홍익대학교
HONGIK UNIVERSITY

Network Security

<CH 11>

Youn Kyu Lee
Hongik University

Software Reverse Engineering (SRE)

SRE – 1/2

- **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 – 2/2

- 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 Tools – 1/2

- 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 – 2/2

- **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

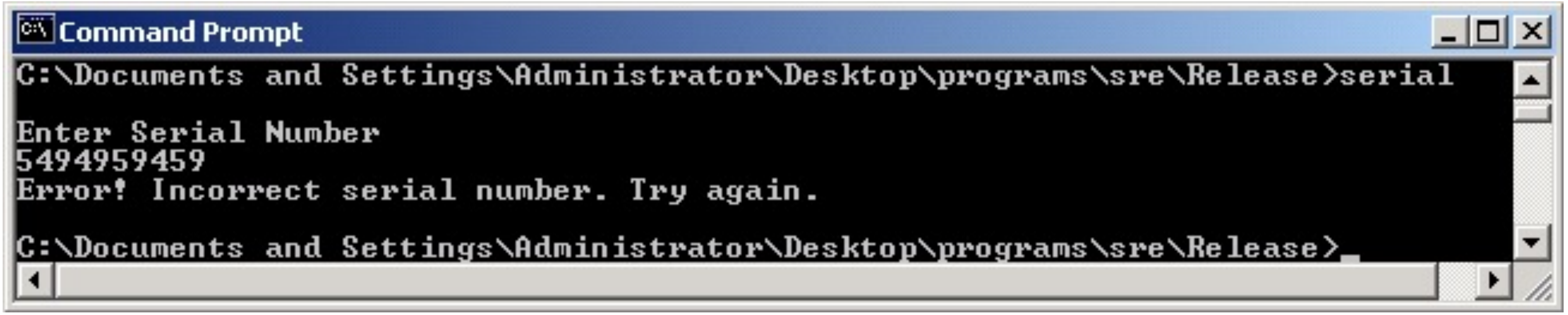
- Working knowledge of target assembly code
- Experience with the tools
  - IDA Pro — sophisticated and complex
  - SoftICE — large two-volume users manual
- Knowledge of Windows **Portable Executable** (PE) file format
- Boundless patience and optimism
- SRE is tedious and labor-intensive process!

SRE Example - 1/10

- 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

SRE Example - 2/10

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



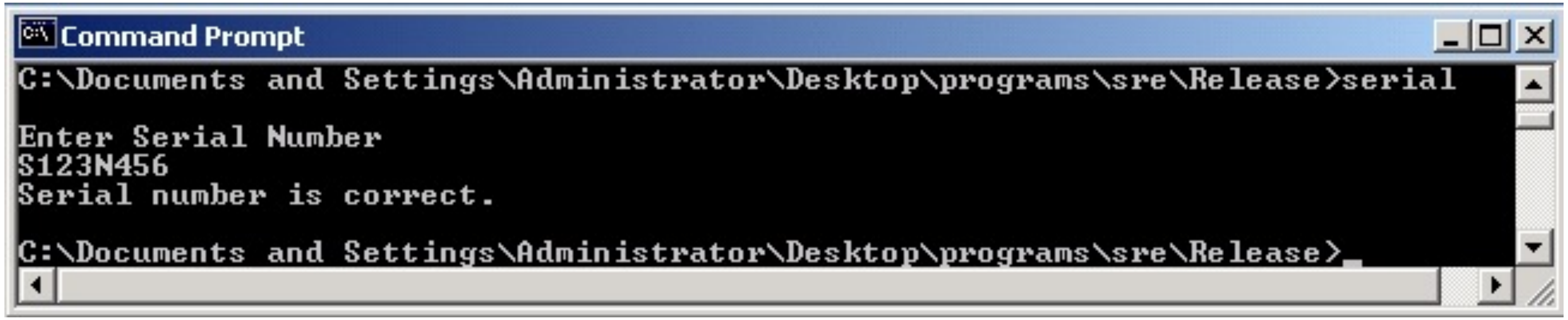
SRE Example - 3/10

- IDA Pro disassembly
- Looks like serial number is S123N456

```
.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
```

