Software and Security

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 100K each
  - Conservative estimate of 50 bugs/exe
  - About 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" 4.5 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

$$\begin{tikzcd} \textbf{error} \arrow[r] & \textbf{fault} \arrow[r] & \textbf{failure} \end{tikzcd}$$

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

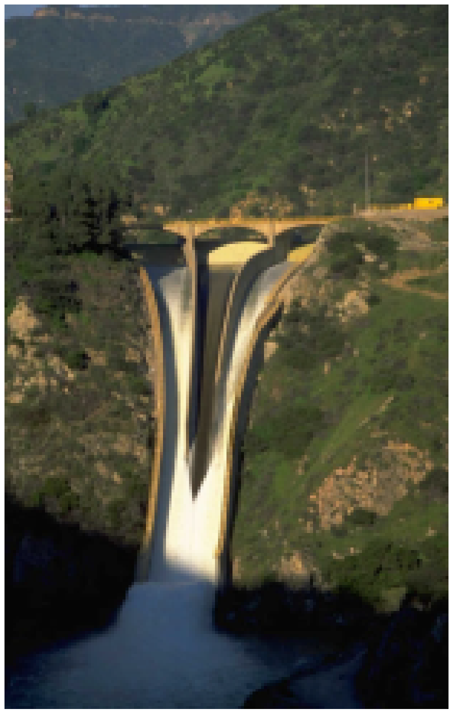
Secure Software

- 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!
- How can we manage the risks?

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



Typical Attack Scenario

- Users enter data into a Web form
- Web form is sent to server
- Server writes data to buffer, without checking length of input data
- Data overflows from buffer
- Sometimes, overflow can enable an attack
- Web form attack could be carried out by anyone with an Internet connection

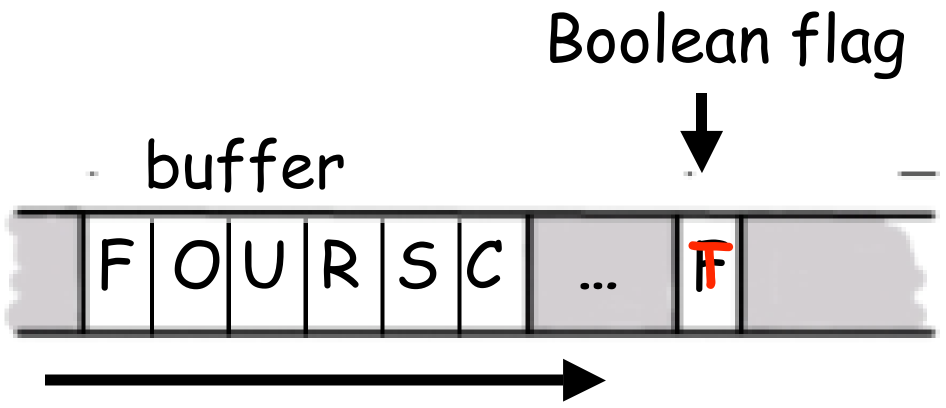
Buffer Overflow

```
    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

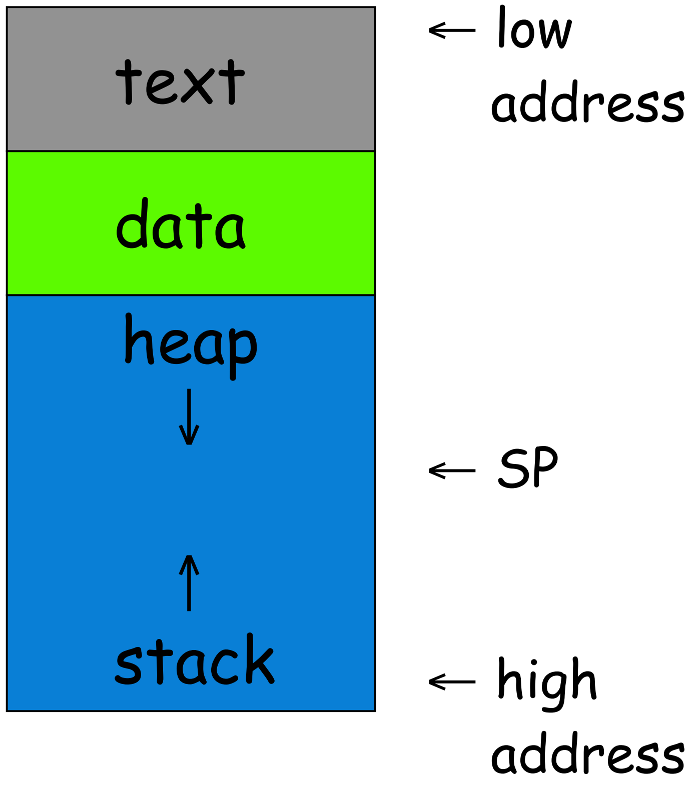
- Consider boolean flag for authentication
- Buffer overflow could overwrite flag allowing anyone to authenticate!



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

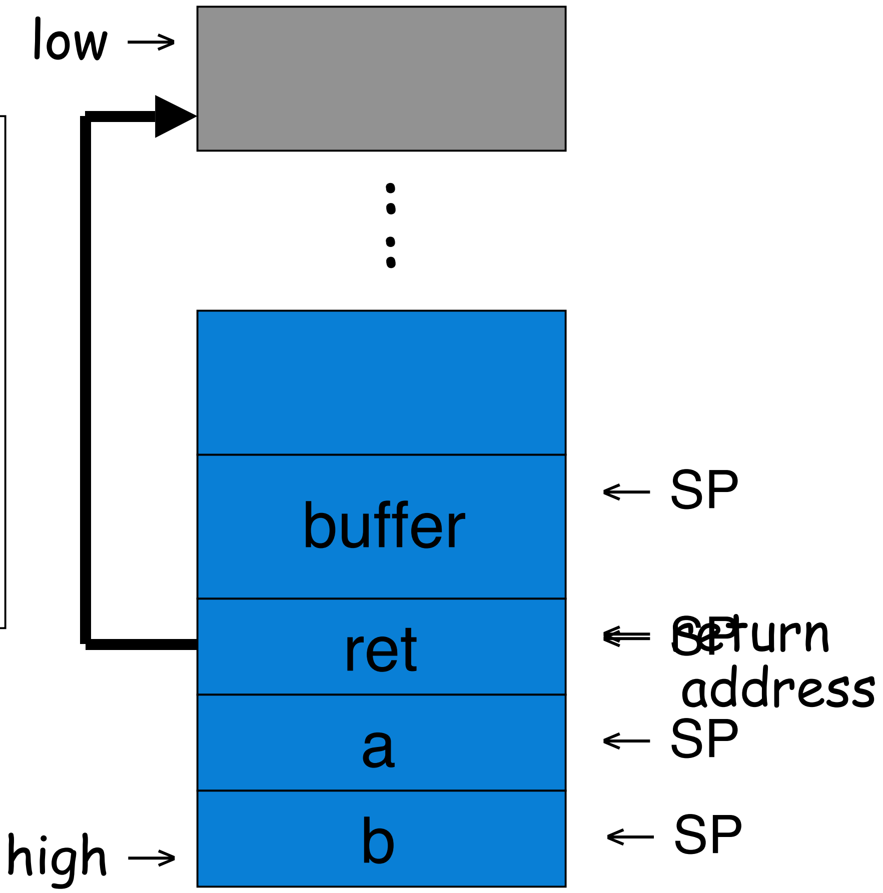
Memory Organization

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



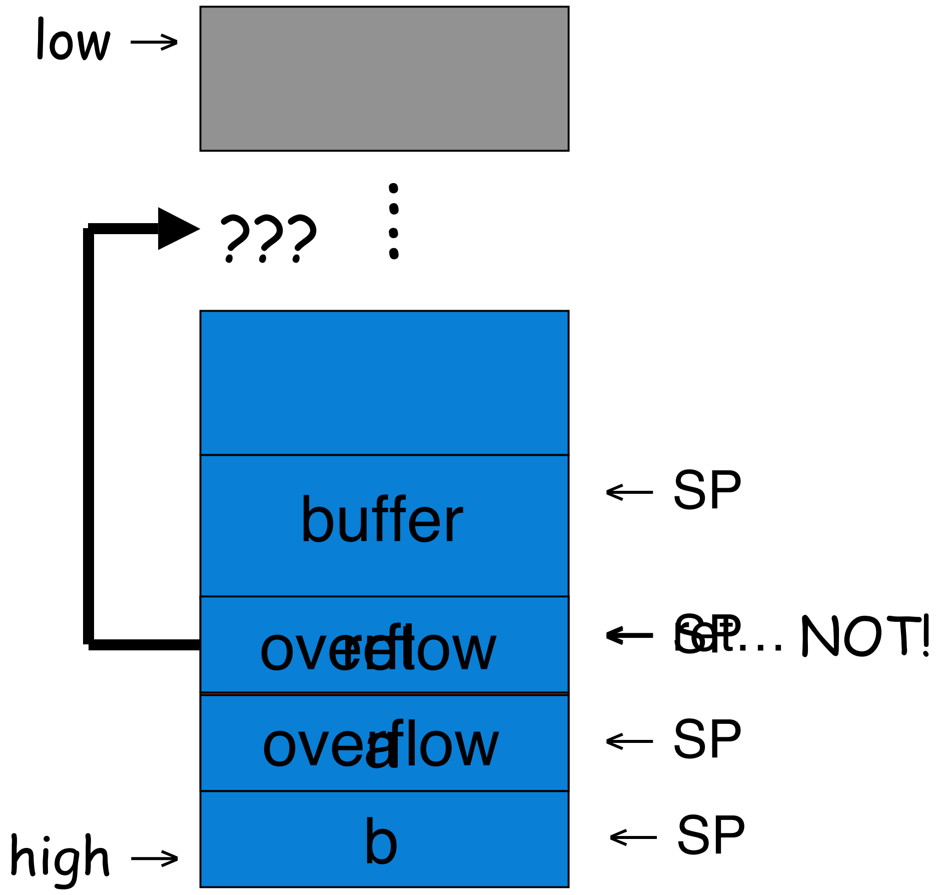
Simplified Stack Example

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



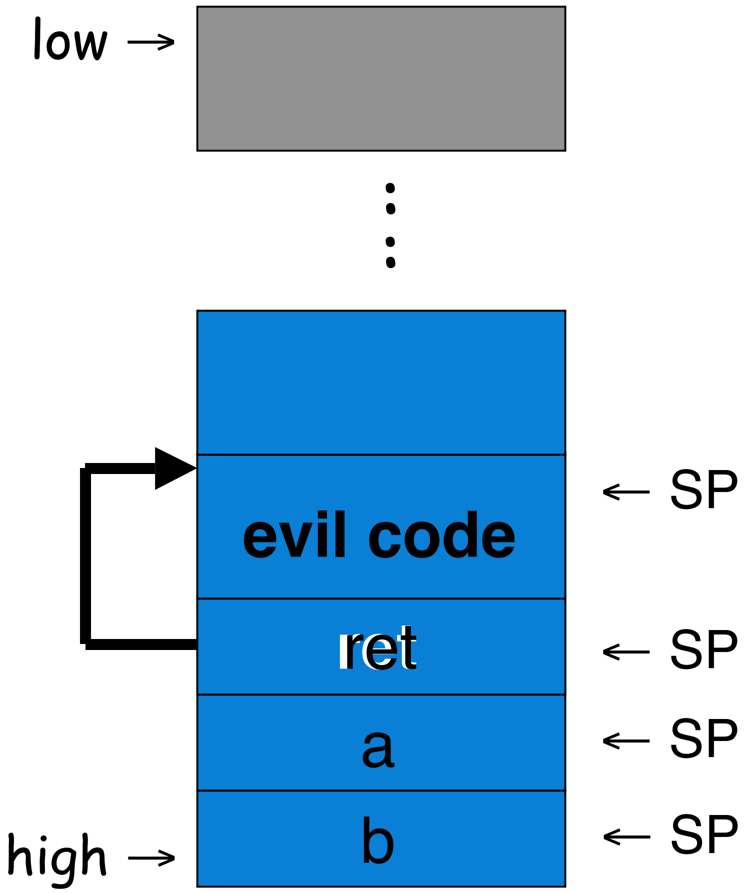
Smashing the Stack

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



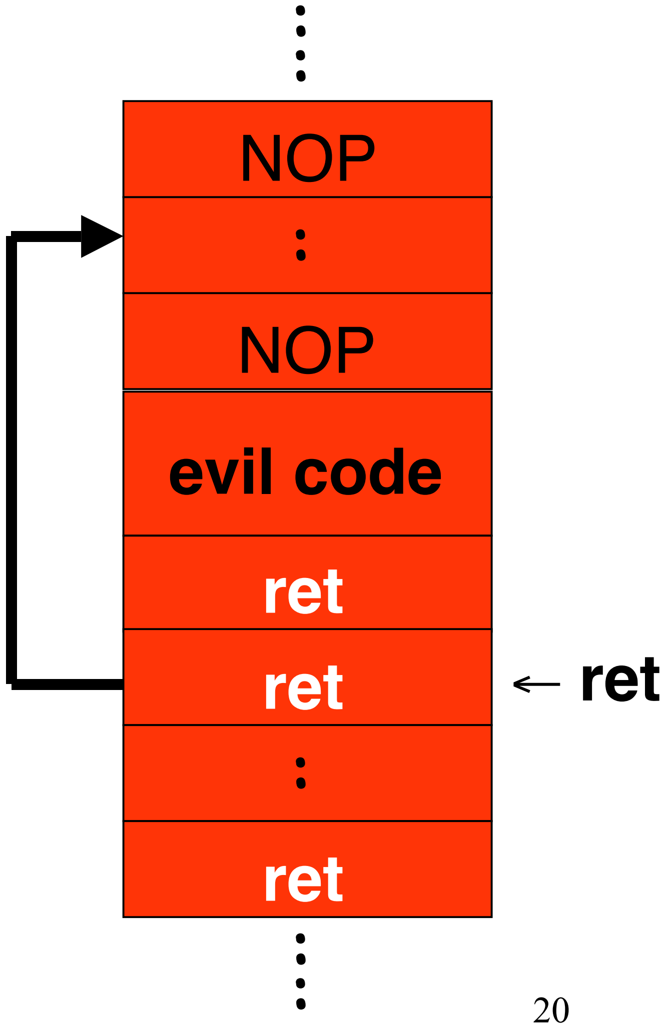
Smashing the Stack

- 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**

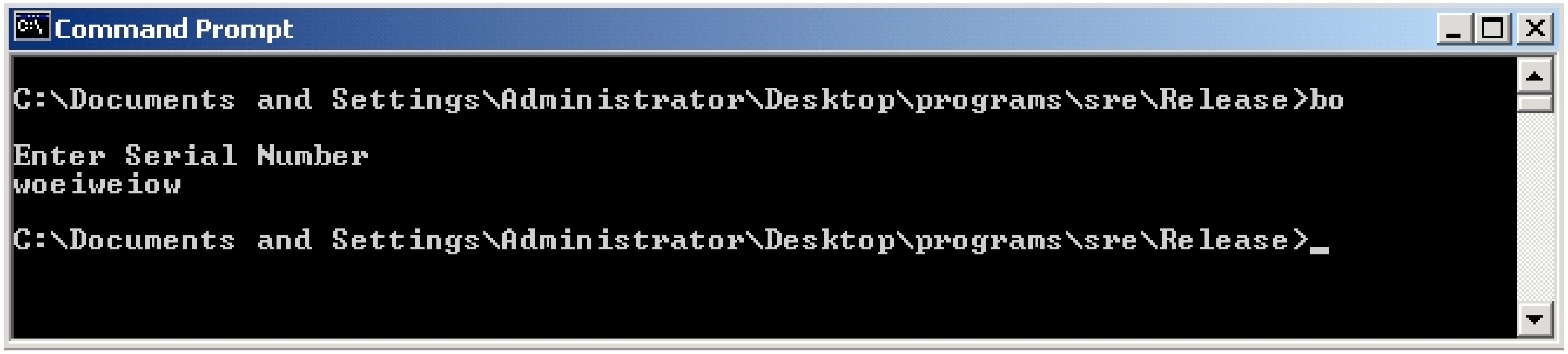


Stack Smashing Summary

- 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
  - Smashing the Stack for Fun and Profit, Aleph One
- Also possible to overflow the heap
- Stack smashing is "attack of the decade"

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)

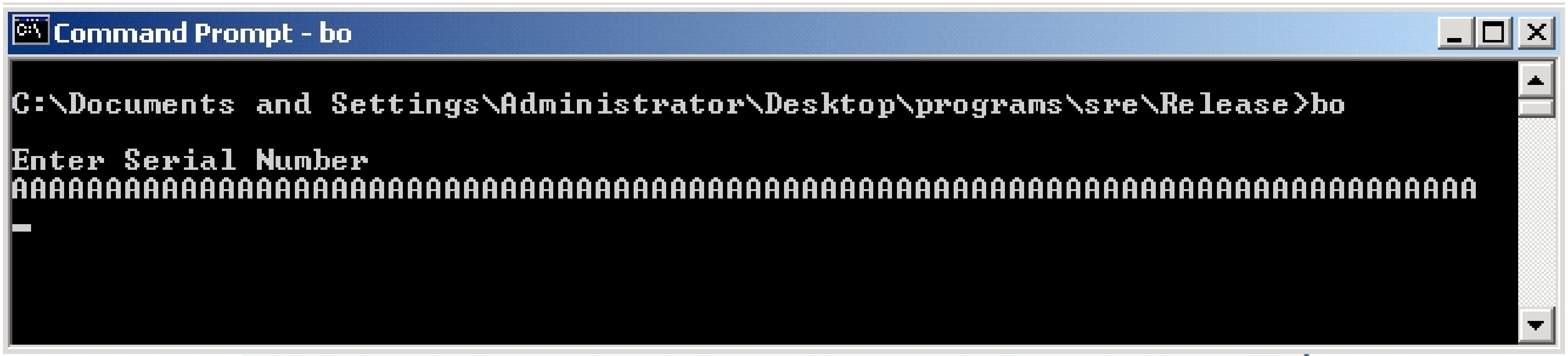


```
C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>bo

Enter Serial Number
woeiweiow

C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>_
```

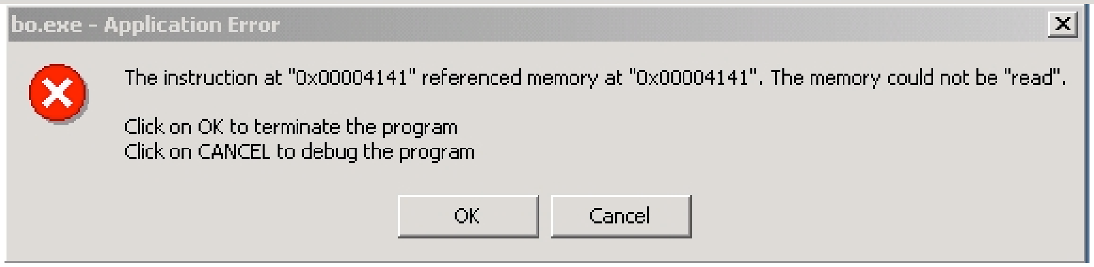
- Program quits on incorrect serial number

Example

- By trial and error, attacker discovers an apparent buffer overflow



```
C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>bo

Enter Serial Number
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```



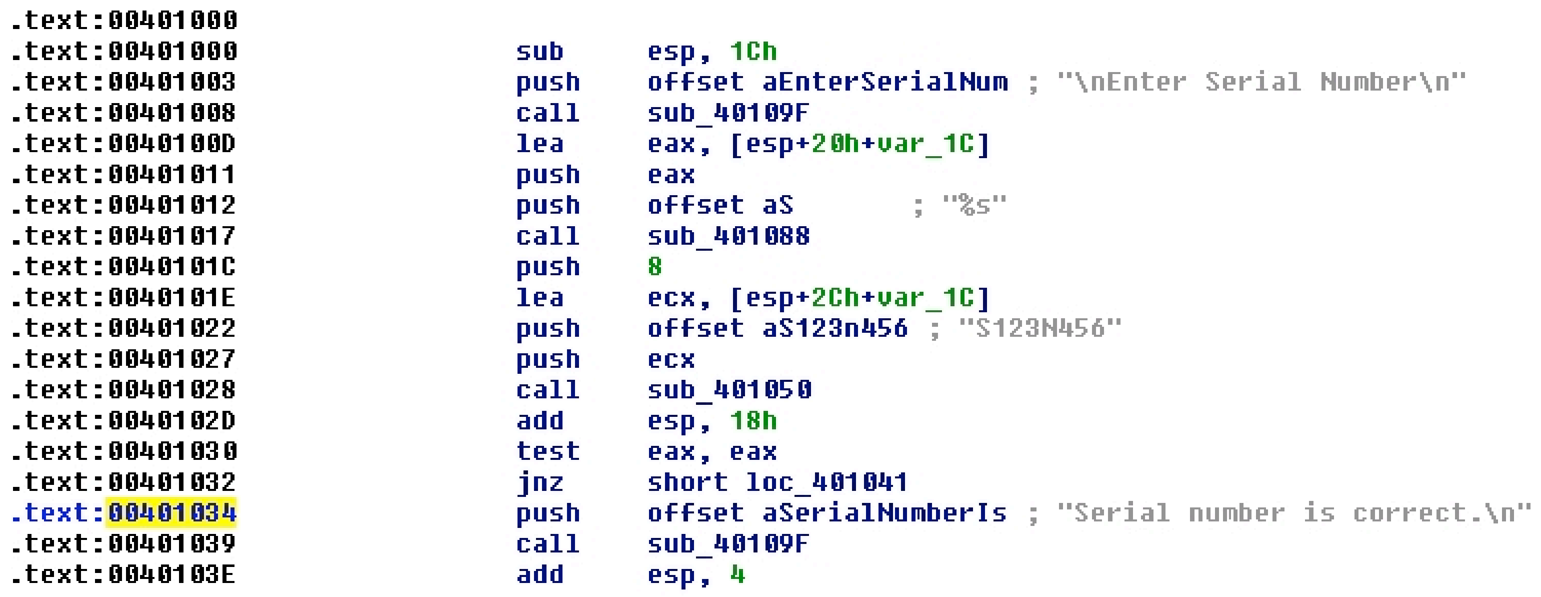
The instruction at "0x00004141" referenced memory at "0x00004141". The memory could not be "read".

