, attack is prevented almost everywhere

Cyber 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

Miscellaneous Attacks

Miscellaneous Attacks

- Numerous attacks involve software
- We'll discuss a few issues that do not fit in previous categories
  - Salami attack
  - Linearization attack
  - Time bomb
- Can you ever trust software?

Salami Attacks – 1/3

- What is Salami attack?
  - Programmer “slices off” money
  - Slices are hard for victim to detect
- Example
  - Bank calculates interest on accounts
  - Programmer “slices off” any fraction of a cent and puts it in his own account
  - No customer notices missing partial cent
  - Bank may not notice any problem
  - Over time, programmer makes lots of money!

Salami Attacks – 2/3

- Such attacks are possible for insiders
- Do salami attacks actually occur?
- Programmer added a few cents to every employee payroll tax withholding
  - But money credited to programmer's tax
  - Programmer got a big tax refund!
- Rent-a-car franchise in Florida inflated gas tank capacity to overcharge customers

Salami Attacks – 3/3

- Employee reprogrammed Taco Bell cash register: $2.99 item registered as $0.01
  - Employee pocketed $2.98 on each such item
  - A large “slice” of salami!
- In LA four men installed computer chip that overstated amount of gas pumped
  - Customer complained when they had to pay for more gas than tank could hold!
  - Hard to detect since chip programmed to give correct amount when 5 or 10 gallons purchased
  - Inspector usually asked for 5 or 10 gallons!

Linearization Attack – 1/4

- Program checks for serial number S123N456
- For efficiency, check made one character at a time
- Can attacker take advantage of this?

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
    int i;
    char serial[9]="S123N456\n";

    for(i = 0; i < 8; ++i)
    {
        if(argv[1][i] != serial[i]) break;
    }
    if(i == 8)
    {
        printf("\nSerial number is correct!\n\n");
    }
}
```

Linearization Attack – 2/4

- Correct string takes longer than incorrect
- Attacker tries all 1 character strings
  - Finds S takes most time
- Attacker then tries all 2 char strings $S*$
  - Finds $S1$ takes most time
- And so on...
- Attacker is able to recover serial number one character at a time!

Linearization Attack – 3/4

- What is the advantage of attacking serial number one character at a time?
- Suppose serial number is 8 characters and each has 128 possible values
  - Then $128^8 = 2^{56}$ possible serial numbers
  - Attacker would guess the serial number in about 2^{55} tries — a lot of work!
  - Using the linearization attack, the work is about $8*(128/2) = 2^9$ which is trivial!

Linearization Attack – 4/4

- A real-world linearization attack
- TENEX (an ancient timeshare system)
  - Passwords checked one character at a time
  - Careful timing was not necessary, instead…
  - …could arrange for a "page fault" when next unknown character guessed correctly
  - The page fault register was user accessible
  - Attack was very easy in practice

Time Bomb

- In 1986 Donald Gene Burleson told employer to stop withholding taxes from his paycheck
- His company refused
- He planned to sue his company
  - He used company computer to prepare legal docs
  - Company found out and fired him
- Burleson had been working on a malware…
- After being fired, his software “time bomb” deleted important company data

Time Bomb

- Company was reluctant to pursue the case
- So Burleson sued company for back pay!
  - Then company finally sued Burleson
- In 1988 Burleson fined $11,800
  - Took years to prosecute
  - Cost thousands of dollars to prosecute
  - Resulted in a slap on the wrist
- One of the first computer crime cases
- Many cases since follow a similar pattern
  - Companies often reluctant to prosecute

Trusting Software – 1/2

- Can you ever trust software?
  - See Reflections on Trusting Trust
- Consider the following thought experiment
- Suppose C compiler has a virus
  - When compiling login program, virus creates backdoor (account with known password)
  - When recompiling the C compiler, virus incorporates itself into new C compiler
- Difficult to get rid of this virus!

Trusting Software – 2/2

- Suppose you notice something is wrong
- So you start over from scratch
- First, you recompile the C compiler
- Then you recompile the OS
  - Including login program…
  - You have not gotten rid of the problem!
- In the real world
  - Attackers try to hide viruses in virus scanner
  - Imagine damage that would be done by attack on virus signature updates

Q & A

aiclasshongik@gmail.com