SRE Example - 4/10

- Try the serial number S123N456



SRE Example - 5/10

- 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
```

```
.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 - 6/10

- 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?

```
.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
```

SRE Example - 7/10

- 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
.text:00401034                push    offset aErrorIncorrect ; "Error! Incorrect serial number.
.text:00401039                call    sub_4010AF
```

← jz always true!!!

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

SRE Example - 8/10

- 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 - 9/10

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

```
Command Prompt
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>_
```

SRE Example - 10/10

- 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
  - 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

Example of “anti disassembly”

- Suppose actual code instructions are

inst 1	jmp	junk	inst 3	inst 4	…

- What the disassembler sees

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

Anti-debugging

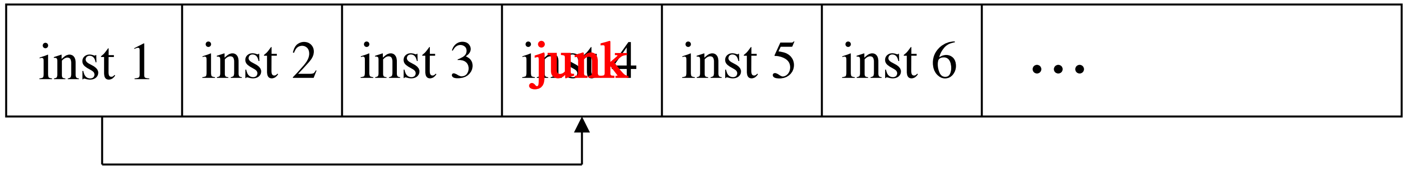
- Monitor for
  - Use of debug registers
  - Inserted breakpoints
- Debuggers don’t handle threads well
  - Interacting threads may confuse debugger
- Many other debugger-unfriendly tricks
- Undetectable debugger possible in principle
  - Hardware-based debugging (HardICE) is possible

Anti-debugger Example-1/3

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 in CPU
  - This is done to increase efficiency
- Suppose when debugger executes inst 1, it does **not** pre-fetch instructions in CPU
- Can we use this difference to confuse the debugger?

Anti-debugger Example-2/3



- 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 at the CPU
- And execute inst1 then inst 4 in memory will be junk, but the inst4 fetched at CPU is not changed

Anti-debugger Example-2/3

- But when debugger executes inst1, it does not prefetch instructions
- So, 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! (junk code will be problem)
- Also, code is very platform-dependent
- Again, clever attacker will figure this out!

Tamper-resistance (Guards)

- 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 – 1/4

- 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 – 2/4

- Code obfuscation sometimes promoted as a powerful security technique
- Recently it has been shown that obfuscation probably cannot provide strong security
  - On the (im)possibility of obfuscating programs