Click on OK to terminate the program
Click on CANCEL to debug the program

- Note that 0x41 is "A"
- Looks like **ret** overwritten by 2 bytes!

Example

- Next, disassemble bo.exe to find

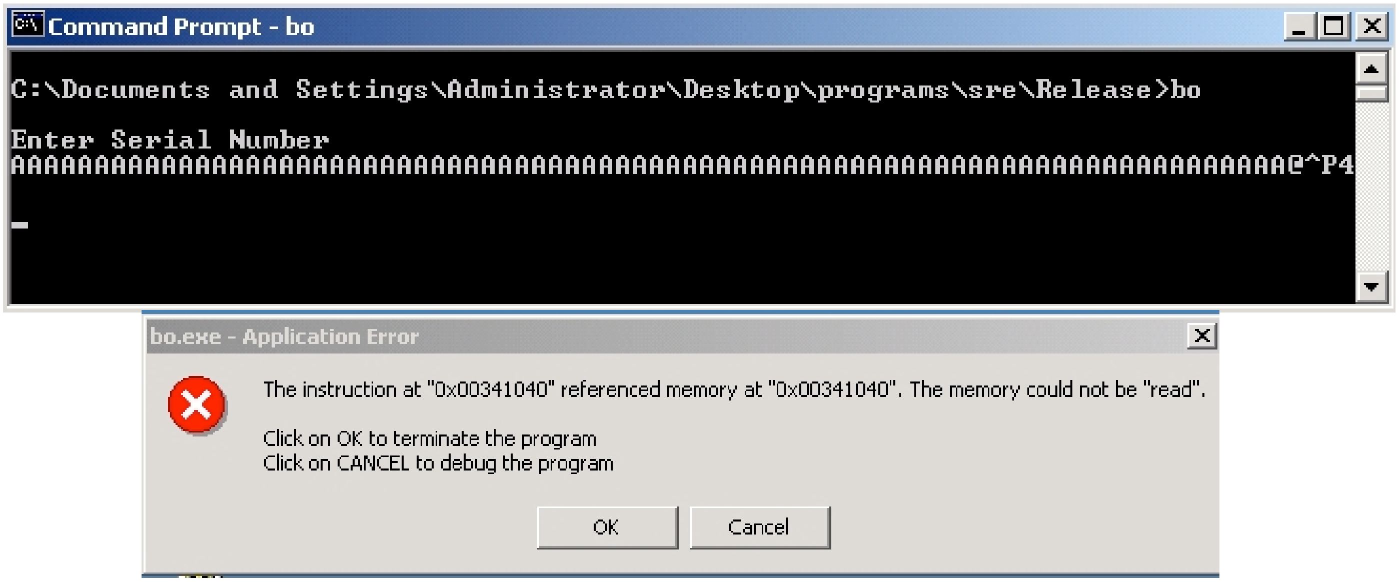


```
.text:00401000
.text:00401000                 sub     esp, 1Ch
.text:00401003                 push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008                 call    sub_40109F
.text:0040100D                 lea     eax, [esp+20h+var_1C]
.text:00401011                 push    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

Example

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



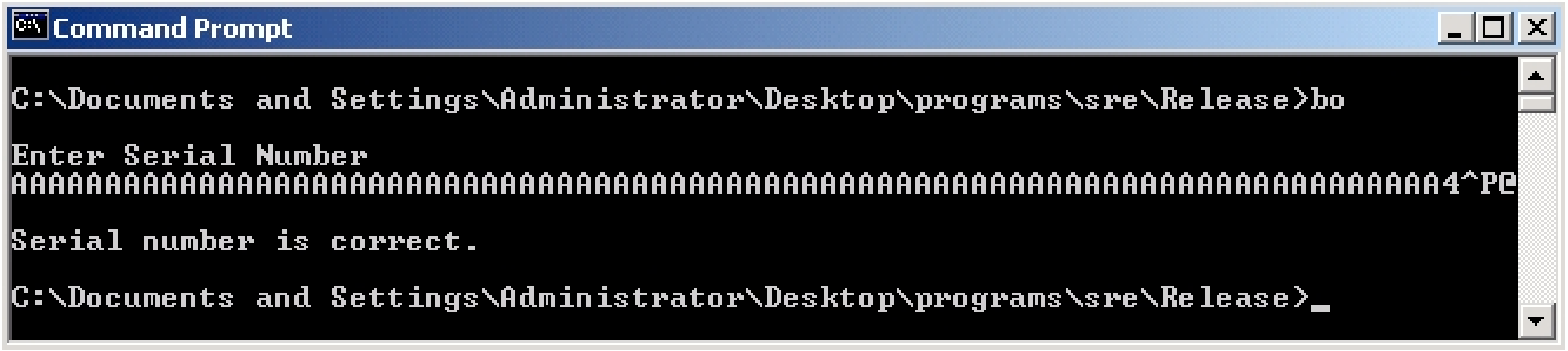
```
C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>bo

Enter Serial Number
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@^P4
```

```
The instruction at "0x00341040" referenced memory at "0x00341040". The memory could not be "read".

Click on OK to terminate the program
Click on CANCEL to debug the program
```

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

Example

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



```
C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>bo

Enter Serial Number
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA4^P@

Serial number is correct.

C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>_
```

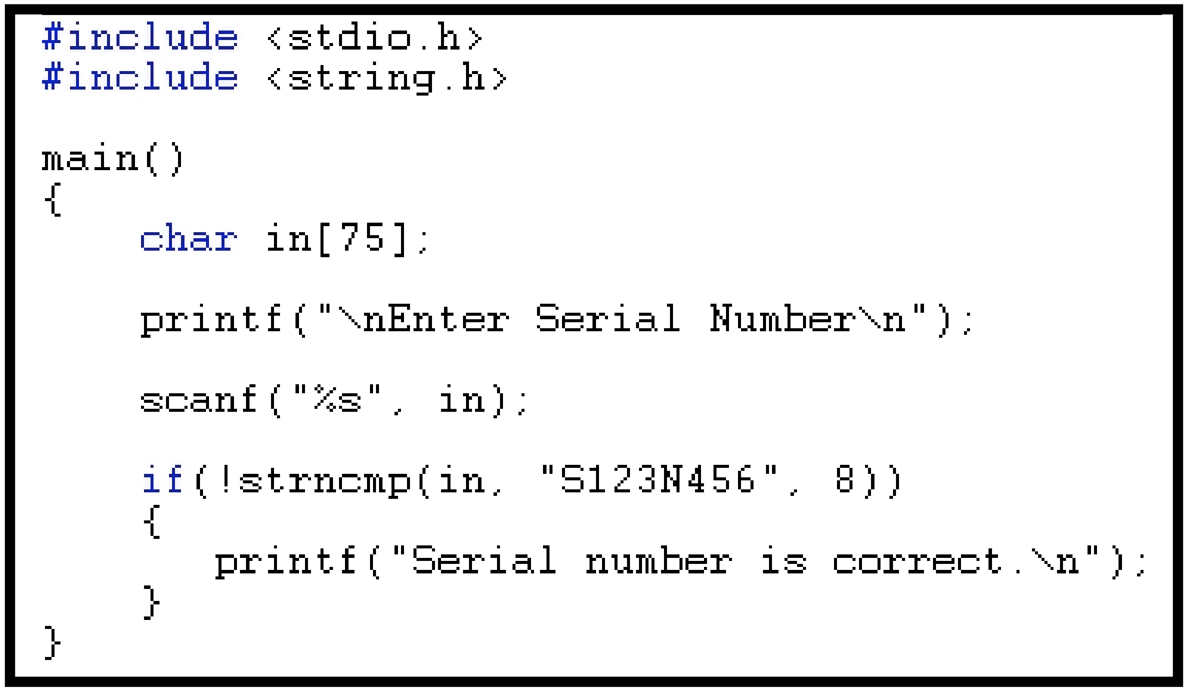
- Success! We've bypassed serial number check by exploiting a buffer overflow
- Overwrote the return address on the stack

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

Example

- Source code of the buffer overflow
- Flaw easily found by attacker
- **Even without the source code!**



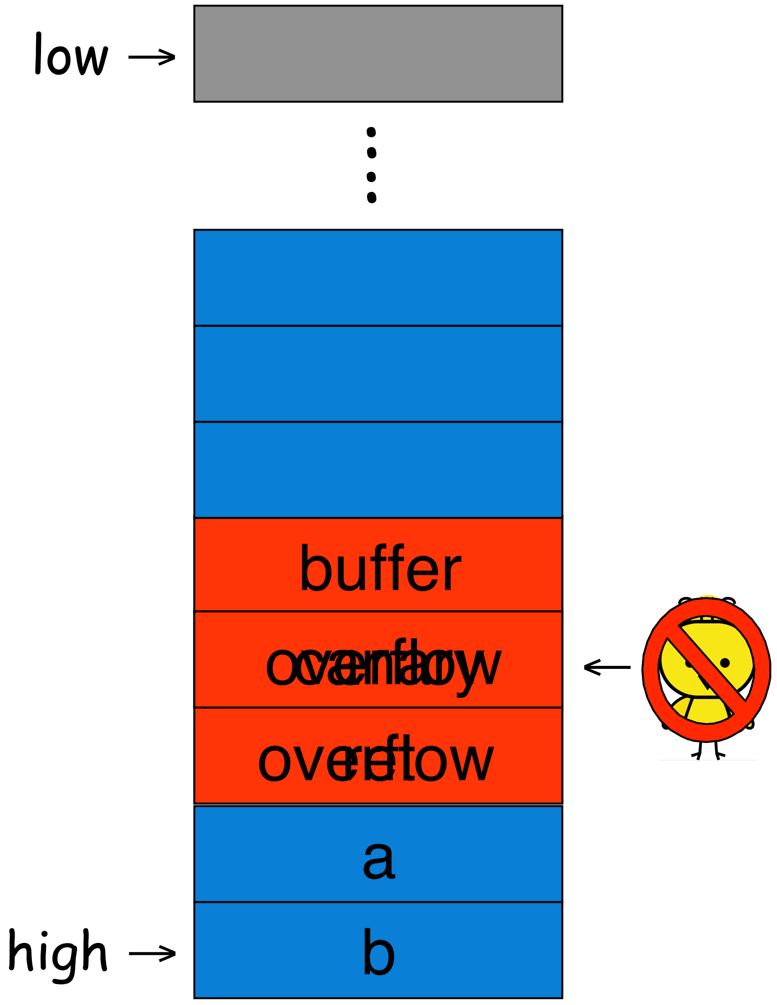
```
#include <stdio.h>
#include <string.h>

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 Prevention

- 1st choice: 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 does this)
- 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
- 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!

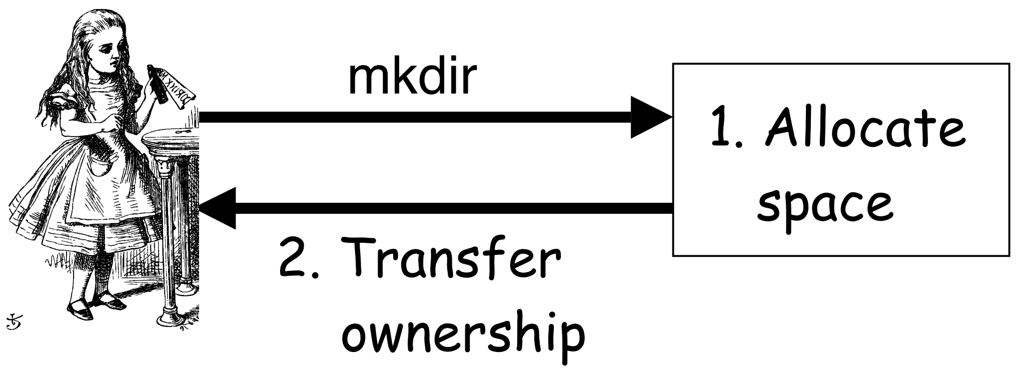
Race Conditions

Race Condition

- Security processes should be **atomic**
  - Occur “all at once”
- 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
- Example: Unix mkdir

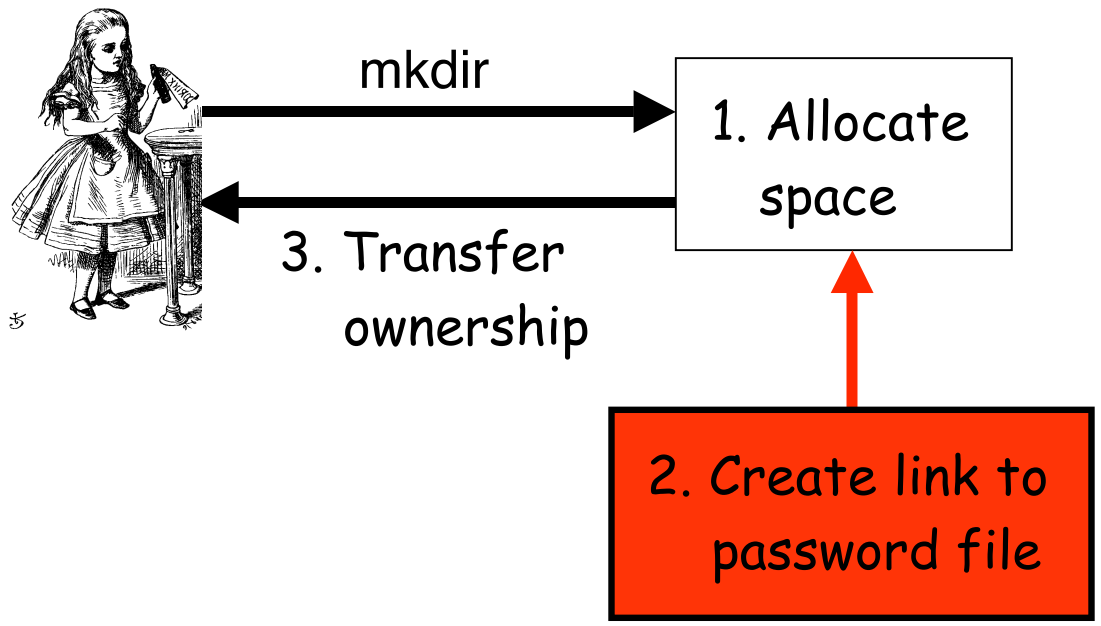
mkdir Race Condition

- 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
  - Buffer overflow is "low hanging fruit" today
- 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

Viruses/Worms

- Where do viruses live?
- Boot sector
  - Take control before anything else
- Memory resident
  - Stays in memory
- Applications, macros, data, etc.
- Library routines
- Compilers, debuggers, virus checker, etc.
  - These are particularly nasty!

Malware Timeline

- Preliminary work by Cohen (early 80's)
- Brain virus (1986)
- Morris worm (1988)
- Code Red (2001)
- SQL Slammer (2004)
- 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 malicious

Morris Worm

- 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

- 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

- Once access had been obtained to machine
- "Bootstrap loader" sent to victim
  - Consisted of 99 lines of C code
- Victim machine compiled and executed code
- Bootstrap loader then fetched the rest of the worm
- Victim even **authenticated** the sender!

Morris Worm

- 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

- 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

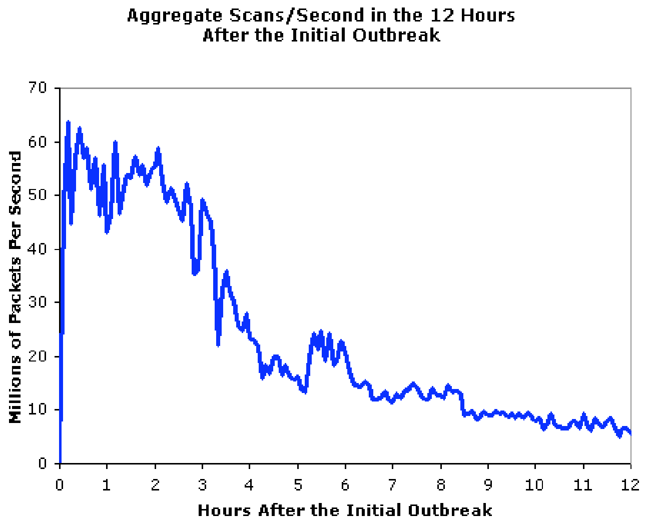
- Appeared in July 2001
- Infected more than **250,000 systems in about 15 hours**
- In total, infected 750,000 out of 6,000,000 susceptible systems
- Exploited buffer overflow in Microsoft IIS server software
- Then monitored traffic on port 80 for other susceptible servers

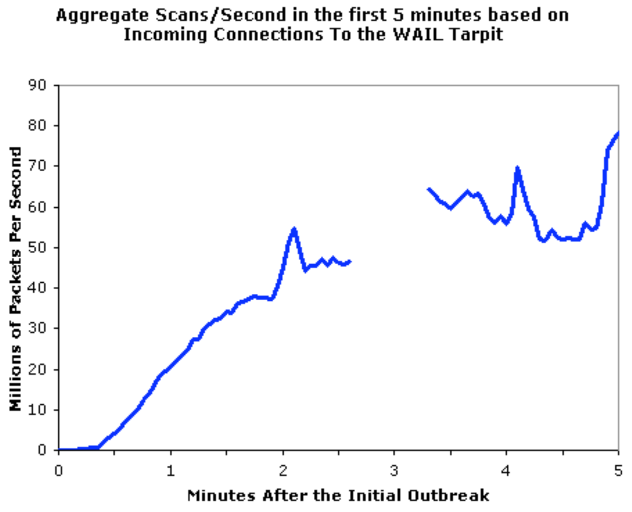
Code Red Worm

- What it did
  - Day 1 to 19 of month: tried to spread infection
  - Day 20 to 27: distributed denial of service 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

- 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

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

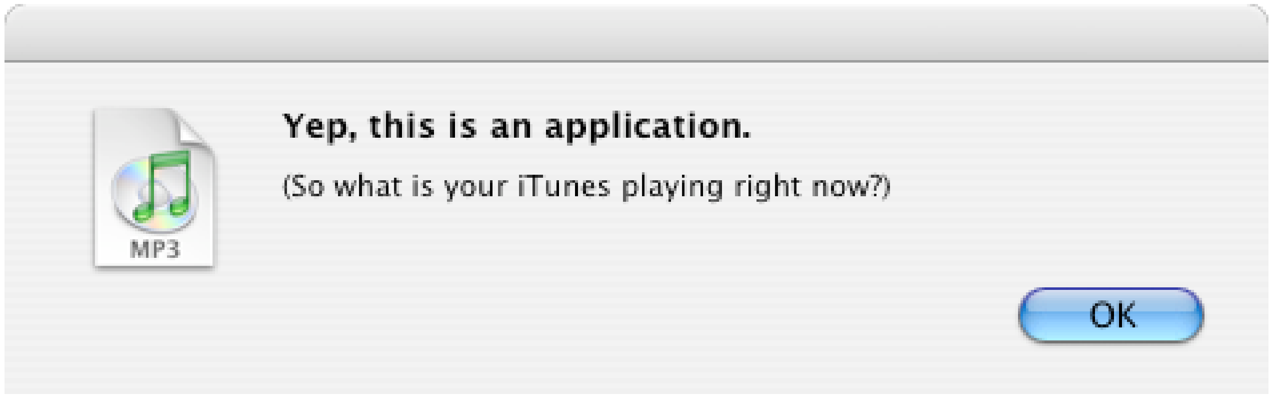
Trojan Horse Example

- 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 Example

- Double click on freeMusic.mp3
  - iTunes opens (expected)
  - "Wild Laugh" (probably not expected)
  - Message box (unexpected)

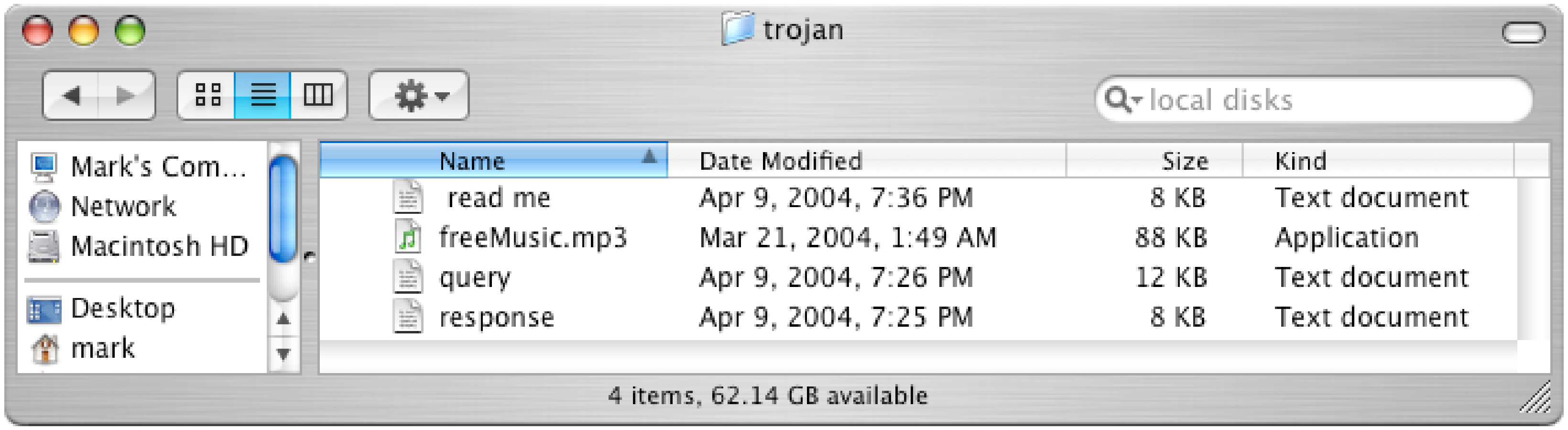


Yep, this is an application.

(So what is your iTunes playing right now?)

Trojan Example

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



Name	Date Modified	Size	Kind
read me	Apr 9, 2004, 7:36 PM	8 KB	Text document
freeMusic.mp3	Mar 21, 2004, 1:49 AM	88 KB	Application
query	Apr 9, 2004, 7:26 PM	12 KB	Text document
response	Apr 9, 2004, 7:25 PM	8 KB	Text document

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

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

- 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

- 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

- 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

- 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

- 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

- 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!
  - As difficult as AI?

Future of Malware

- Polymorphic 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

- Polymorphic worm (usually) encrypted
- New key is used each time worm propagates
  - 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
- Detectable by signature-based method
  - Though more challenging than non-polymorphic…

Metamorphic 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 Worm

- 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

- "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
- Can a worm do "better" than Slammer?

Warhol Worm

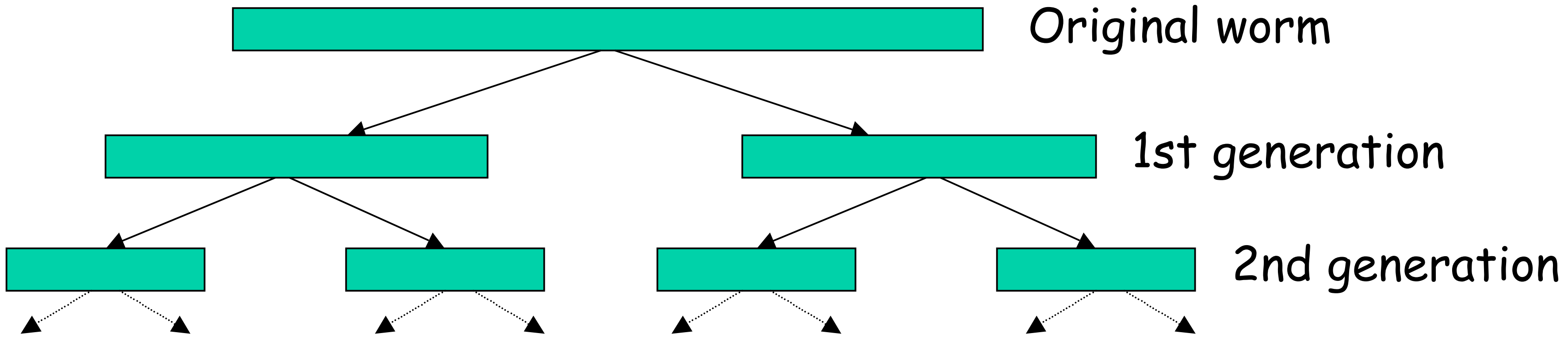
- One approach to a Warhol worm…
- Seed worm with an initial **hit list** containing a set of vulnerable IP addresses
  - Depends on the particular exploit
  - Tools exist for finding vulnerable systems
- Each successful initial infection would attack selected part of IP address space
- No worm this sophisticated has yet been seen in the wild (as of 2004)
  - Slammer generated random IP addresses
- Could infect entire Internet in 15 minutes!

Flash Worm

- 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

- 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

- 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

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

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 Attack

- 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 Attack

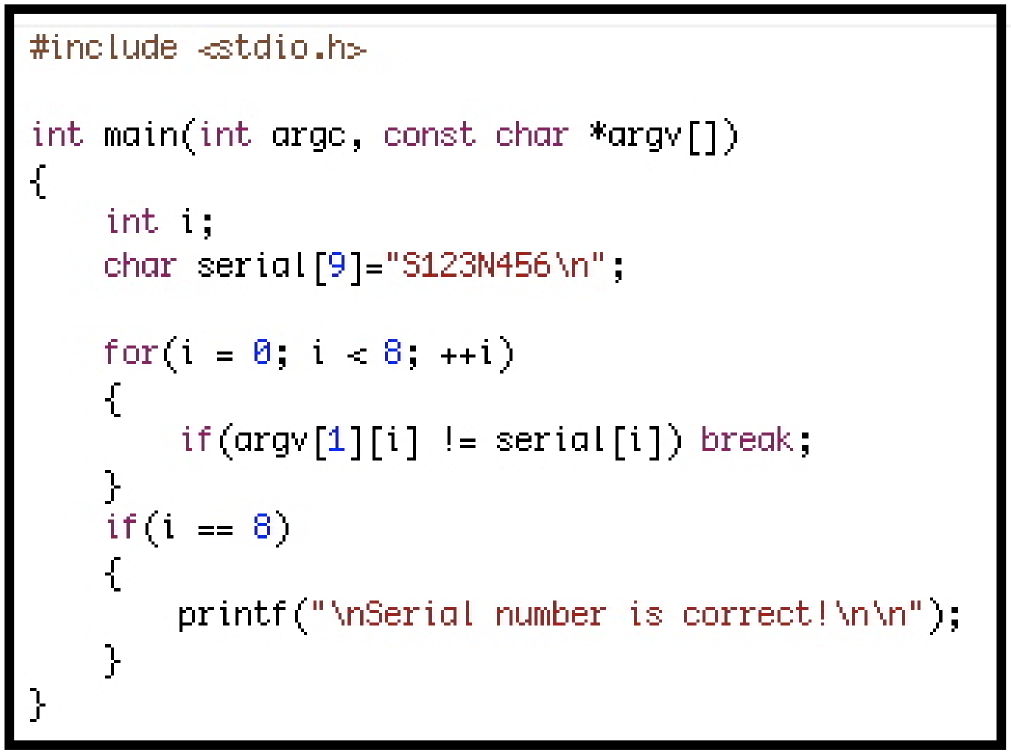
- Such attacks are possible for insiders
- Do salami attacks actually occur?
- Programmer added a few cents to every employee payroll tax withholding
  - But money credited to programmer's tax
  - Programmer got a big tax refund!
- Rent-a-car franchise in Florida inflated gas tank capacity to overcharge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

Salami Attacks

- 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

- 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

- 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

- 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

- 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

- 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

- 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

Software Reverse Engineering (SRE)

SRE

- **Software Reverse Engineering**
  - Also known as Reverse Code Engineering (RCE)
  - Or simply "reversing"
- Can be used for **good**...
  - Understand malware
  - Understand legacy code
- ...or **not-so-good**
  - Remove usage restrictions from software
  - Find and exploit flaws in software
  - Cheat at games, etc.

SRE

- We assume that
  - Reverse engineer is an attacker
  - Attacker only has exe (no source code)
- Attacker might want to
  - Understand the software
  - Modify the software
- SRE usually focused on Windows
- So we'll focus on Windows

SRE Tools

- Disassembler
  - Converts exe to assembly — as best it can
  - Cannot always disassemble correctly
  - Generally, it is not possible to assemble disassembly into working exe
- Debugger
  - Must step thru code to completely understand it
  - Labor intensive — lack of automated tools
- Hex Editor
  - To "patch" (make changes to) exe file
- Regmon, Filemon, VMware, etc.

SRE Tools

- IDA Pro is the top-rated disassembler
  - Cost is a few hundred dollars
  - Converts binary to assembly (as best it can)
- SoftICE is "alpha and omega" of debuggers
  - Cost is in the $1000's
  - Kernel mode debugger
  - Can debug anything, even the OS
- OllyDbg is a high quality shareware debugger
  - Includes a good disassembler
- Hex editor — to view/modify bits of exe
  - UltraEdit is good — freeware
  - HIEW — useful for patching exe
- Regmon, Filemon — freeware

Why is a Debugger Needed?

- Disassembler gives **static** results
  - Good overview of program logic
  - But need to "mentally execute" program
  - Difficult to jump to specific place in the code
- Debugger is **dynamic**
  - Can set break points
  - Can treat complex code as "black box"
  - Not all code disassembles correctly
- Disassembler **and** debugger both required for any serious SRE task

SRE Necessary Skills

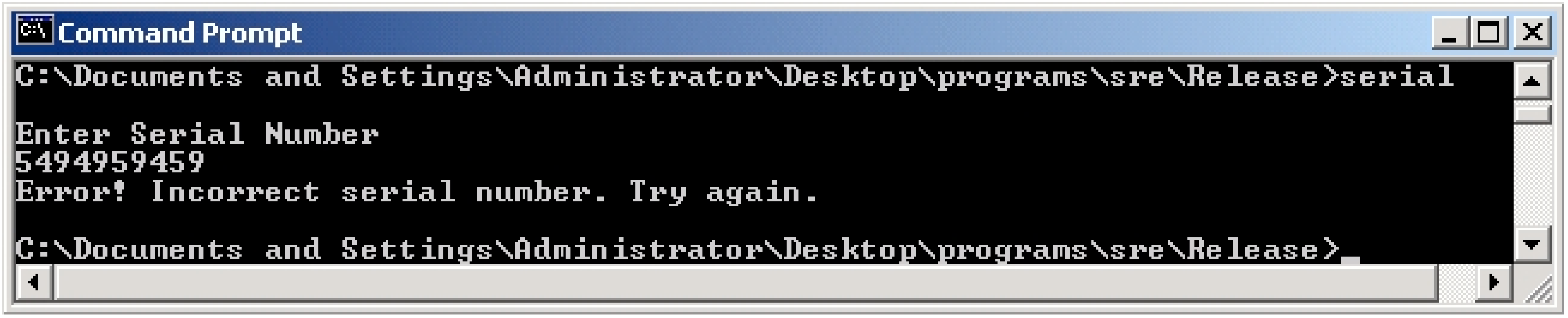
- Working knowledge of target assembly code
- Experience with the tools
  - IDA Pro — sophisticated and complex
  - SoftICE — large two-volume users manual
- Knowledge of Windows **Portable Executable** (PE) file format
- Boundless patience and optimism
- SRE is tedious and labor-intensive process!

SRE Example

- Consider simple example
- This example only requires disassembler (IDA Pro) and hex editor
  - Trudy disassembles to understand code
  - Trudy also wants to patch the code
- For most real-world code, also need a debugger (SoftICE or OllyDbg)

SRE Example

- Program requires serial number
- But Trudy doesn't know the serial number!

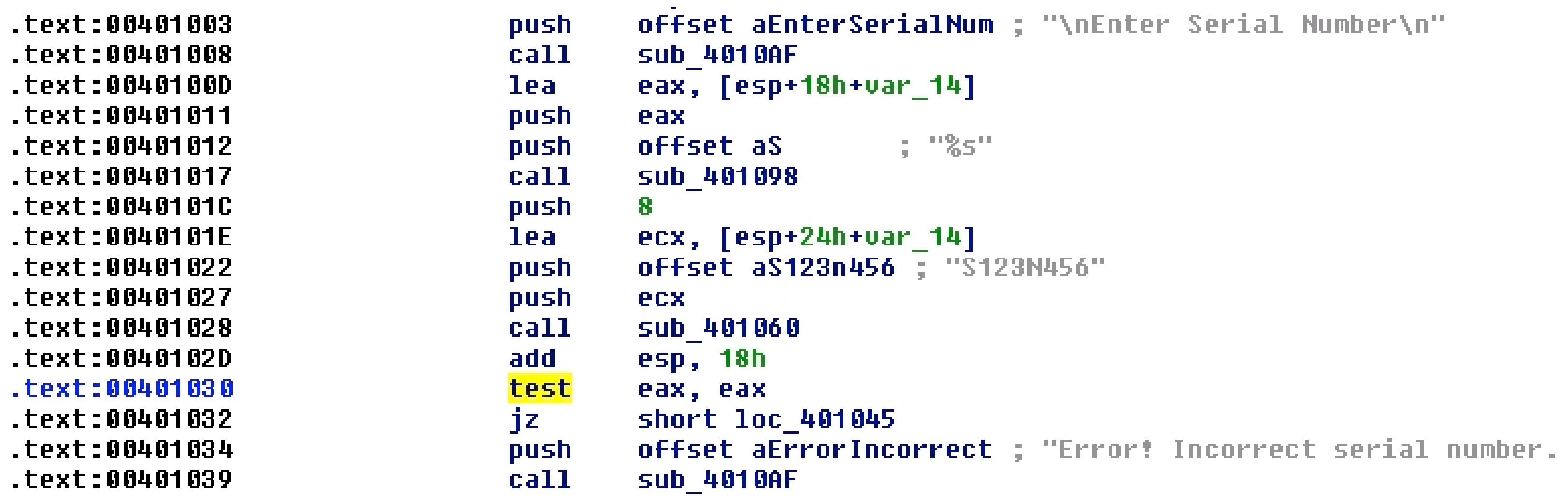


```
C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>serial

Enter Serial Number
5494959459
Error! Incorrect serial number. Try again.

C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>_
```

- Can Trudy find the serial number?

SRE Example

- IDA Pro disassembly

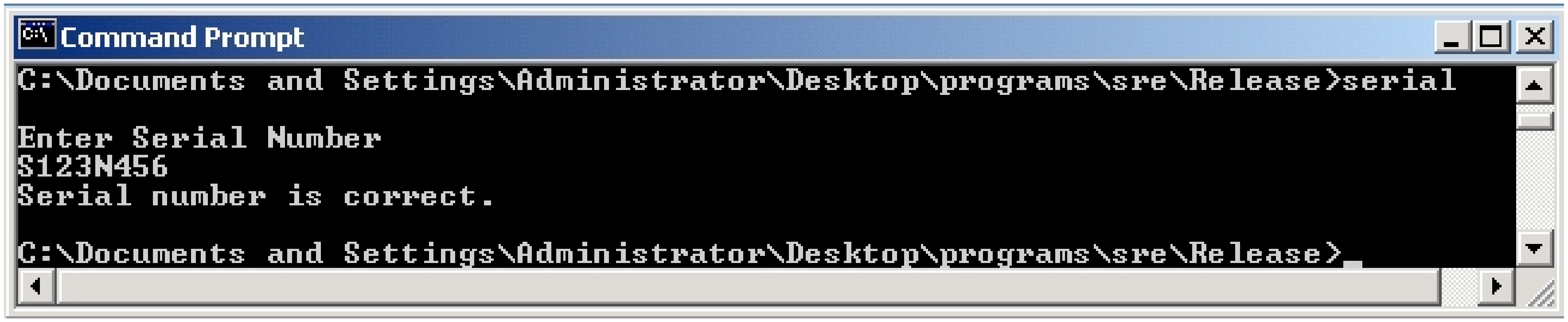


```
.text:00401003                 push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008                 call    sub_4010AF
.text:0040100D                 lea     eax, [esp+18h+var_14]
.text:00401011                 push    eax
.text:00401012                 push    offset aS       ; "%s"
.text:00401017                 call    sub_401098
.text:0040101C                 push    8
.text:0040101E                 lea     ecx, [esp+24h+var_14]
.text:00401022                 push    offset aS123n456 ; "S123N456"
.text:00401027                 push    ecx
.text:00401028                 call    sub_401060
.text:0040102D                 add     esp, 18h
.text:00401030                 test    eax, eax
.text:00401032                 jz      short loc_401045
.text:00401034                 push    offset aErrorIncorrect ; "Error! Incorrect serial number.
.text:00401039                 call    sub_4010AF
```

- Looks like serial number is S123N456

SRE Example

- Try the serial number S123N456

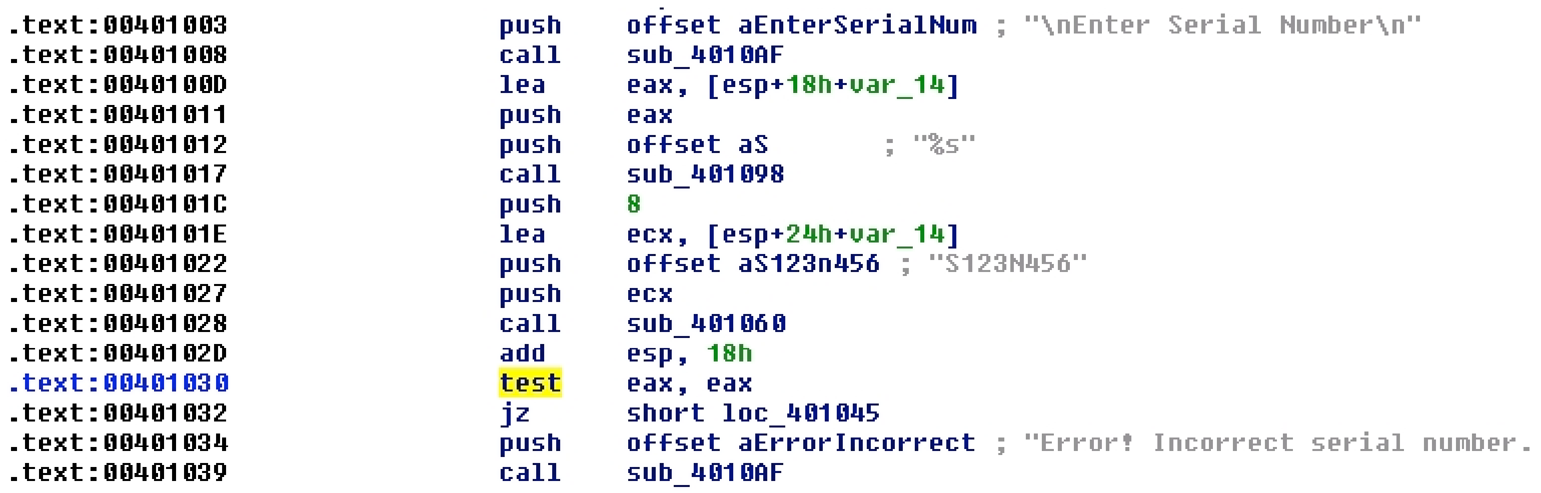


```
C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>serial

Enter Serial Number
S123N456
Serial number is correct.

C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>_
```

- It works!
- Can Trudy do better?