- Obfuscation might still have practical uses!
  - Even if it can never be as strong as crypto

Code Obfuscation – 3/4

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

Code Obfuscation – 4/4

- 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 – 1/5

- 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 – 2/5

- 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 – 3/5

- 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

Metamorphic Software – 4/5

- Suppose 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 can exist in all instances
  - But a specific buffer overflow attack will only work against **some** instances
  - Buffer overflow attacks are delicate!

Metamorphic Software – 5/5

- 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 (DRM)

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?

- DRM is an attempt to provide "Remote Control" over digital content
- "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.

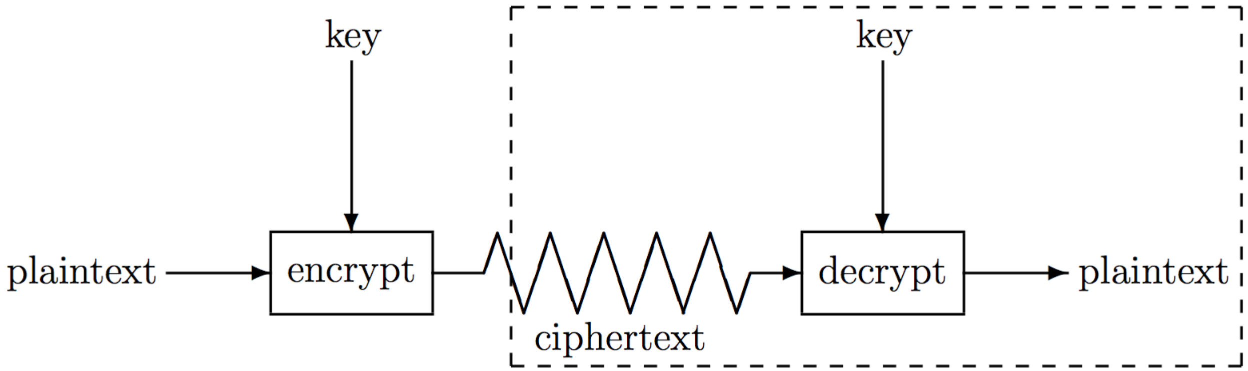
Persistent Protection

- "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?
- Tamper-resistant hardware?
  - Closed systems: Game Cube, etc.
  - Open systems: TCG/NGSCB for PCs

Is Crypto the Answer?–1/2



- 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?–2/2

- 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
  - So, want something that "works" in practice
  - And what works depends on context
- DRM is not strictly a technical problem!

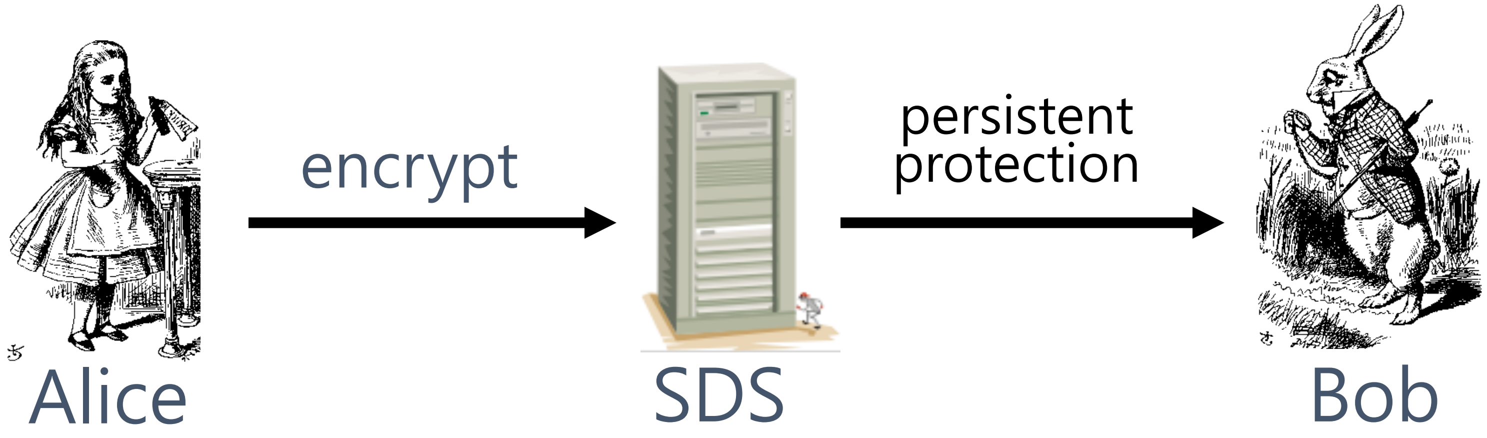
Software-based DRM

- 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

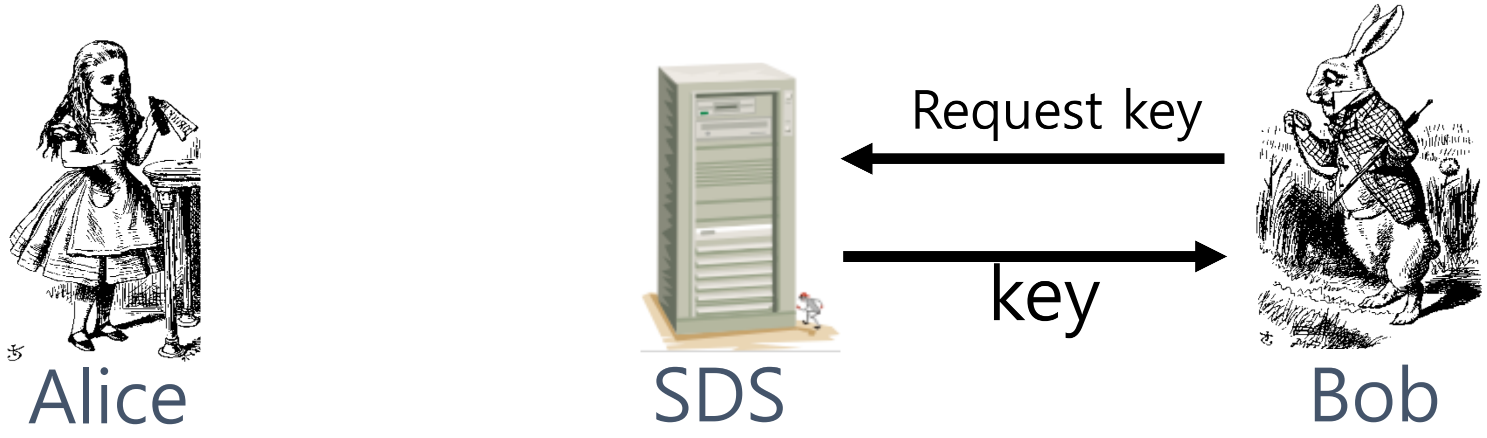
- Based on design of MediaSnap, Inc., a small Silicon Valley startup company
- Developed a DRM system
  - Designed to protect PDF documents
- Two parts to the system
  - Server — Secure Document Server (SDS)
  - Client — PDF Reader "plugin" software

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

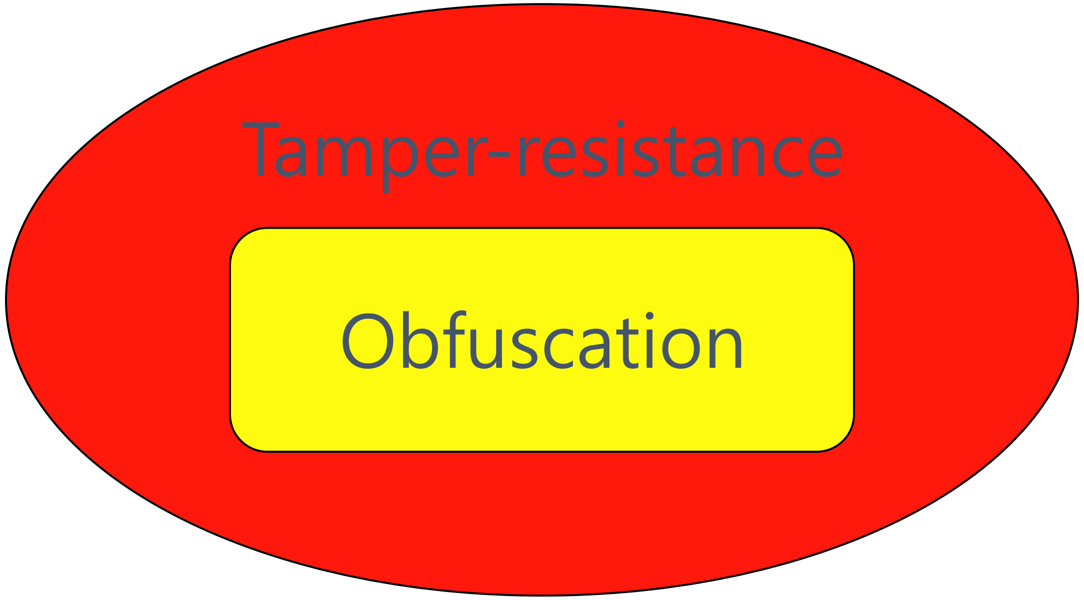


- Bob authenticates to SDS
- Bob requests key from SDS
- Bob can then access document, but only thru special DRM software

Security Issues

- Server side (SDS)
  - Protect keys, authenticate user, etc.
  - Apply persistent protection
- Client side (PDF plugin)
  - Protect keys, authenticate user, etc.
  - Enforce persistent protection

Security Overview (client side)



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

Tamper-Resistance

Anti-debugger $\rightleftarrows$ Encrypted code

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

Other Security Features

- 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

- MediaSnap DRM system employed nearly all known S/W protection techniques