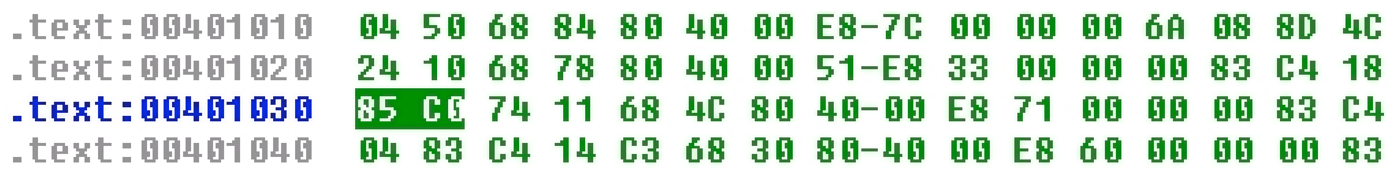
SRE Example

- Again, IDA Pro disassembly



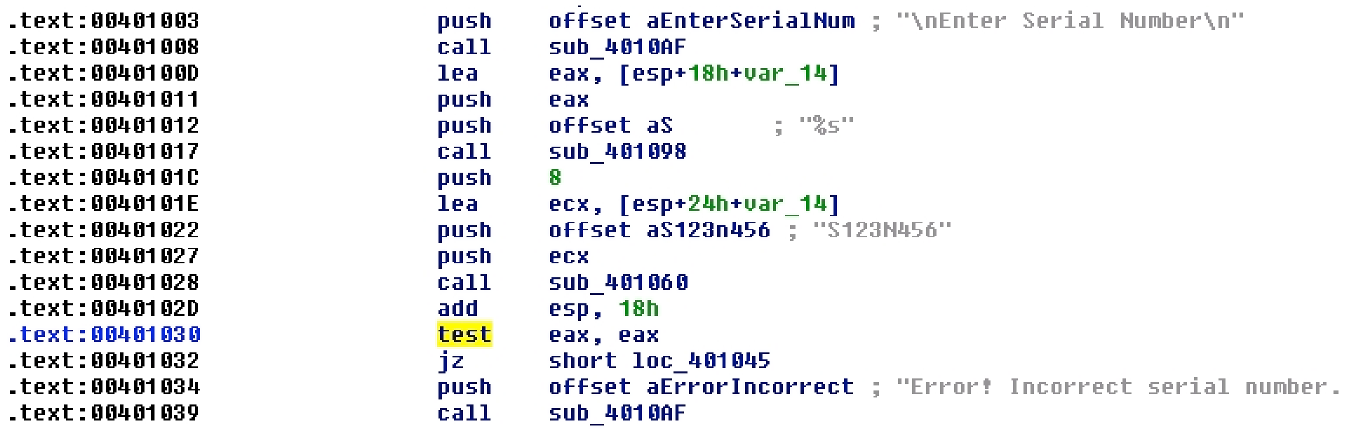
```
.text:00401003                 push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008                 call    sub_4010AF
.text:0040100D                 lea     eax, [esp+18h+var_14]
.text:00401011                 push    eax
.text:00401012                 push    offset aS       ; "%s"
.text:00401017                 call    sub_401098
.text:0040101C                 push    8
.text:0040101E                 lea     ecx, [esp+24h+var_14]
.text:00401022                 push    offset aS123n456 ; "S123N456"
.text:00401027                 push    ecx
.text:00401028                 call    sub_401060
.text:0040102D                 add     esp, 18h
.text:00401030                 test    eax, eax
.text:00401032                 jz      short loc_401045
.text:00401034                 push    offset aErrorIncorrect ; "Error! Incorrect serial number.
.text:00401039                 call    sub_4010AF
```

- And hex view…



```
.text:00401010  04 50 68 84 80 40 00 E8-7C 00 00 00 6A 08 8D 4C
.text:00401020  24 10 68 78 80 40 00 51-E8 33 00 00 00 83 C4 18
.text:00401030  85 C0 74 11 68 4C 80 40-00 E8 71 00 00 00 83 C4
.text:00401040  04 83 C4 14 C3 68 30 80-40 00 E8 60 00 00 00 83
```

SRE Example

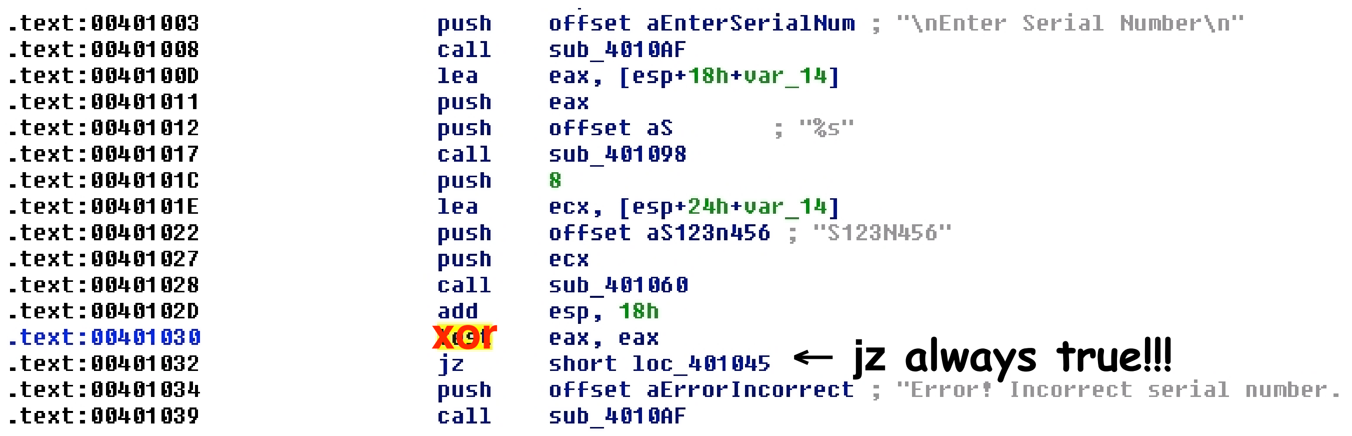


```
.text:00401003                 push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008                 call    sub_4010AF
.text:0040100D                 lea     eax, [esp+18h+var_14]
.text:00401011                 push    eax
.text:00401012                 push    offset aS       ; "%s"
.text:00401017                 call    sub_401098
.text:0040101C                 push    8
.text:0040101E                 lea     ecx, [esp+24h+var_14]
.text:00401022                 push    offset aS123n456 ; "S123N456"
.text:00401027                 push    ecx
.text:00401028                 call    sub_401060
.text:0040102D                 add     esp, 18h
.text:00401030                 test    eax, eax
.text:00401032                 jz      short loc_401045
.text:00401034                 push    offset aErrorIncorrect ; "Error! Incorrect serial number.
.text:00401039                 call    sub_4010AF
```

- test eax,eax gives AND of eax with itself
  - Result is 0 only if eax is 0
  - If test returns 0, then jz is true
- Trudy wants jz to always be true!
- Can Trudy patch exe so that jz always true?

SRE Example

- Can Trudy patch exe so that jz always true?

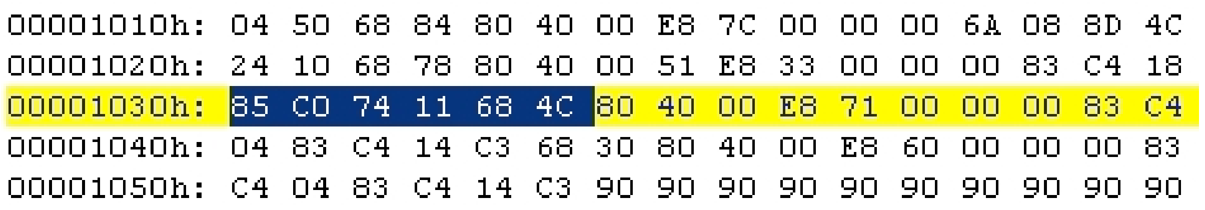


```
.text:00401003          push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008          call    sub_4010AF
.text:0040100D          lea     eax, [esp+18h+var_14]
.text:00401011          push    eax
.text:00401012          push    offset aS       ; "%s"
.text:00401017          call    sub_401098
.text:0040101C          push    8
.text:0040101E          lea     ecx, [esp+24h+var_14]
.text:00401022          push    offset aS123n456 ; "S123N456"
.text:00401027          push    ecx
.text:00401028          call    sub_401060
.text:0040102D          add     esp, 18h
.text:00401030          xor     eax, eax
.text:00401032          jz      short loc_401045  ← jz always true!!!
.text:00401034          push    offset aErrorIncorrect ; "Error! Incorrect serial number.
.text:00401039          call    sub_4010AF
```

Assembly		Hex
test	eax,eax	85 C0 …
xor	eax,eax	33 C0 …

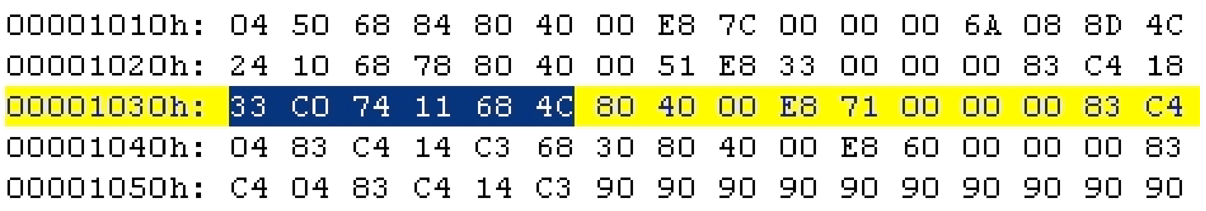
SRE Example

- Edit serial.exe with hex editor



serial.exe

```
00001010h: 04 50 68 84 80 40 00 E8 7C 00 00 00 6A 08 8D 4C
00001020h: 24 10 68 78 80 40 00 51 E8 33 00 00 00 83 C4 18
00001030h: 85 C0 74 11 68 4C 80 40 00 E8 71 00 00 00 83 C4
00001040h: 04 83 C4 14 C3 68 30 80 40 00 E8 60 00 00 00 83
00001050h: C4 04 83 C4 14 C3 90 90 90 90 90 90 90 90 90 90
```

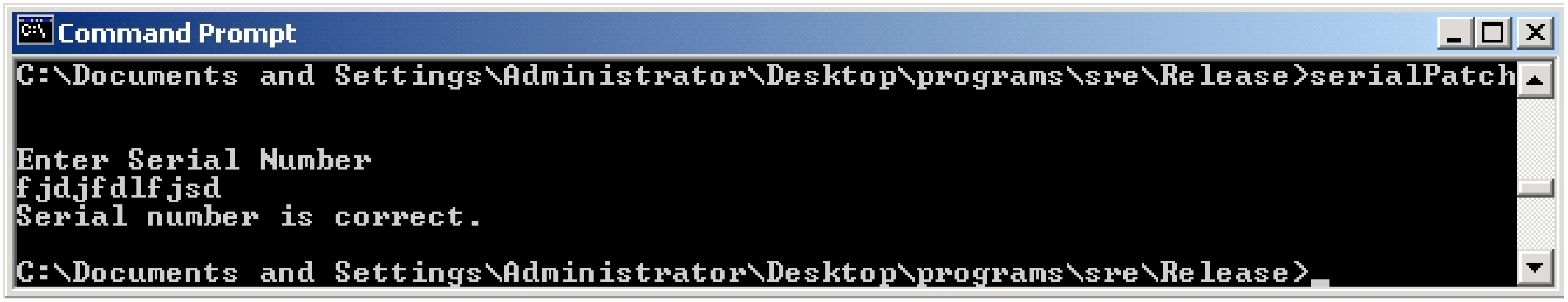


serialPatch.exe

```
00001010h: 04 50 68 84 80 40 00 E8 7C 00 00 00 6A 08 8D 4C
00001020h: 24 10 68 78 80 40 00 51 E8 33 00 00 00 83 C4 18
00001030h: 33 C0 74 11 68 4C 80 40 00 E8 71 00 00 00 83 C4
00001040h: 04 83 C4 14 C3 68 30 80 40 00 E8 60 00 00 00 83
00001050h: C4 04 83 C4 14 C3 90 90 90 90 90 90 90 90 90 90
```

- Save as serialPatch.exe

SRE Example



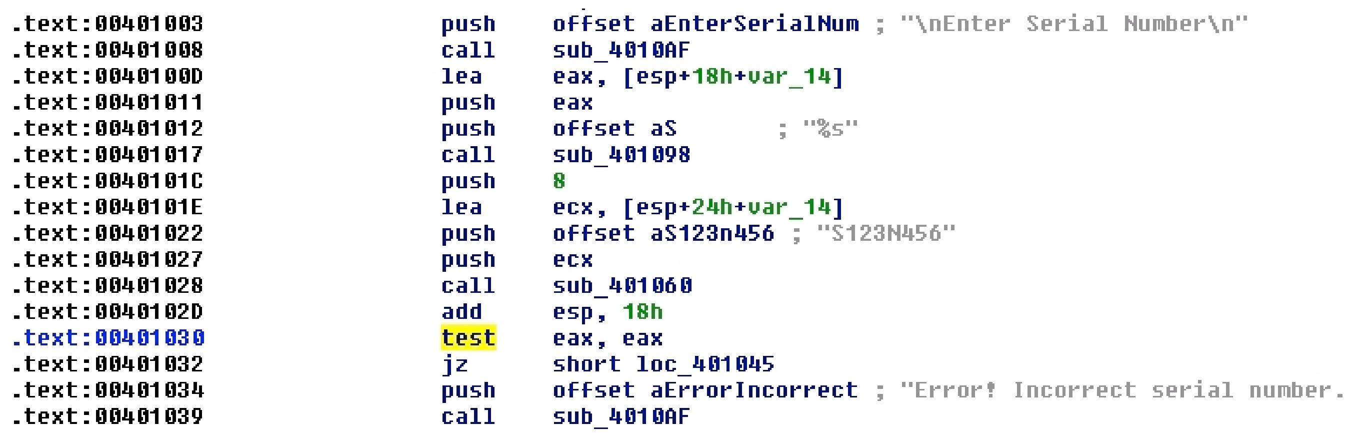
```
C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>serialPatch


Enter Serial Number
fjdjfdlfjsd
Serial number is correct.

C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>_
```

- ***Any*** "serial number" now works!
- Very convenient for Trudy!

SRE Example

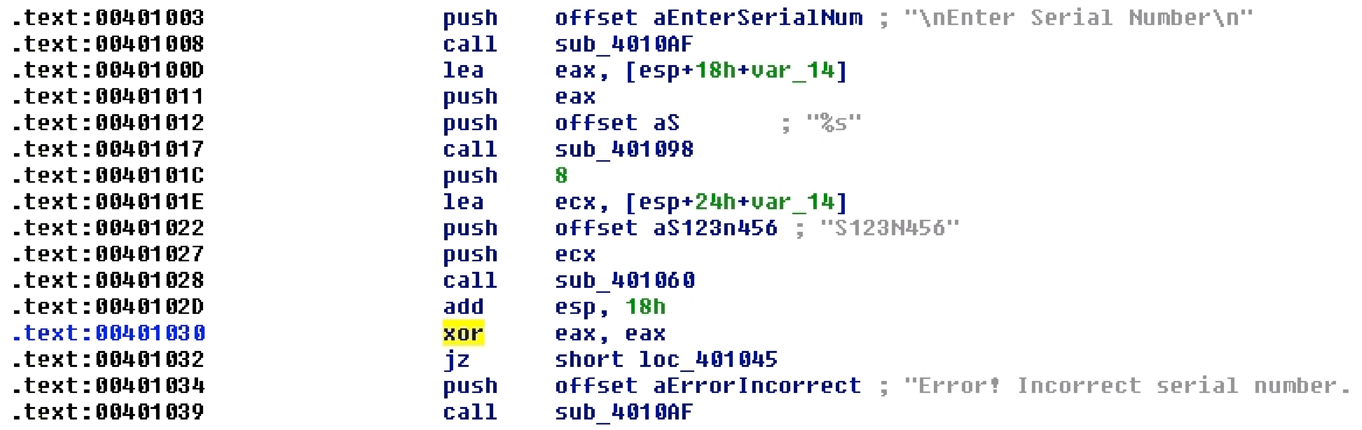
- Back to IDA Pro disassembly…

serial.exe



```
.text:00401003                 push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008                 call    sub_4010AF
.text:0040100D                 lea     eax, [esp+18h+var_14]
.text:00401011                 push    eax
.text:00401012                 push    offset aS       ; "%s"
.text:00401017                 call    sub_401098
.text:0040101C                 push    8
.text:0040101E                 lea     ecx, [esp+24h+var_14]
.text:00401022                 push    offset aS123n456 ; "S123N456"
.text:00401027                 push    ecx
.text:00401028                 call    sub_401060
.text:0040102D                 add     esp, 18h
.text:00401030                 test    eax, eax
.text:00401032                 jz      short loc_401045
.text:00401034                 push    offset aErrorIncorrect ; "Error! Incorrect serial number.
.text:00401039                 call    sub_4010AF
```

serialPatch.exe



```
.text:00401003                 push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
.text:00401008                 call    sub_4010AF
.text:0040100D                 lea     eax, [esp+18h+var_14]
.text:00401011                 push    eax
.text:00401012                 push    offset aS       ; "%s"
.text:00401017                 call    sub_401098
.text:0040101C                 push    8
.text:0040101E                 lea     ecx, [esp+24h+var_14]
.text:00401022                 push    offset aS123n456 ; "S123N456"
.text:00401027                 push    ecx
.text:00401028                 call    sub_401060
.text:0040102D                 add     esp, 18h
.text:00401030                 xor     eax, eax
.text:00401032                 jz      short loc_401045
.text:00401034                 push    offset aErrorIncorrect ; "Error! Incorrect serial number.
.text:00401039                 call    sub_4010AF
```

SRE Attack Mitigation

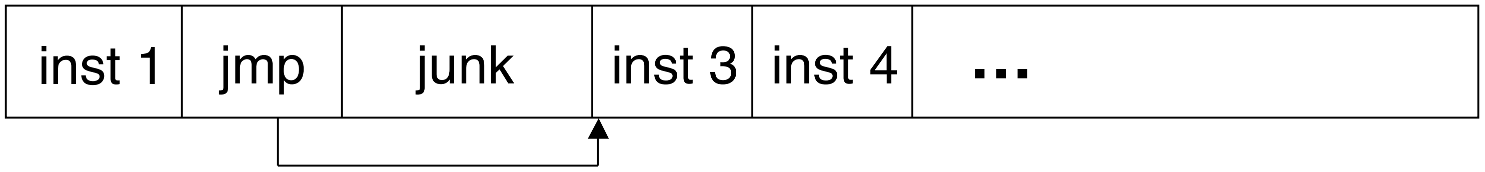
- **Impossible** to prevent SRE on open system
- But can make such attacks more difficult
- Anti-disassembly techniques
  - To confuse static view of code
- Anti-debugging techniques
  - To confuse dynamic view of code
- Tamper-resistance
  - Code checks itself to detect tampering
- Code obfuscation
  - Make code more difficult to understand

Anti-disassembly

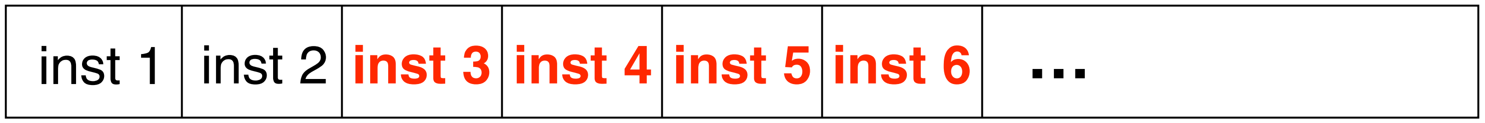
- Anti-disassembly methods include
  - Encrypted object code
  - False disassembly
  - Self-modifying code
  - Many others
- Encryption **prevents** disassembly
  - But still need code to decrypt the code!
  - Same problem as with polymorphic viruses

Anti-disassembly Example

- Suppose actual code instructions are



- What the disassembler sees



- This is example of "false disassembly"
- Clever attacker will figure it out!

Anti-debugging

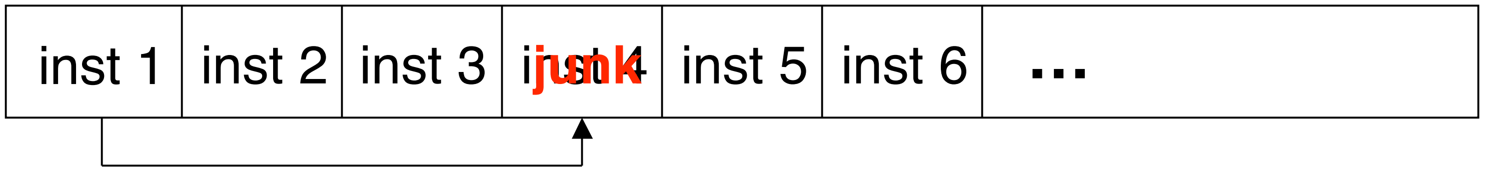
- Monitor for
  - Use of debug registers
  - Inserted breakpoints
- Debuggers don't handle threads well
  - Interacting threads may confuse debugger
- Many other debugger-unfriendly tricks
- Undetectable debugger possible in principle
  - Hardware-based debugging (HardICE) is possible

Anti-debugger Example

inst 1	inst 2	inst 3	inst 4	inst 5	inst 6	…

- Suppose when program gets inst 1, it pre-fetches inst 2, inst 3 and inst 4
  - This is done to increase efficiency
- Suppose when debugger executes inst 1, it does **not** pre-fetch instructions
- Can we use this difference to confuse the debugger?

Anti-debugger Example



- Suppose inst 1 **overwrites** inst 4 in memory
- Then program (without debugger) will be OK since it fetched inst 4 at same time as inst 1
- Debugger will be confused when it reaches **junk** where inst 4 is supposed to be
- Problem for program if this segment of code executed more than once!
- Also, code is very platform-dependent
- Again, clever attacker will figure this out!

Tamper-resistance

- Goal is to make patching more difficult
- Code can hash parts of itself
- If tampering occurs, hash check fails
- Research has shown can get good coverage of code with small performance penalty
- But don't want all checks to look similar
  - Or else easy for attacker to remove checks
- This approach sometimes called "guards"

Code Obfuscation

- Goal is to make code hard to understand
- Opposite of good software engineering!
- Simple example: spaghetti code
- Much research into more robust obfuscation
  - Example: **opaque predicate**

    ```
    int x,y
      :
    if((x–y)∗(x–y) > (x∗x–2∗x∗y+y∗y)){…}
    ```
  - The if() conditional is always false
- Attacker will waste time analyzing dead code

Code Obfuscation

- Code obfuscation sometimes promoted as a powerful security technique
- Diffie and Hellman's original ideas for public key crypto were based on similar ideas!
- Recently it has been shown that obfuscation probably cannot provide strong security
  - On the (im)possibility of obfuscating programs
  - Some question significance of result (Thomborson)
- Obfuscation might still have practical uses!
  - Even if it can never be as strong as crypto

Authentication Example

- Software used to determine authentication
- Ultimately, authentication is 1-bit decision
  - Regardless of method used (pwd, biometric, …)
- Somewhere in authentication software, a single bit determines success/failure
- If attacker can find this bit, he can force authentication to always succeed
- Obfuscation makes it more difficult for attacker to find this all-important bit

Obfuscation

- Obfuscation forces attacker to analyze larger amounts of code
- Method could be combined with
  - Anti-disassembly techniques
  - Anti-debugging techniques
  - Code tamper-checking
- All of these increase work (and pain) for attacker
- But a persistent attacker will ultimately win!

Software Cloning

- Suppose we write a piece of software
- We then distribute an identical copy (or clone) to each customers
- If an attack is found on one copy, the same attack works on all copies
- This approach has no resistance to "break once, break everywhere" (BOBE)
- This is the usual situation in software development

Metamorphic Software

- Metamorphism is used in malware
- Can metamorphism also be used for good?
- Suppose we write a piece of software
- Each copy we distribute is different
    - This is an example of metamorphic software
- Two levels of metamorphism are possible
    - All instances are functionally distinct (only possible in certain application)
    - All instances are functionally identical but differ internally (always possible)
- We consider the latter case

Metamorphic Software

- If we distribute N copies of cloned software
  - One successful attack breaks all N
- If we distribute N metamorphic copies, where each of N instances is functionally identical, but they differ internally
  - An attack on one instance does not necessarily work against other instances
  - In the best case, N times as much work is required to break all N instances

Metamorphic Software

- We cannot prevent SRE attacks
- The best we can hope for is BOBE resistance
- Metamorphism will improve BOBE resistance
- Consider the analogy to genetic diversity
  - If all plants in a field are genetically identical, one disease can kill **all** of the plants
  - If the plants in a field are genetically diverse, one disease can only kill **some** of the plants

Cloning vs Metamorphism

- Spse our software has a buffer overflow
- **Cloned** software
  - Same buffer overflow attack will work against **all** cloned copies of the software
- **Metamorphic** software
  - Unique instances — all are functionally the same, but they differ in internal structure
  - Buffer overflow exists in all instances
  - But a specific buffer overflow attack will only work against **some** instances
  - Buffer overflow attacks are delicate!

Metamorphic Software

- Metamorphic software is intriguing concept
- But raises concerns regarding
  - Software development
  - Software upgrades, etc.
- Metamorphism does not prevent SRE, but could make it infeasible on a large scale
- May be one of the best tools for increasing BOBE resistance
- Metamorphism currently used in malware
- But metamorphism not just for evil!

Digital Rights Management

Digital Rights Management

- DRM is a good example of limitations of doing security in software
- We'll discuss
  - What is DRM?
  - A PDF document protection system
  - DRM for streaming media
  - DRM in P2P application
  - DRM within an enterprise

What is DRM?

- "Remote control" problem
  - Distribute digital content
  - Retain some control on its use, **after delivery**
- **Digital book** example
  - Digital book sold online could have huge market
  - But might only sell 1 copy!
  - Trivial to make perfect digital copies
  - A fundamental change from pre-digital era
- Similar comments for digital music, video, etc.

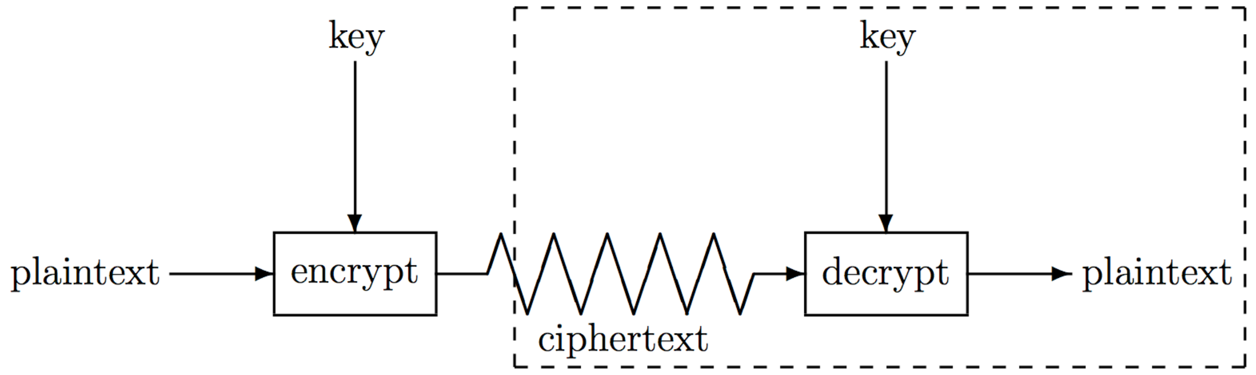
Persistent Protection

- "Persistent protection" is the fundamental problem in DRM
  - How to enforce restrictions on use of content **after** delivery?
- Examples of such restrictions
  - No copying
  - Limited number of reads/plays
  - Time limits
  - No forwarding, etc.

What Can be Done?

- The honor system?
  - Example: Stephen King's, *The Plant*
- Give up?
  - Internet sales? Regulatory compliance? etc.
- Lame software-based DRM?
  - The standard DRM system today
- Better software-based DRM?
  - MediaSnap's goal
- Tamper-resistant hardware?
  - Closed systems: Game Cube, etc.
  - Open systems: TCG/NGSCB for PCs

Is Crypto the Answer?



- Attacker's goal is to recover the **key**
- In standard crypto scenario, attacker has
  - Ciphertext, some plaintext, side-channel info, etc.
- In DRM scenario, attacker has
  - Everything in the box (at least)
- Crypto was not designed for this problem!

Is Crypto the Answer?

- But crypto is necessary
  - To securely deliver the bits
  - To prevent trivial attacks
- Then attacker will not try to directly attack crypto
- Attacker will try to find keys in software
  - DRM is "hide and seek" with keys in software!

Current State of DRM

- At best, **security by obscurity**
  - A derogatory term in security
- Secret designs
  - In violation of **Kerckhoffs Principle**
- Over-reliance on crypto
  - "Whoever thinks his problem can be solved using cryptography, doesn't understand his problem and doesn't understand cryptography."

    — Attributed by Roger Needham and Butler Lampson to each other

DRM Limitations

- The analog hole
  - When content is rendered, it can be captured in analog form
  - DRM cannot prevent such an attack
- Human nature matters
  - Absolute DRM security is impossible
  - Want something that "works" in practice
  - What works depends on context
- DRM is not strictly a technical problem!

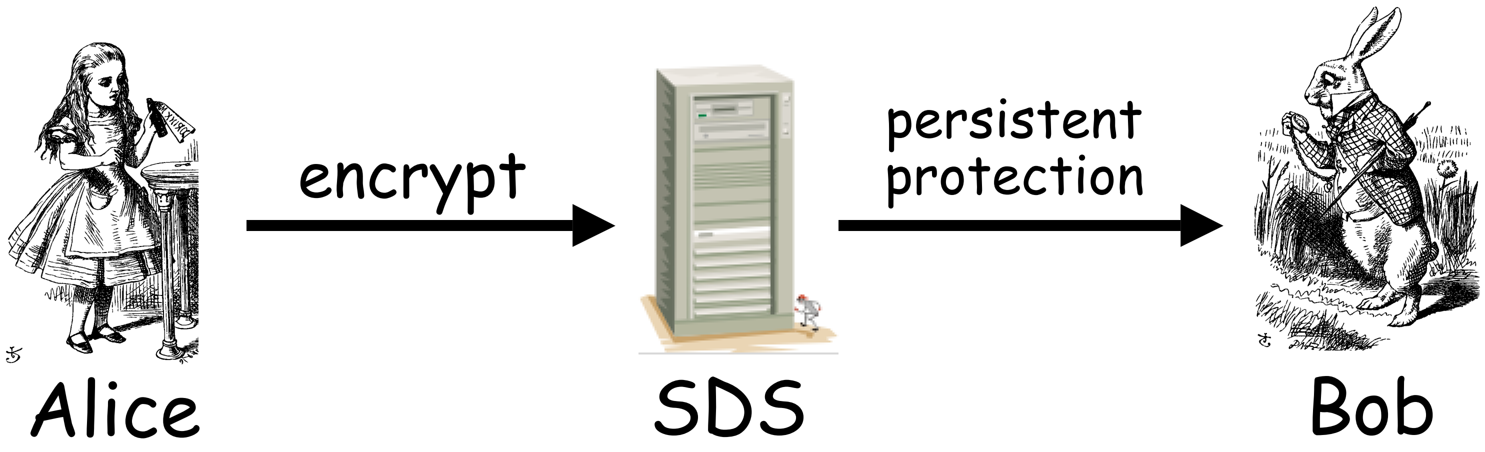
Software-based DRM

- Strong software-based DRM is impossible
- Why?
  - We can't really hide a secret in software
  - We cannot prevent SRE
  - User with full admin privilege can eventually break any anti-SRE protection
- Bottom line: **The** killer attack on software-based DRM is SRE

DRM for PDF Documents

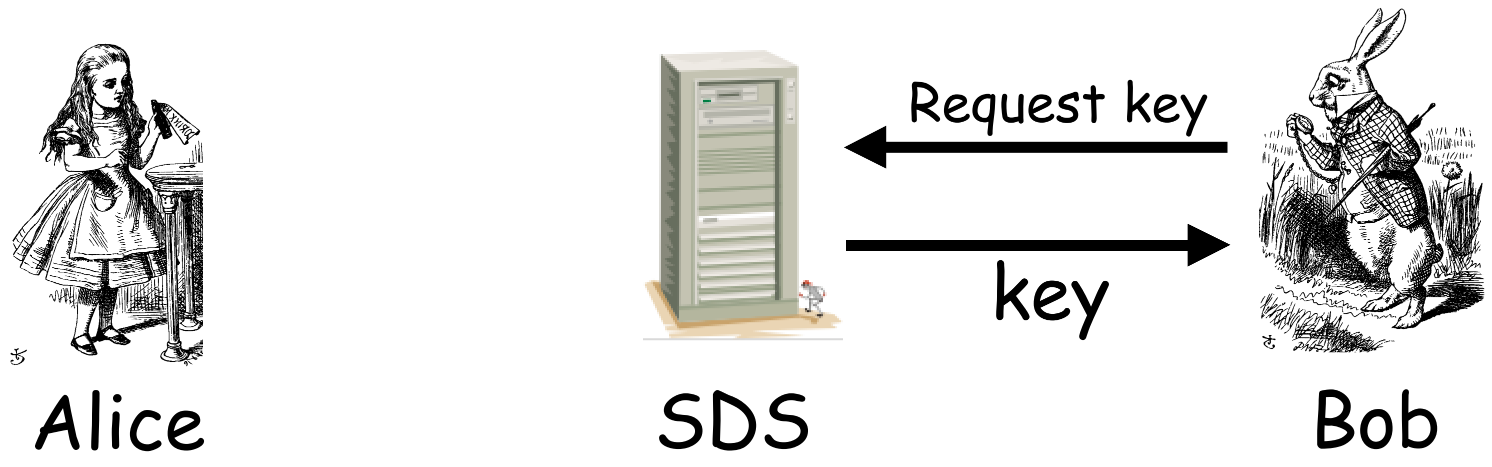
- Based on design of MediaSnap, Inc., a small Silicon Valley startup company
- Developed a DRM system
  - Designed to protect PDF documents
- Two parts to the system
  - Server — Secure Document Server (SDS)
  - Client — PDF Reader "plugin" software

Protecting a Document



- Alice creates PDF document
- Document encrypted and sent to SDS
- SDS applies desired "persistent protection"
- Document sent to Bob

Accessing a Document

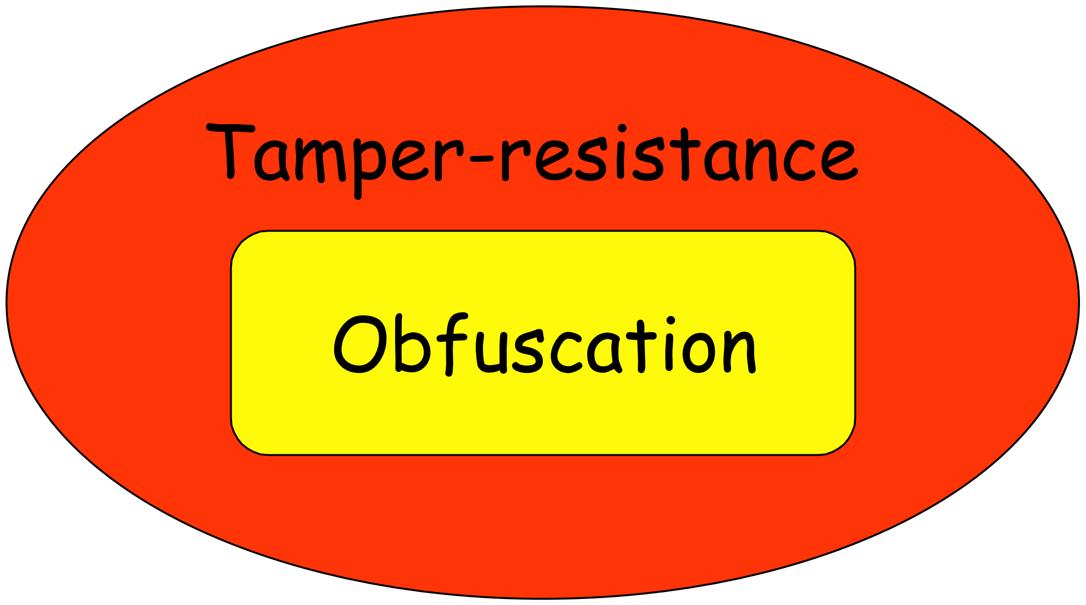


- Bob authenticates to SDS
- Bob requests key from SDS
- Bob can then access document, but only thru special DRM software

Security Issues

- Server side (SDS)
  - Protect keys, authentication data, etc.
  - Apply persistent protection
- Client side (PDF plugin)
  - Protect keys, authenticate user, etc.
  - Enforce persistent protection
- Remaining discussion concerns **client**

Security Overview



- A tamper-resistant outer layer
- Software obfuscation applied within

Tamper-Resistance

$$\begin{tikzcd}
\text{Anti-debugger} \arrow[r, shift left] & \text{Encrypted code} \arrow[l, shift left]
\end{tikzcd}$$

- Encrypted code will prevent static analysis of PDF plugin software
- Anti-debugging to prevent dynamic analysis of PDF plugin software
- These two designed to protect each other
- But the persistent attacker will get thru!

Obfuscation

- Obfuscation can be used for
  - Key management
  - Authentication
  - Caching (keys and authentication info)
  - Encryption and “scrambling”
  - Key parts (data and/or code)
  - Multiple keys/key parts
- Obfuscation can only slow the attacker
- The persistent attacker still wins!

Other Security Features

- Code tamper checking (hashing)
  - To validate all code executing on system
- Anti-screen capture
  - To prevent obvious attack on digital documents
- Watermarking
  - In theory, can trace stolen content
  - In practice, of limited value
- Metamorphism (or individualization)
  - For BOBE-resistance

Security Not Implemented

- More general code obfuscation
- Code "fragilization"
  - Code that hash checks itself
  - Tampering should cause code to break
- OS cannot be trusted
  - How to protect against "bad" OS?
  - Not an easy problem!

DRM for Streaming Media

- Stream digital content over Internet
  - Usually audio or video
  - Viewed in real time
- Want to charge money for the content
- Can we protect content from capture?
  - So content can't be redistributed
  - We want to make money!

Attacks on Streaming Media

- Spoof the stream between endpoints
- Man in the middle
- Replay and/or redistribute data
- **Capture the plaintext**
  - This is the threat we are concerned with
  - Must prevent malicious software from capturing plaintext stream at client end

Design Features

- Scrambling algorithms
  - Encryption-like algorithms
  - Many distinct algorithms available
  - A strong form of metamorphism!
- Negotiation of scrambling algorithm
  - Server and client must both know the algorithm
- Decryption at receiver end
  - To remove the strong encryption
- De-scrambling in device driver
  - De-scramble just prior to rendering

Scrambling Algorithms

- Server has a large set of scrambling algorithms
  - Suppose N of these numbered 1 thru N
- Each client has a subset of algorithms
  - For example: LIST = {12,45,2,37,23,31}
- The LIST is stored on client, encrypted with server's key: $E(\text{LIST},K_{\text{server}})$

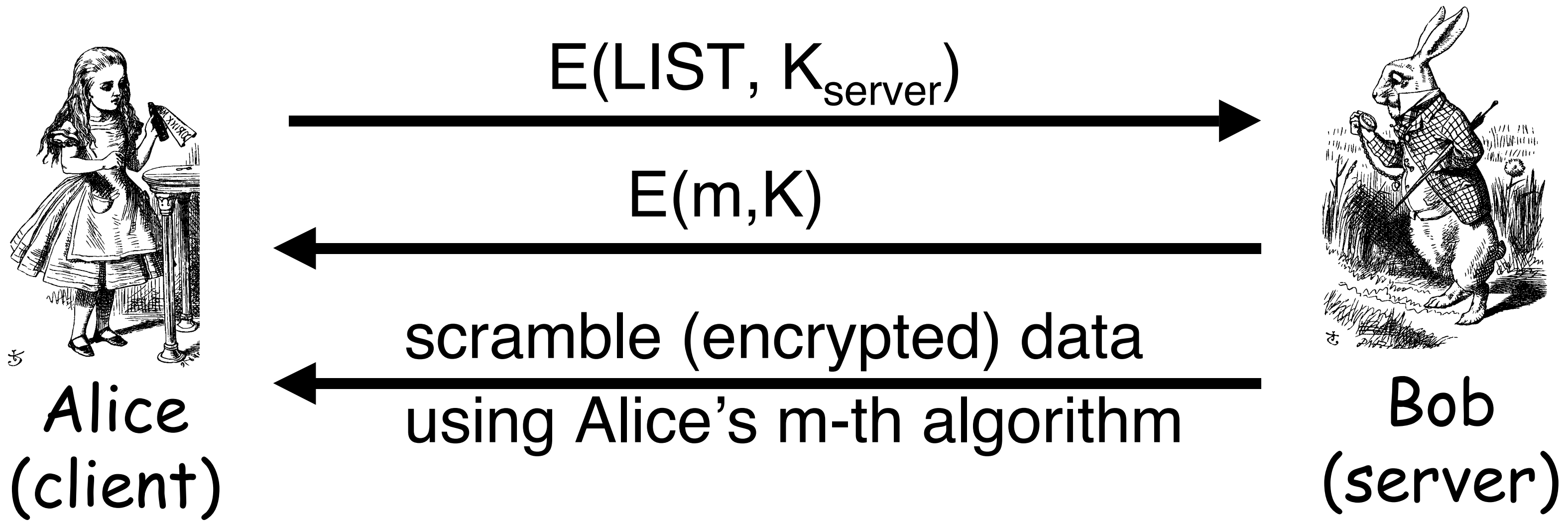
Server-side Scrambling

- On server side

$$\begin{tikzcd}\text{data} \arrow[r] & \begin{array}{c}\text{scrambled}\\ \text{data}\end{array} \arrow[r] & \begin{array}{c}\text{encrypted}\\ \text{scrambled data}\end{array}\end{tikzcd}$$

- Server must scramble data with an algorithm the client supports
- Client must send server list of algorithms it supports
- Server must securely communicate algorithm choice to client

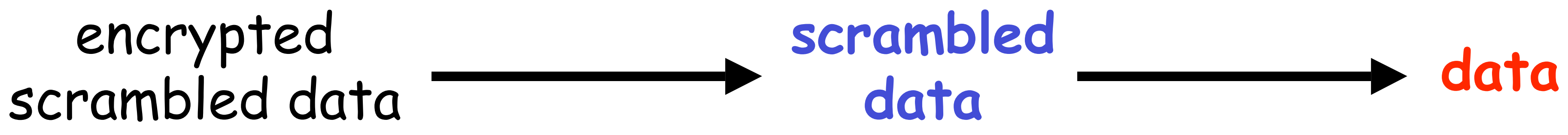
Select Scrambling Algorithm



- The key K is a session key
- The LIST is unreadable by client
  - Reminiscent of Kerberos TGT

Client-side De-scrambling

- On client side



- Try to keep plaintext away from potential attacker
- "Proprietary" device driver
  - Scrambling algorithms "baked in"
  - Able to de-scramble at last moment

Why Scrambling?