- But Code “fragilization” (guard) is not employed
  - Since it is not compatible with encrypted executable code
- 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

- De-scrambling in device driver

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: $\mathrm{LIST} = \{12,45,2,37,23,31\}$
- The LIST is stored on client, encrypted with server's key:

Server-side Scrambling

- On server side

data → **scrambled data** → encrypted scrambled data

- 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

Alice
(client)

E(LIST, K_{server}) →

← E(m,K)

← scramble (encrypted) data using Alice's m-th algorithm

Bob
(server)

- The key K is a session key
- The LIST is unreadable by client

Client-side De-scrambling

- On client side

encrypted scrambled data → **scrambled data** → **data**

- Try to keep plaintext away from potential attacker
- "Proprietary" device driver
  - (a "scrambling" algorithm is essentially a proprietary cipher)
  - 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 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
- Protections 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 SW 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? 1/2

- 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
- Sometimes called "network economics"

Why Penetrate and Patch? 2/2

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

Open Source Security – 1/2

- 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 – 2/2

- 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

Security and Testing - 1/6

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

  $$E = K/t \qquad \text{where } K \text{ is a constant}$$

- 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 - 2/6

- To have 1,000,000 hours between security failures, must test (on the order of) 1,000,000 hours!
- Suppose open source project has $\mathrm{MTBF} = t/K$
- If flaws in closed source are twice as hard to find, **do we then have**

$$\mathbf{MTBF} = 2(t/K)\ ?$$

Security and Testing - 3/6

- **No!**
  Closed source testing is only half as effective as in the open source case, so

  $$\mathrm{MTBF} = 2(t/2)/K = t/K$$

- The same result for open and closed source!

Security and Testing - 4/6

- 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 - 5/6

- **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 - 6/6

- 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

Security Testing: Do the Math

- Recall that $MTBF = t/K$
- Suppose 10^6 security flaws in some SW
  - Say, Windows XP
- Suppose each bug has MTBF of 10^9 hours