- **Metamorphism** deeply embedded in system
- If a scrambling algorithm is known to be broken, server will not choose it
- If client has too many broken algorithms, server can force software upgrade
- Proprietary algorithm harder for SRE
- We cannot trust crypto strength of proprietary algorithms, so we also encrypt

Why Metamorphism?

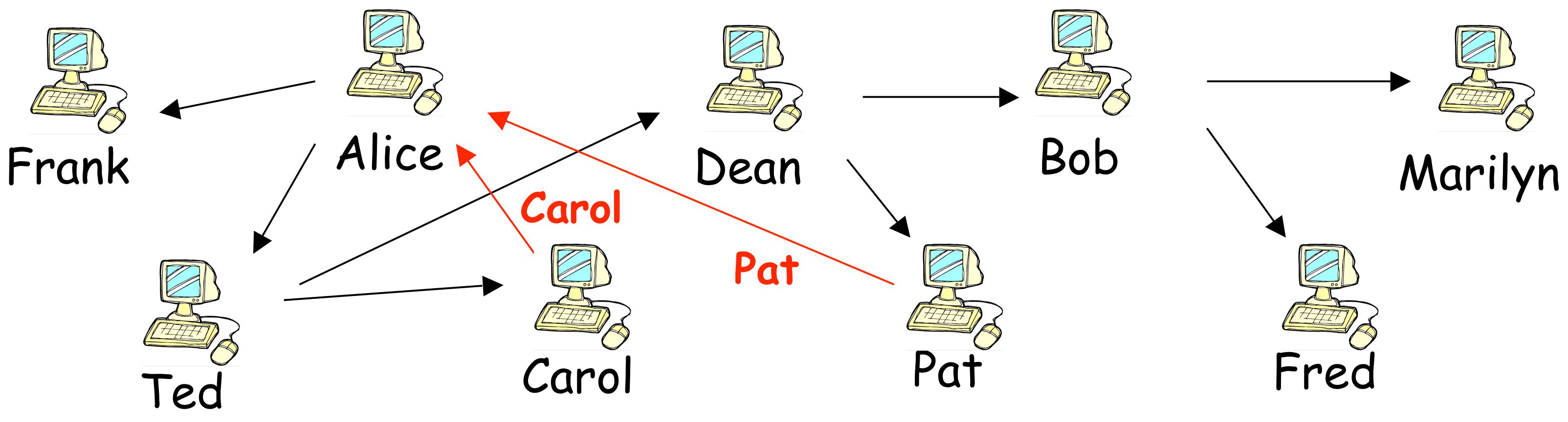
- The most serious threat is **SRE**
- Attacker does not need to reverse engineer any standard crypto algorithm
  - Attacker only needs to find the key
- Reverse engineering a scrambling algorithm may be difficult
- This is just **security by obscurity**
- But appears to help with BOBE-resistance

DRM for a P2P Application

- Today, much digital content is delivered via peer-to-peer (P2P) networks
  - P2P networks contain lots of pirated music
- Is it possible to get people to pay for digital content on such P2P networks?
- How can this possibly work?
- A peer offering service (POS) is one idea

P2P File Sharing: Query

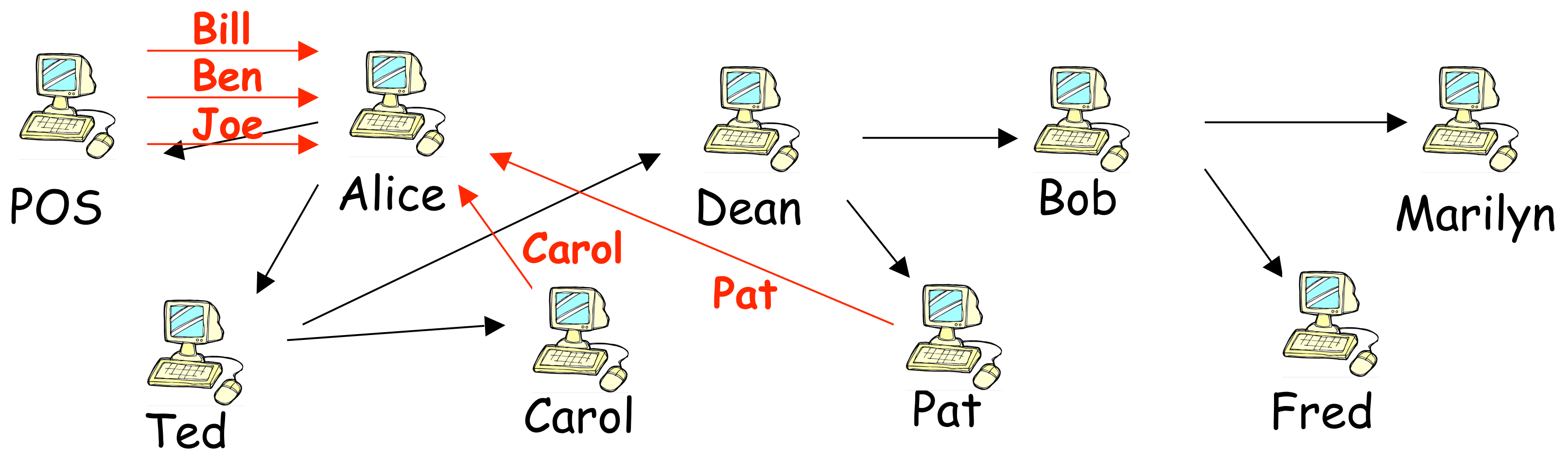
- Suppose Alice requests "Hey Jude"
- **Black** arrows: query flooding
- **Red** arrows: positive responses



- Alice can select from: **Carol**, **Pat**

P2P File Sharing with POS

- Suppose Alice requests "Hey Jude"
- **Black** arrow: query
- **Red** arrow: positive response



- Alice selects from: **Bill**, **Ben**, **Carol**, **Joe**, **Pat**
- **Bill**, **Ben**, and **Joe** have legal content!

POS

- Bill, Ben and Joe must appear normal to Alice
- If "victim" (Alice) clicks POS response
  - DRM protected (legal) content downloaded
  - **Then** small payment required to play
- Alice can choose not to pay
  - But then she must download again
  - Is it worth the hassle to avoid paying small fee?
  - POS content can also offer extras

POS Conclusions

- A very clever idea!
- Piggybacking on existing P2P networks
- Weak DRM works very well here
  - Pirated content already exists
  - DRM only needs to be more hassle to break than the hassle of clicking and waiting
- Current state of POS?
  - Very little interest from the music industry
  - Considerable interest from the "adult" industry

DRM in the Enterprise

- Why enterpise DRM?
- Health Insurance Portability and Accountability Act (HIPAA)
  - Medical records must be protected
  - Fines of up to $10,000 "per incident"
- Sarbanes-Oxley Act (SOA)
  - Must preserve documents of interest to SEC
- DRM-like protections needed by corporations for **regulatory compliance**

What's Different in Enterprise DRM?

- Technically, similar to e-commerce
- But motivation for DRM is different
  - Regulatory compliance
  - To satisfy a legal requirement
  - Not to make money — to avoid losing money!
- Human dimension is completely different
  - Legal threats are far more plausible
- Legally, corporation is OK provided an **active attack** on DRM is required

Enterprise DRM

- Moderate DRM security is sufficient
- **Policy management issues**
  - Easy to set policies for groups, roles, etc.
  - Yet policies must be flexible
- ***Authentication issues***
  - Must interface with existing system
  - Must prevent network authentication spoofing (authenticate the authentication server)
- Enterprise DRM is a solvable problem!

DRM Failures

- Many examples of DRM failures
  - One system defeated by a felt-tip pen
  - One defeated my holding down shift key
  - Secure Digital Music Initiative (SDMI) completely broken before it was finished
  - Adobe eBooks
  - Microsoft MS-DRM (version 2)
  - Many, many others!

DRM Conclusions

- DRM nicely illustrates limitations of doing security in software
- Software in a hostile environment is extremely vulnerable to attack
- Protection options are very limited
- Attacker has enormous advantage
- Tamper-resistant hardware and a trusted OS can make a difference
  - We'll discuss this more later: TCG/NGSCB

Secure Software Development

Penetrate and Patch

- Usual approach to software development
  - Develop product as quickly as possible
  - Release it without adequate testing
  - Patch the code as flaws are discovered
- In security, this is "penetrate and patch"
  - A **bad** approach to software development
  - A **horrible** approach to secure software!

Why Penetrate and Patch?

- First to market advantage
  - First to market likely to become market leader
  - Market leader has huge advantage in software
  - Users find it safer to “follow the leader”
  - Boss won’t complain if your system has a flaw, as long as everybody else has the same flaw
  - User can ask more people for support, etc.
- Sometimes called “network economics”

Why Penetrate and Patch?

- Secure software development is hard
  - Costly and time consuming development
  - Costly and time consuming testing
  - Easier to let customers do the work!
- No serious economic disincentive
  - Even if software flaw causes major losses, the software vendor is not liable
  - Is any other product sold this way?
  - Would it matter if vendors were legally liable?

Penetrate and Patch Fallacy

- **Fallacy:** If you keep patching software, eventually it will be secure
- Why is this a fallacy?
  - Empirical evidence to the contrary
  - Patches often add new flaws
  - Software is a moving target due to new versions, features, changing environment, new uses, etc.

Open vs Closed Source

- Open source software
  - The source code is available to user
  - For example, Linux
- Closed source
  - The source code is not available to user
  - For example, Windows
- What are the security implications?

Open Source Security

- Claimed advantages of open source is
  - **More eyeballs:** more people looking at the code should imply fewer flaws
  - A variant on Kerchoffs Principle
- Is this valid?
  - How many "eyeballs" looking for security flaws?
  - How many "eyeballs" focused on boring parts?
  - How many "eyeballs" belong to security experts?
  - Attackers can also look for flaws!
  - Evil coder might be able to insert a flaw

Open Source Security

- Open source example: wu-ftp
  - About 8,000 lines of code
  - A security-critical application
  - Was deployed and widely used
  - After 10 years, serious security flaws discovered!
- More generally, open source software has done little to reduce security flaws
- Why?
  - Open source follows penetrate and patch model!

Closed Source Security

- Claimed advantage of closed source
  - Security flaws not as visible to attacker
  - This is a form of "security by obscurity"
- Is this valid?
  - Many exploits do not require source code
  - Possible to analyze closed source code…
  - …though it is a lot of work!
  - Is "security by obscurity" real security?

Open vs Closed Source

- Advocates of open source often cite the **Microsoft fallacy** which states
  1. Microsoft makes bad software
  2. Microsoft software is closed source
  3. Therefore all closed source software is bad
- Why is this a fallacy?
  - Not logically correct
  - More relevant is the fact that Microsoft follows the penetrate and patch model

Open vs Closed Source

- No obvious security advantage to either open or closed source
- More significant than open vs closed source is software development practices
- Both open and closed source follow the "penetrate and patch" model

Open vs Closed Source

- If there is no security difference, why is Microsoft software attacked so often?
  - Microsoft is a big target!
  - Attacker wants most "bang for the buck"
- Few exploits against Mac OS X
  - **Not** because OS X is inherently more secure
  - An OS X attack would do less damage
  - Would bring less "glory" to attacker
- Next, we'll consider the theoretical differences between open and closed source
  - See Ross Anderson's paper

Security and Testing

- Can be shown that probability of a security failure after t units of testing is about

  $E = K/t$ where K is a constant
- This approximation holds over large range of t
- Then the "mean time between failures" is

  $MTBF = t/K$
- The good news: security improves with testing
- The bad news: security only improves **linearly** with testing!

Security and Testing

- The "mean time between failures" is approximately

  MTBF = t/K

- To have 1,000,000 hours between security failures, must test (on the order of) 1,000,000 hours!
- Suppose **open source** project has MTBF = t/K
- If flaws in **closed source** are twice as hard to find, do we then have MTBF = 2t/K ?
  - No! Testing is only half as effective as in the open source case, so MTBF = 2(t/2)/K = t/K
- The same result for open and closed source!

Security and Testing

- Closed source advocates might argue
  - Closed source has "open source" alpha testing, where flaws found at (higher) open source rate
  - Followed by closed source beta testing and use, giving attackers the (lower) closed source rate
  - Does this give closed source an advantage?
- Alpha testing is minor part of total testing
  - Recall, first to market advantage
  - Products rushed to market
- Probably no real advantage for closed source

Security and Testing

- No security difference between open and closed source?
- Provided that flaws are found "linearly"
- Is this valid?
  - Empirical results show security improves linearly with testing
  - Conventional wisdom is that this is the case for large and complex software systems

Security and Testing

- The fundamental problem
  - Good guys must find (almost) all flaws
  - Bad guy only needs 1 (exploitable) flaw
- Software reliability far more difficult in security than elsewhere
- How much more difficult?
  - See the next slide…

Security Testing: Do the Math

- Recall that MTBF = t/K
- Suppose 10^6 security flaws in some software
  - Say, Windows XP
- Suppose each bug has MTBF of 10^9 hours
- Expect to find 1 bug for every 10^3 hours testing
- Good guys spend 10^7 hours testing: **find 10^4 bugs**
  - Good guys have found 1% of all the bugs
- Bad guy spends 10^3 hours of testing: **finds 1 bug**
- Chance good guys found bad guy's bug is only **1%** !!!

Software Development

- General software development model
  - Specify
  - Design
  - Implement
  - Test
  - Review
  - Document
  - Manage
  - Maintain

Secure Software Development

- Goal: move away from "penetrate and patch"
- Penetrate and patch will always exist
  - But if more care taken in development, then fewer and less severe flaws to patch
- Secure software development not easy
- Much more time and effort required thru entire development process
- Today, little economic incentive for this!

Secure Software Development

- We briefly discuss the following
  - Design
  - Hazard analysis
  - Peer review
  - Testing
  - Configuration management
  - Postmortem for mistakes

Design

- Careful initial design
- Try to avoid high-level errors
  - Such errors may be impossible to correct later
  - Certainly costly to correct these errors later
- Verify assumptions, protocols, etc.
- Usually informal approach is used
- Formal methods
  - Possible to rigorously **prove** design is correct
  - In practice, only works in simple cases

Hazard Analysis

- Hazard analysis (or threat modeling)
  - Develop hazard list
  - List of what ifs
  - Schneier's "attack tree"
- Many formal approaches
  - Hazard and operability studies (HAZOP)
  - Failure modes and effective analysis (FMEA)
  - Fault tree analysis (FTA)

Peer Review

- Three levels of peer review
  - Review (informal)
  - Walk-through (semi-formal)
  - Inspection (formal)
- Each level of review is important
- Much evidence that peer review is effective
- Though programmers might not like it!

Levels of Testing

- Module testing — test each small section of code
- Component testing — test combinations of a few modules
- Unit testing — combine several components for testing
- Integration testing — put everything together and test

Types of Testing

- Function testing — verify that system functions as it is supposed to
- Performance testing — other requirements such as speed, resource use, etc.
- Acceptance testing — customer involved
- Installation testing — test at install time
- Regression testing — test after any change

Other Testing Issues

- Active fault detection
  - Don't wait for system to fail
  - Actively try to make it fail — attackers will!
- Fault injection
  - Insert faults into the process
  - Even if no obvious way for such a fault to occur
- Bug injection
  - Insert bugs into code
  - See how many of injected bugs are found
  - Can use this to estimate number of bugs
  - Assumes injected bugs similar to unknown bugs

Testing Case History

- In one system with 184,000 lines of code
- Flaws found
  - 17.3% inspecting system design
  - 19.1% inspecting component design
  - 15.1% code inspection
  - 29.4% integration testing
  - 16.6% system and regression testing
- Conclusion: must do many kinds of testing
  - Overlapping testing is necessary
  - Provides a form of "defense in depth"

Security Testing: The Bottom Line

- **Security testing** is far more demanding than non-security testing
- Non-security testing — does system do what it is supposed to?
- Security testing — does system do what it is supposed to **and nothing more**?
- Usually impossible to do exhaustive testing
- How much testing is enough?

Security Testing: The Bottom Line

- How much testing is enough?
- Recall MTBF = t/K
- Seems to imply testing is nearly hopeless!
- But there is some hope…
  - If we can eliminate an entire class of flaws then statistical model breaks down
  - For example, if we have a single test (or a few tests) to eliminate all buffer overflows

Configuration Issues

- Types of changes
  - Minor changes — maintain daily functioning
  - Adaptive changes — modifications
  - Perfective changes — improvements
  - Preventive changes — no loss of performance
- Any change can introduce new flaws!

Postmortem

- After fixing any security flaw…
- Carefully analyze the flaw
- To learn from a mistake
  - Mistake must be analyzed and understood
  - Must make effort to avoid repeating mistake
- In security, **always** learn more when things go wrong than when they go right
- Postmortem may be the most under-used tool in all of security engineering!

Software Security

- First to market advantage
  - Also known as "network economics"
  - Security suffers as a result
  - Little economic incentive for secure software!
- **Penetrate and patch**
  - Fix code as security flaws are found
  - Fix can result in worse problems
  - Mostly done **after** code delivered
- Proper development can reduce flaws
  - But costly and time-consuming

Software and Security

- Even with best development practices, security flaws will still exist
- Absolute security is (almost) never possible
- So, it is not surprising that absolute software security is impossible
- The goal is to minimize and manage risks of software flaws
- Do not expect dramatic improvements in consumer software security anytime soon!

Operating Systems and Security

OS Security

- OSs are large, complex programs
  - Many bugs in any such program
  - We have seen that bugs can be security threats
- Here we are concerned with security provided by OS
  - Not concerned with threat of bad OS software
- Concerned with OS as security **enforcer**
- In this section we only scratch the surface

OS Security Challenges

- Modern OS is **multi-user** and **multi-tasking**
- OS must deal with
  - Memory
  - I/O devices (disk, printer, etc.)
  - Programs, threads
  - Network issues
  - Data, etc.
- OS must protect processes from other processes and users from other users
  - Whether accidental or malicious

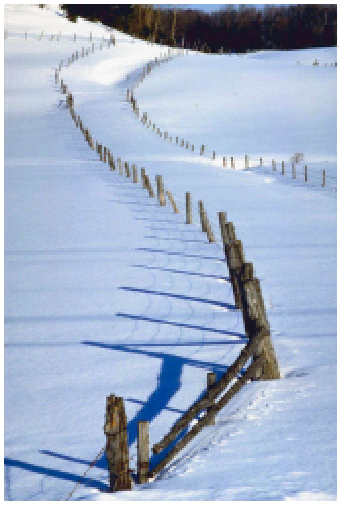
OS Security Functions

- Memory protection
  - Protect memory from users/processes
- File protection
  - Protect user and system resources
- Authentication
  - Determines and enforce authentication results
- Authorization
  - Determine and enforces access control

Memory Protection

- Fundamental problem
  - How to keep users/processes separate?
- Separation
  - Physical separation — separate devices
  - Temporal separation — one at a time
  - Logical separation — sandboxing, etc.
  - Cryptographic separation — make information unintelligible to outsider
  - Or any combination of the above

Memory Protection



- **Fence** — users cannot cross a specified address
  - Static fence — fixed size OS
  - Dynamic fence — fence register
- Base/bounds register — lower and upper address limit
- Assumes contiguous space

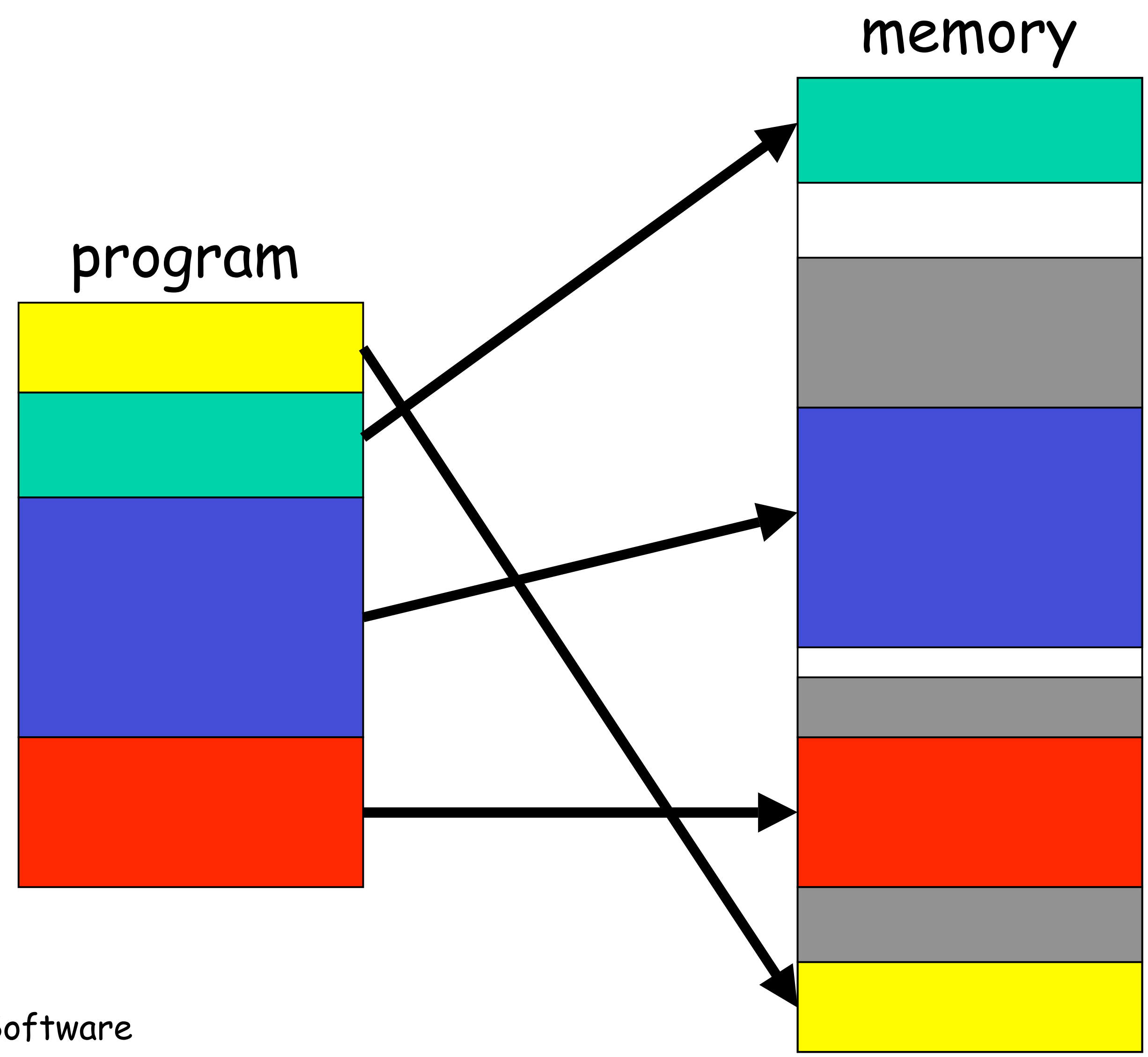
Memory Protection

- Tagging — specify protection of each address
  - + Extremely fine-grained protection
  - - High overhead — can be reduced by tagging sections instead of individual addresses
  - - Compatibility
- More common is segmentation and/or paging
  - Protection is not as flexible
  - But much more efficient

Segmentation

- Divide memory into logical units, such as
  - Single procedure
  - Data in one array, etc.
- Can enforce different access restrictions on different segments
- Any segment can be placed in any memory location (if location is large enough)
- OS keeps track of actual locations

Segmentation



Segmentation

- OS can place segments anywhere
- OS keeps track of segment locations as <segment,offset>
- Segments can be moved in memory
- Segments can move out of memory
- All address references go thru OS

Segmentation Advantages

- Every address reference can be checked
  - Possible to achieve **complete mediation**
- Different protection can be applied to different segments
- Users can share access to segments
- Specific users can be restricted to specific segments

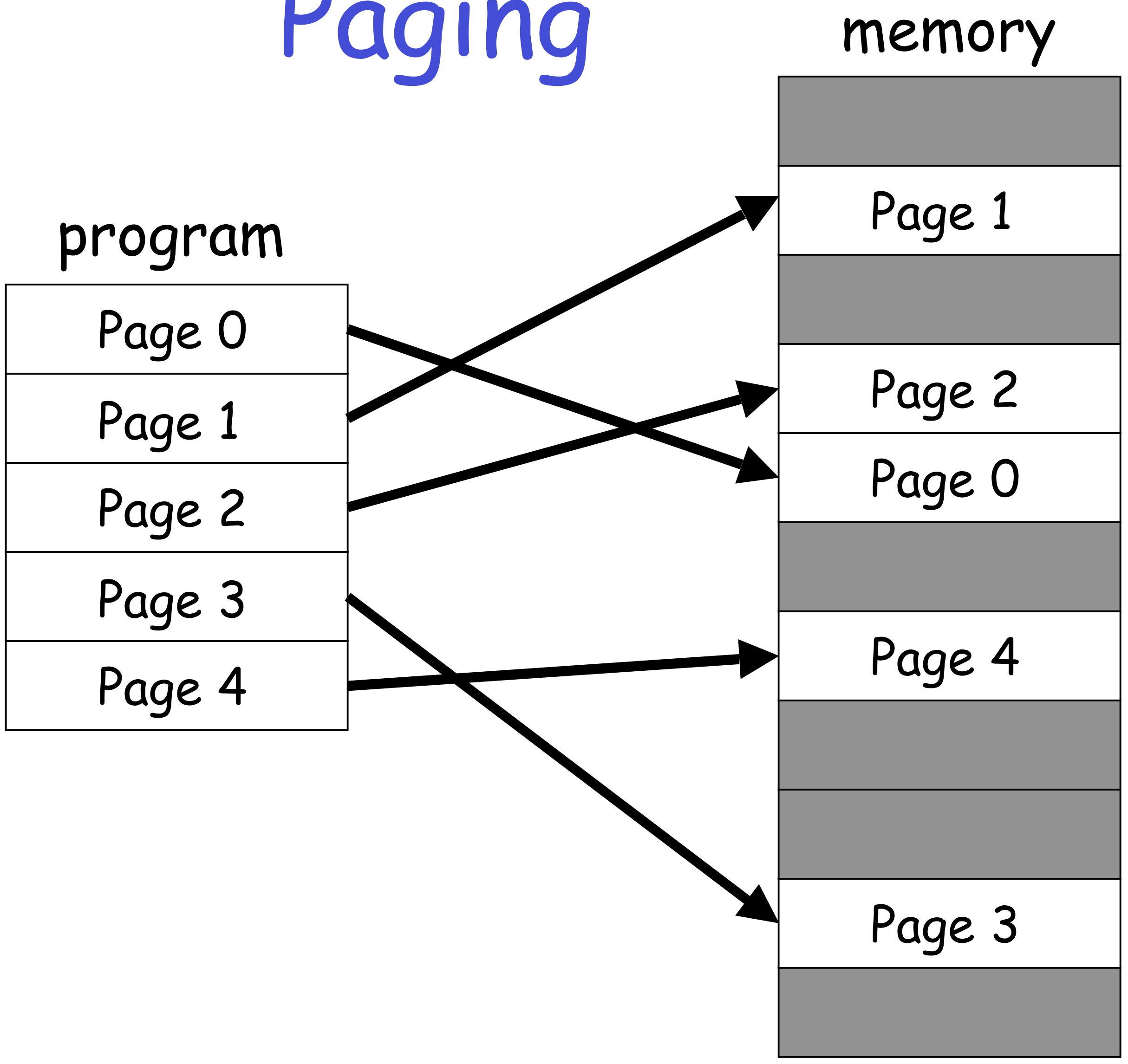
Segmentation Disadvantages

- How to reference <segment,offset> ?
  - OS must know segment **size** to verify access is within segment
  - But some segments can grow during execution (for example, dynamic memory allocation)
  - OS must keep track of **variable** segment sizes
- Memory fragmentation is also a problem
  - Compacting memory changes tables
- A lot of work for the OS
- More complex ⇒ more chance for mistakes

Paging

- Like segmentation, but fixed-size segments
- Access via <page,offset>
- Plusses and minuses
  - + Avoids fragmentation, improved efficiency
  - + OS need not keep track of variable segment sizes
  - - No logical unity to pages
  - - What protection to apply to a given page?

Paging


memory
program
Page 0
Page 1
Page 2
Page 3
Page 4
Page 1
Page 2
Page 0
Page 4
Page 3

Other OS Security Functions

- OS must enforce access control
- Authentication
  - Passwords, biometrics
  - Single sign-on, etc.
- Authorization
  - ACL
  - Capabilities
- These topics discussed previously
- OS is an attractive target for attack!

Trusted Operating System

Trusted Operating System

- An OS is **trusted** if we rely on it for
  - Memory protection
  - File protection
  - Authentication
  - Authorization
- Every OS does these things
- But if a trusted OS fails to provide these, our security fails

Trust vs Security

- **Trust** implies reliance
- Trust is binary
- Ideally, only trust secure systems
- All trust relationships should be explicit

- **Security** is a judgment of effectiveness
- Judged based on specified policy
- Security depends on trust relationships

- Note: Some authors use different terminology!

Trusted Operating Systems

- **Trust** implies reliance
- A trusted system is relied on for security
- An untrusted system is not relied on for security
- If all untrusted systems are compromised, your security is unaffected
- Ironically, **only a trusted system can break your security!**

Trusted OS

- OS mediates interactions between subjects (users) and objects (resources)
- Trusted OS must decide
  - Which objects to protect and how
  - Which subjects are allowed to do what

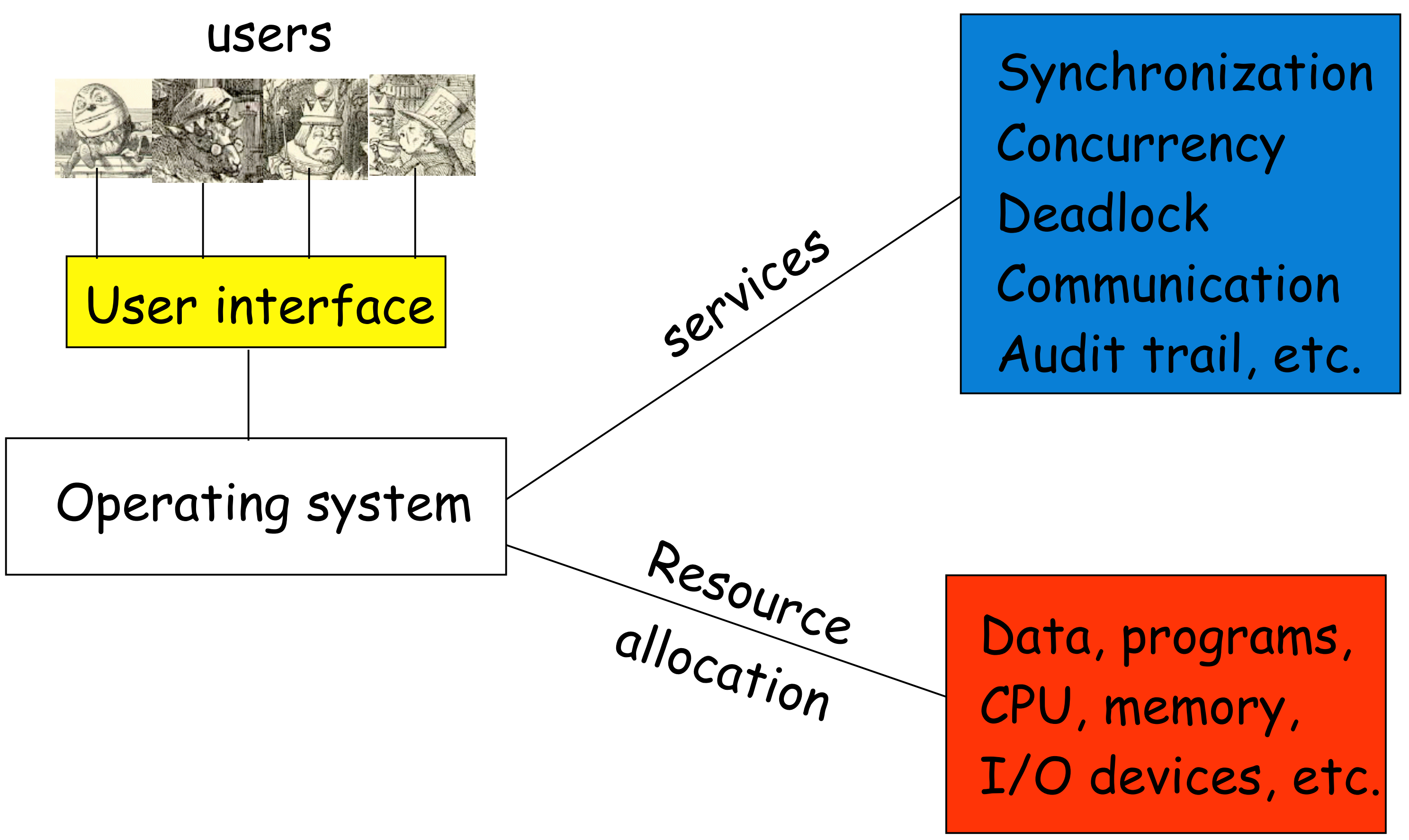
General Security Principles

- Least privilege — like "low watermark"
- Simplicity
- Open design (Kerchoffs Principle)
- Complete mediation
- White listing (preferable to black listing)
- Separation
- Ease of use
- But commercial OSs emphasize features
  - Results in complexity and poor security

OS Security

- Any OS must provide some degree of
  - Authentication
  - Authorization (users, devices and data)
  - Memory protection
  - Sharing
  - Fairness
  - Inter-process communication/synchronization
  - OS protection

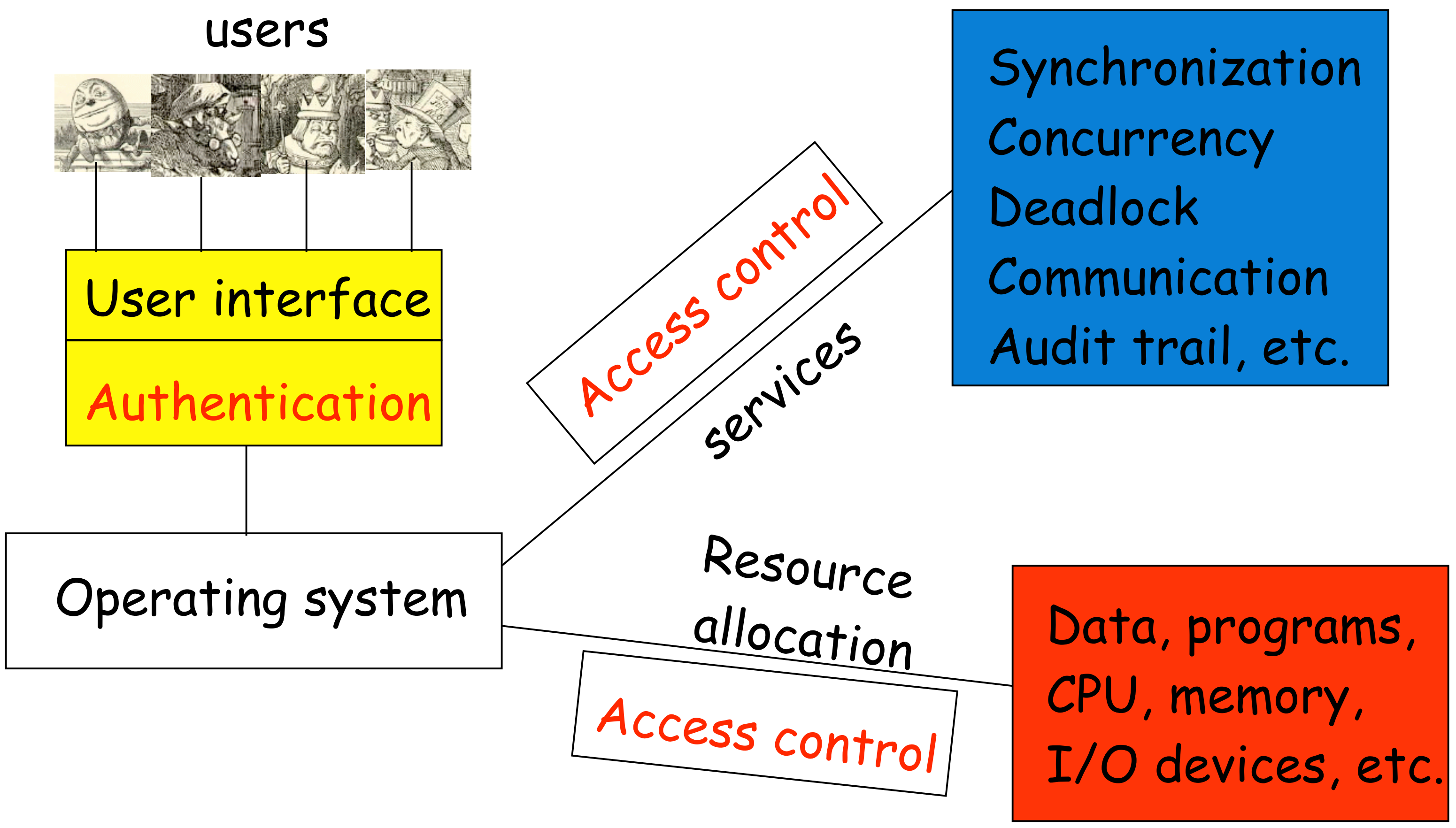
OS Services


users
User interface
Operating system
services
Synchronization
Concurrency
Deadlock
Communication
Audit trail, etc.
Resource
allocation
Data, programs,
CPU, memory,
I/O devices, etc.

Trusted OS

- A trusted OS also provides some or all of
  - User authentication/authorization
  - Mandatory access control (**MAC**)
  - Discretionary access control (**DAC**)
  - Object reuse protection
  - Complete mediation — access control
  - Trusted path
  - Audit/logs

Trusted OS Services



MAC and DAC

- Mandatory Access Control (MAC)
  - Access not controlled by owner of object
  - Example: User does not decide who holds a **TOP SECRET** clearance
- Discretionary Access Control (DAC)
  - Owner of object determines access
  - Example: UNIX/Windows file protection
- If DAC and MAC both apply, MAC wins

Object Reuse Protection

- OS must prevent leaking of info
- Example
  - User creates a file
  - Space allocated on disk
  - But same space previously used
  - "Leftover" bits could leak information
  - Magnetic remanence is a related issue

Trusted Path

- Suppose you type in your password
  - What happens to the password?
- Depends on the software!
- How can you be sure software is not evil?
- Trusted path problem

"I don't know how to to be confident even of a digital signature I make on my own PC, and I've worked in security for over fifteen years. Checking all of the software in the critical path between the display and the signature software is way beyond my patience. "

— Ross Anderson

Audit

- System should log security-related events
- Necessary for postmortem
- What to log?
  - Everything? Who (or what) will look at it?
  - Don't want to overwhelm administrator
  - Needle in haystack problem
- Should we log incorrect passwords?
  - "Almost" passwords in log file?
- Logging is not a trivial matter

Security Kernel

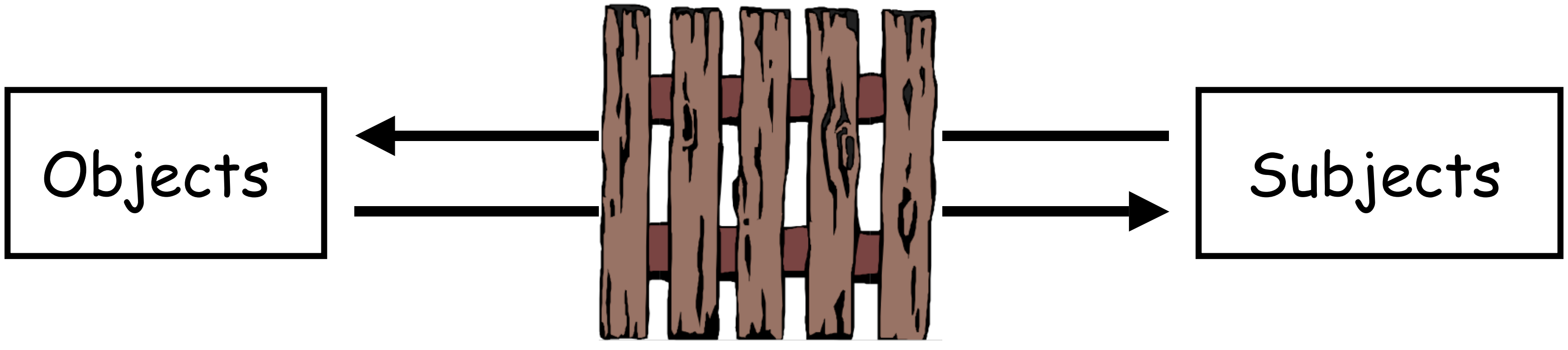
- **Kernel** is the lowest-level part of the OS
- Kernel is responsible for
  - Synchronization
  - Inter-process communication
  - Message passing
  - Interrupt handling
- The **security kernel** is the part of the kernel that deals with security
- Security kernel contained within the kernel

Security Kernel

- Why have a security kernel?
- All accesses go thru kernel
  - Ideal place for access control
- Security-critical functions in one location
  - Easier to analyze and test
  - Easier to modify
- More difficult for attacker to get in "below" security functions

Reference Monitor

- The part of the security kernel that deals with access control
  - Mediates access of subjects to objects
  - Tamper-resistant
  - Analyzable (small, simple, etc.)