- Expect to find 1 bug for every 10^3 hours testing
  - $10^3 = MTBF(10^9) / Flaws(10^6)$

Security Testing: Do the Math

- Good guys spend 10^7 hours testing:
  - **find 10^4 bugs** ($10^4 = 10^7 / 10^3$)
  - Good guys have found 1% of all the bugs
    - Found bugs (10^4) / All bugs (10^6)
- Bad guy spends 10^3 hours of testing:
  - **finds 1 bug ($1 = 10^3 / 10^3$)**
- 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
- Most of them are beyond of this course
- We will review only the issues related significant impact on security

Secure SW 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!

Design - 1/2

- Careful initial design is critical for security
- Try to avoid high-level errors
  - Such errors may be impossible to correct later
  - Certainly costly to correct these errors later
- For example
  - IPv4: no built-in security,
  - IPv6: IPSec mandatory,
  - but transition is slow -> Internet remains less secure…

Design - 2/2

- Verify assumptions, protocols, etc.
  - Usually informal approach is used
  - But sometimes, formal methods can be used
    - Possible to rigorously **prove** design is correct
    - In practice, only works in simple cases

Hazard Analysis - 1/2

- To build secure SW, the likely threats must known in advance
  - That is the field of Hazard Analysis
- In formal Hazard analysis (or threat modeling)
  - Develop hazard list (or List of what ifs) containing potential security problems
  - More systematic approach → Schneier's "attack tree" : possible attacks are organized into tree-like structure

Hazard Analysis - 2/2

- Many formal approaches: we will not discuss in this course
  - 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!

Testing - 1/2

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

Testing - 2/2

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”

Secu Testing: 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?

Secu Testing: Bottom Line

- How much testing is enough?
- Recall $\mathrm{MTBF} = t/K$
- 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 (i.e. small test)
  - For example, if we have a single test (or a few tests) to eliminate all buffer overflows, then we can eliminate this entire class of flaws with small amount of work
- Unfortunately, it does not seem to achieve such result today

Configuration

- 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

Software and 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

- Absolute security is (almost) never possible
  - Even with best development practices, security flaws will still exist
  - 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!

Q & A

aiclasshongik@gmail.com