Reference monitor

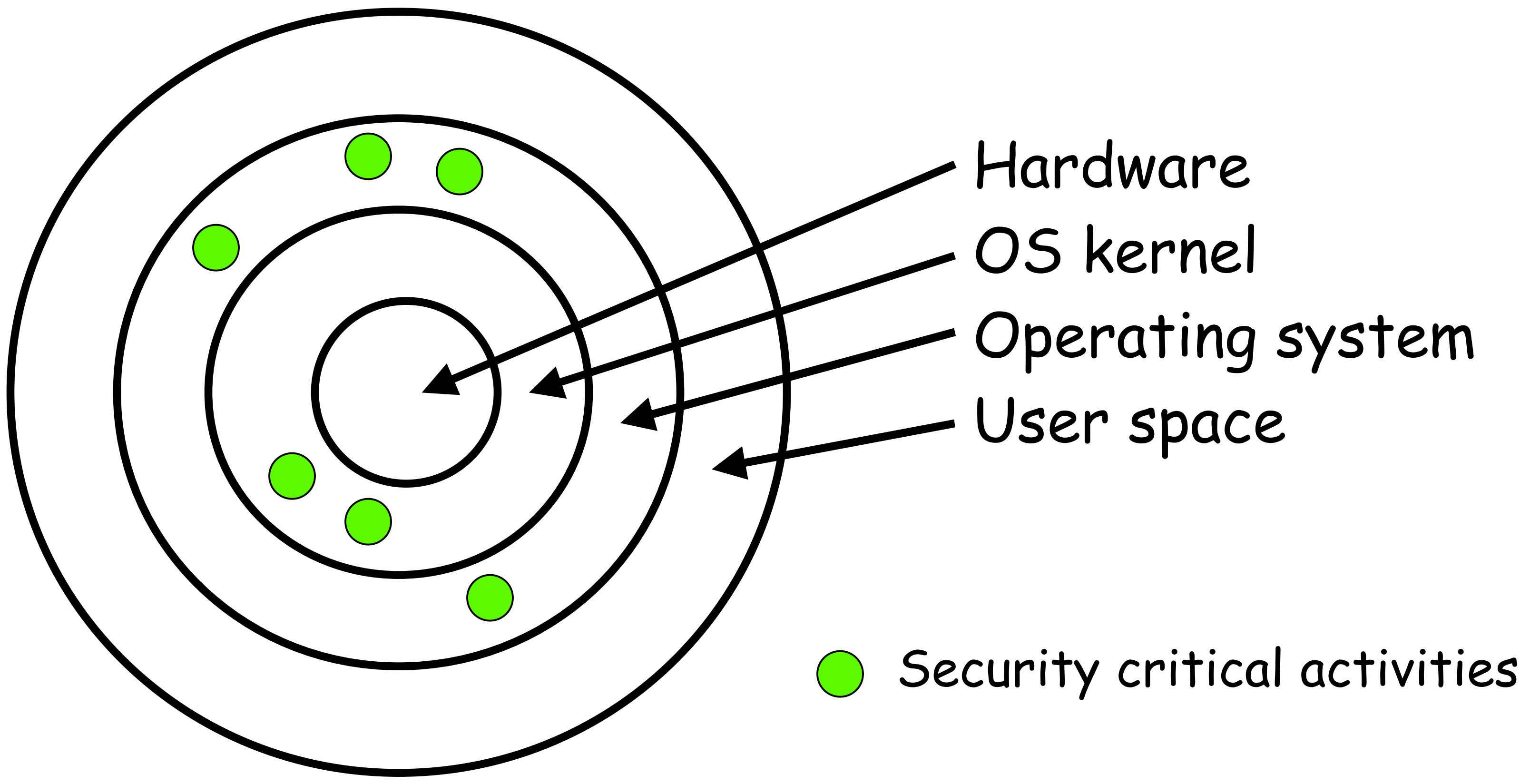
Trusted Computing Base

- **TCB** — everything in the OS that we rely on to enforce security
- If everything outside TCB is subverted, trusted OS would still be trusted
- TCB protects users from each other
  - Context switching between users
  - Shared processes
  - Memory protection for users
  - I/O operations, etc.

TCB Implementation

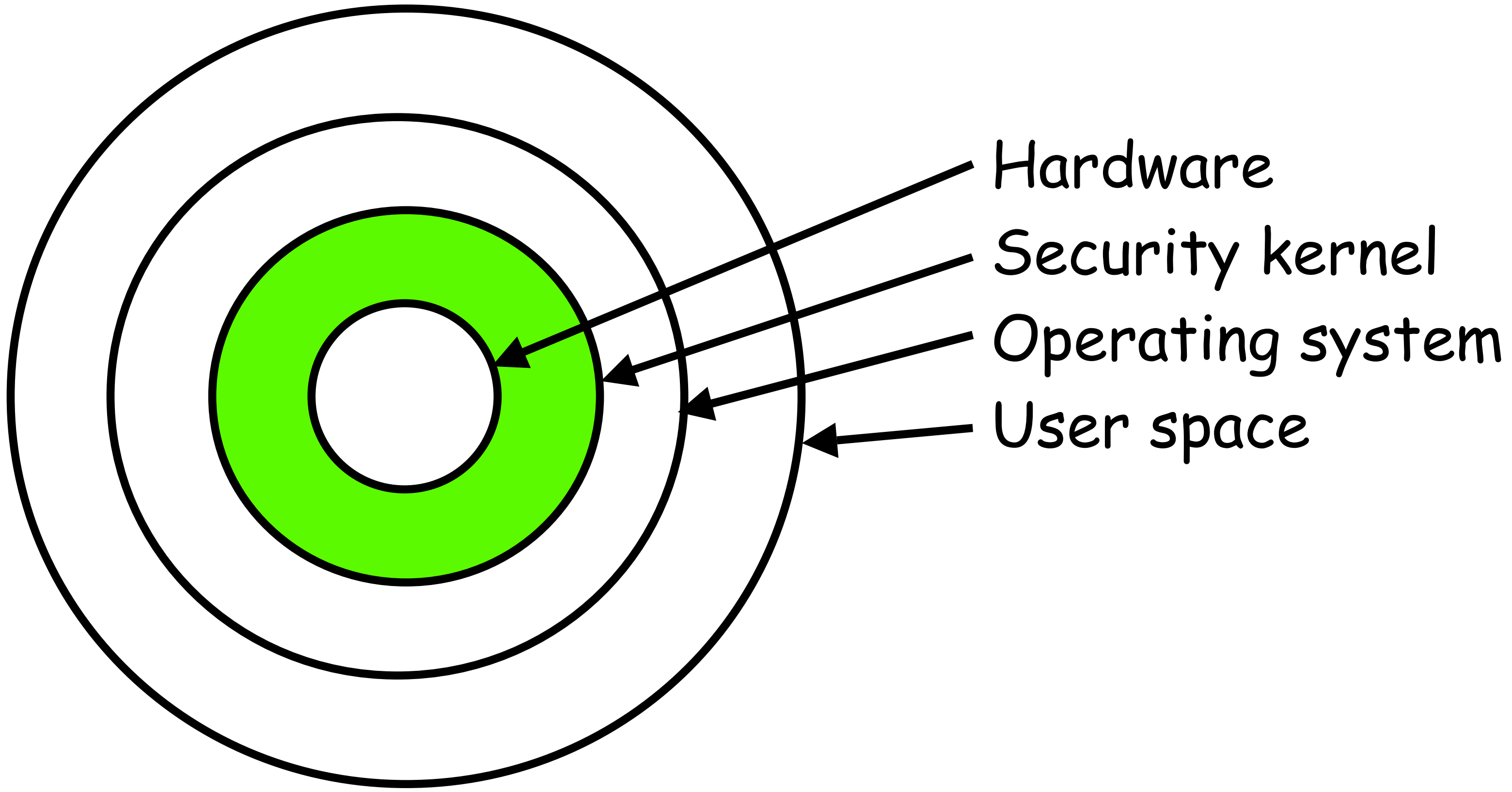
- Security may occur many places within OS
- Ideally, design security kernel first, and build the OS around it
  - Reality is usually the other way around
- Example of a trusted OS: **SCOMP**
  - Developed by Honeywell
  - Less than 10,000 LOC in SCOMP security kernel
  - Win XP has 40,000,000 lines of code!

Poor TCB Design



Problem: No clear security **layer**

Better TCB Design



Security kernel is **the** security layer

Trusted OS Summary

- Trust implies reliance
- TCB (trusted computing base) is everything in OS we rely on for security
- If everything outside TCB is subverted, we still have trusted system
- If TCB subverted, security is broken

$$\begin{tikzcd}
\text{OS} \arrow[d] \\
\text{OS Kernel} \arrow[d] \\
\text{Security Kernel}
\end{tikzcd}$$

NGSCB

Next Generation Secure Computing Base

- NGSCB pronounced "n scub" (the G is silent)
- Will be part of **Microsoft**'s **Longhorn OS**
- TCG (Trusted Computing Group)
  - Led by Intel, TCG makes special hardware
- NGSCB is the part of Windows that will interface with TCG hardware
- TCG/NGSCB formerly TCPA/Palladium
  - Why the name changes?

NGSCB

- The original motivation for TCPA/Palladium was digital rights management (DRM)
- Today, TCG/NGSCB is promoted as general security-enhancing technology
  - DRM just one of many potential applications
- Depending on who you ask, TCG/NGSCB is
  - Trusted computing
  - Treacherous computing

Motivation for TCG/NGSCB

- **Closed systems:** Game consoles, smartcards, etc.
  - Good at protecting secrets (tamper resistant)
  - Good at forcing people to pay
  - Limited flexibility
- **Open systems:** PCs
  - Incredible flexibility
  - Poor at protecting secrets
  - Very poor at defending their own software
- TCG goal is to provide closed system security benefits on an open platform
- "A virtual set-top box inside your PC" — Rivest

TCG/NGSCB

- TCG provides tamper-resistant hardware
  - Secure place to store cryptographic key
  - Key (or other secret) secure even from a user with full admin privileges!
- TCG hardware is in addition to ordinary hardware, **not** in place of it
- PC has two OSs — usual OS and special trusted OS to deal with TCG hardware
- NGSCB is Microsoft's trusted OS

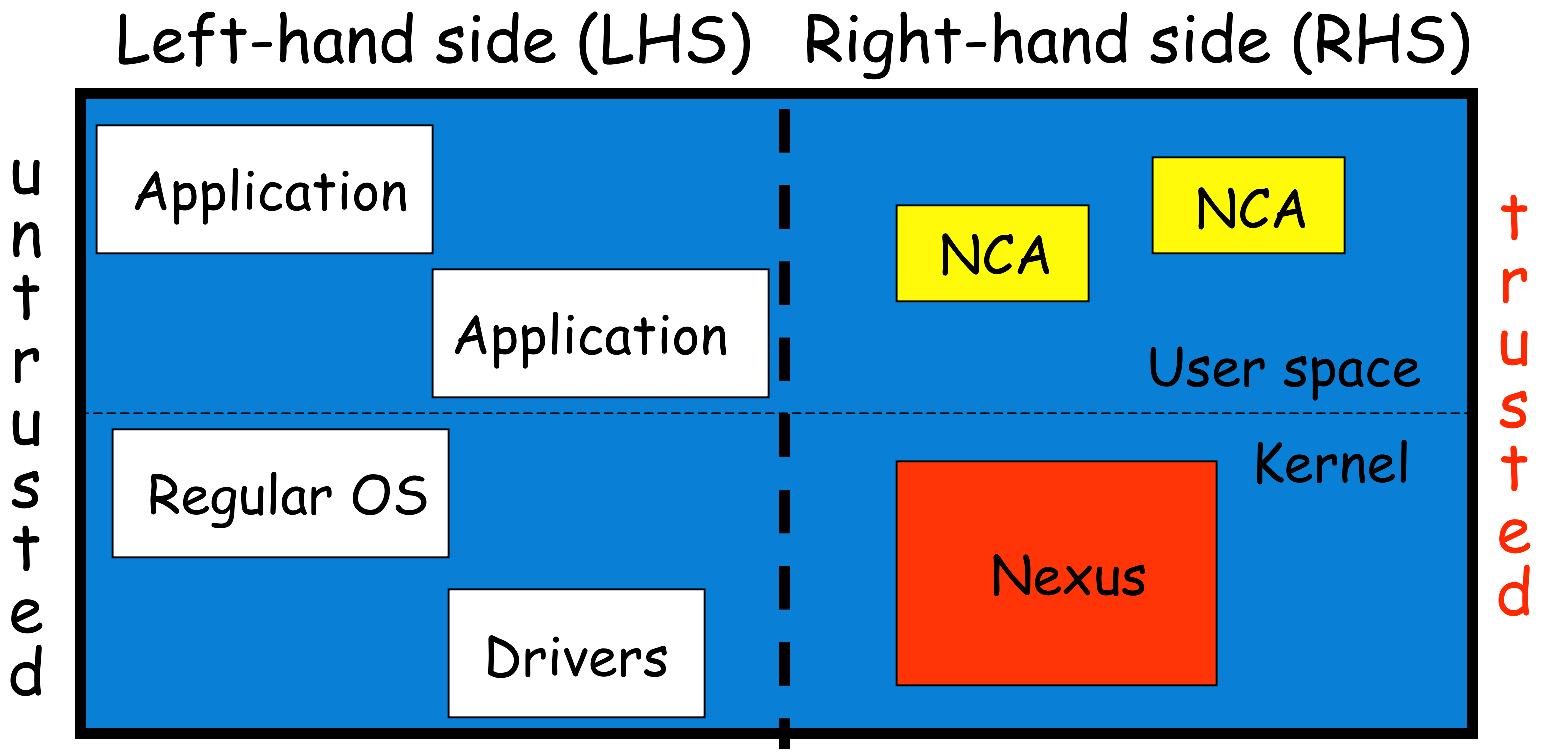
NGSCB Design Goals

- Provide high **assurance**
  - High confidence that system behaves correctly
  - Correct behavior even if system is under attack
- Provide **authenticated** operation
  - Authenticate "things" (software, devices, etc.)
- Protection against hardware tampering is not a design goal of NGSCB
  - Hardware tampering is the domain of TCG

NGSCB Disclaimer

- Specific details are sketchy
- Based on available info, Microsoft has not resolved all of the details
- What follows: author's best guesses
- This should all become much clearer in the not-too-distant future

NGSCB Architecture



- **Nexus** is the Trusted Computing Base in NGSCB
- The **NCA** (Nexus Computing Agents) talk to Nexus and LHS

NGSCB

- NGSCB “feature groups”
  1. **Strong process isolation**
     - Processes do not interfere with each other
  2. **Sealed storage**
     - Data protected (tamper resistant hardware)
  3. **Secure path**
     - Data to and from I/O protected
  4. **Attestation**
     - “Things” securely authenticated
     - Allows TCB to be extended via NCAs
- 1.,2. and 3. aimed at malicious code
- 4. provides for (secure) extensibility

NGSCB Process Isolation

- **Curtained memory**
- Process isolation and the OS
    - Protect trusted OS (Nexus) from untrusted OS
    - Isolate trusted OS from untrusted stuff
- Process isolation and NCAs
    - NCAs isolated from software they do not trust
- Trust determined by users, to an extent…
    - User **can** disable a trusted NCA
    - User **cannot** enable an untrusted NCA

NGSCB Sealed Storage

- Sealed storage contains **secret** data
  - If **code X** wants access to secret, a hash of X must be verified (integrity check of X)
  - Implemented via symmetric key cryptography
- Confidentiality of secret is protected since only accessed by trusted software
- Integrity of secret is assured since it's in sealed storage

NGSCB Secure Path

- Secure path for input
  - From keyboard to Nexus
  - From mouse to Nexus
- Secure path for output
  - From Nexus to the screen
- Uses crypto
  - Digital signatures

NGSCB Attestation (1)

- Secure authentication of **things**
  - Authenticate devices, services, code, etc.
  - Separate from user authentication
- Public key cryptography used
  - Certified key pair required
  - Private key not user-accessible
  - Sign and send result to remote system
- **TCB extended via attestation of NCAs**
  - This is a major feature!

NGSCB Attestation (2)

- Public key used for attestation
  - However, public key reveals the user identity
  - Anonymity is lost
- Trusted third party (TTP) can be used
  - TTP verifies signature
  - Then TTP vouches for signature to recipient
  - Anonymity preserved (except to TTP)
- Support for zero knowledge proofs
  - Verify knowledge of a secret without revealing it
  - Anonymity "preserved unconditionally"

NGSCB Compelling Apps (1)

- Type a Word document in Windows
- Move document to RHS
  - Trusted area
- Read document carefully
- Digitally sign the document
- "What you see is what you sign"
  - Virtually impossible to assure this on your PC!

NGSCB Compelling Apps (2)

- Digital Rights Management (DRM)
- DRM problems solved by NGSCB
  - **Protect secret** — sealed storage
    - Impossible without something like NGSCB
  - **Scraping data** — secure path
    - Impossible to prevent without something like NGSCB
  - **Positively ID users**
    - Higher assurance with NGSCB

NGSCB According to Microsoft

- Everything in regular Windows must still work in LHS (untrusted side) of NGSCB'ed system
- User is in charge of
  - Which Nexuses will run on system
  - Which NCAs will run on system
  - Which NCAs allowed to identify system, etc.
- No external process can enable Nexus or NCA
- Nexus does not block, delete or censor any data (NCA **does**, but NCAs must be **authorized** by user)
- Nexus is open source

NGSCB Critics

- There are **many** critics — we consider two
- Ross Anderson
  - Perhaps the most influential critic
  - One of the harshest critics
- Clark Thomborson
  - Lesser-known critic
  - Criticism strikes at heart of NGSCB

Anderson's NGSCB Criticism (1)

- Digital object controlled by its creator, not user of machine where it resides: Why?
  - Creator can specify the NCA
  - If user does not accept NCA, access is denied
  - Aside: Such control is good in, say, MLS apps
- Spse Microsoft Word encrypts all documents with key only available to Microsoft products
  - Difficult to stop using Microsoft products!

Anderson's NGSCB Criticism (2)

- Files from a compromised machine could be blacklisted to, say, prevent music piracy
- Suppose everyone at SJSU uses same copy of Microsoft Word
  - If you stop this copy from working on all NGSCB machines, SJSU users won't use NGSCB
  - Instead, make all NGSCB machines refuse to open documents created with this instance of Word
  - SJSU users can't share docs with any NGSCB user!

Anderson's NGSCB Criticism (3)

- Going off the deep end?
  - "The Soviet Union tried to register and control all typewriters. NGSCB attempts to register and control all computers."
  - "In 2010 President Clinton may have two red buttons on her desk — one that sends missiles to China and another that turns off all of the PCs in China…"

Thomborson's NGSCB Criticism

- NGSCB acts like a **security guard**
- By passive observation, NGSCB "security guard" sees sensitive information
- How can a user know NGSCB is not spying on them?
- According to Microsoft
  - Nexus software will be public
  - NCAs can be debugged (required for app development)
  - NGSCB is strictly "opt in"
- Loophole?
  - Release version of NCA can't be debugged **and** debug and release versions have different hash values!

NGSCB Bottom Line (1)

- TCG/NGCSB embeds a **trusted OS** within an open platform
- Without something similar, PC may lose out
  - Particularly in entertainment-related areas
  - Copyright holders won't trust PC
- With NGSCB it is often claimed that users will lose control over their PCs
- But users must choose to "opt in"
  - If user does not opt in, what has been lost?

NGSCB Bottom Line (2)

- NGSCB is a **trusted system**
- **Only trusted system can break security**
  - By definition, an untrusted system is not trusted with security critical tasks
  - Also by definition, a trusted system is trusted with security critical tasks
  - If untrusted system is compromised, security is not at risk
  - If trusted system is compromised (or malfunctions), security is at risk

Software Summary

- Software flaws
  - Buffer overflow
  - Race conditions
  - Incomplete mediation
- Malware
  - Viruses, worms, etc.
- Other software-based attacks

Software Summary

- Software Reverse Engineering (SRE)
- Digital Rights Management (DRM)
- Secure software development
  - Penetrate and patch
  - Open vs closed source
  - Testing

Software Summary

- Operating systems and security
  - How does OS enforce security?
- Trusted OS design principles
- Microsoft's NGSCB
  - A trusted OS for DRM

Course Summary

- Crypto
  - Symmetric key, public key, hash functions, cryptanalysis
- Access Control
  - Authentication, authorization
- Protocols
  - Simple auth., SSL, IPSec, Kerberos, GSM
- Software
  - Flaws, malware, SRE, Software development, trusted OS