Reverse Engineering, DRM, & Operating Systems Security

Prof. Tom Austin

San José State University

Spring 2014

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…
  - Reverse engineer is an attacker
  - Attacker only has exe (no source code)
  - Not bytecode (i.e., no Java, .Net)
- Attacker might want to
  - Understand the software
  - Modify (“patch”) the software
- SRE usually focused on Windows
  - So we focus on Windows

SRE Tools

- Disassembler
  - Converts exe to assembly (as best it can)
  - Cannot always disassemble 100% correctly
  - In general, it is not possible to re-assemble disassembly into working exe
- Debugger
  - Must step thru code to completely understand it
  - Labor intensive — lack of useful tools
- Hex Editor
  - To **patch** (modify) exe file
- Process Monitor, VMware, etc.

SRE Tools

- **IDA Pro** — the top-rated disassembler
  - Cost is a few hundred dollars
  - Converts binary to assembly (as best it can)
- **OllyDbg** — 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
- Process Monitor — freeware

Why is Debugger Needed?

- Disassembler gives **static** results
  - Good overview of program logic
  - User must “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”
  - And code not always disassembled correctly
- Disassembler **and** debugger both required for any serious SRE task

SRE Necessary Skills

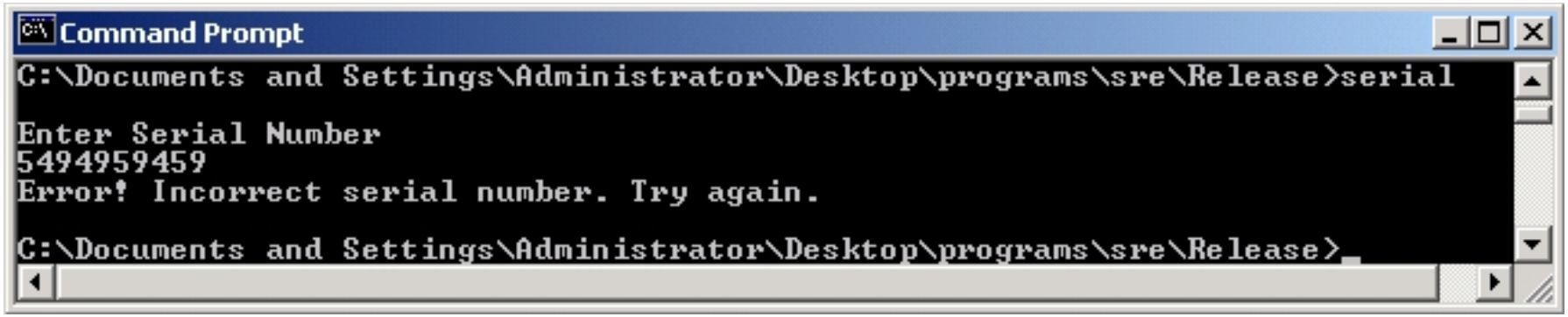
- Working knowledge of target assembly code
- Experience with the tools
  - IDA Pro — sophisticated and complex
  - **OllyDbg** — best choice for this class
- Knowledge of Windows **Portable Executable** (PE) file format
- Boundless patience and optimism
- SRE is a tedious, labor-intensive process!

SRE Example

- We consider a 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, would also need a debugger (OllyDbg)

SRE Example

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



- Can Trudy get serial number from exe?

SRE Example

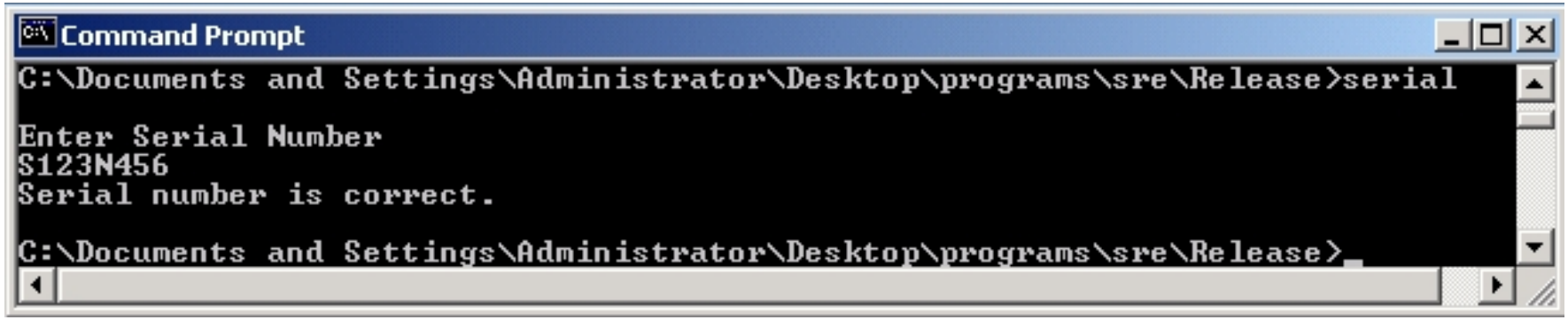
- IDA Pro disassembly

```
.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



- 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" is AND of eax with itself
  - Flag bit set to 0 only if eax is 0
  - If test yields 0, then jz is true
- Trudy wants jz to always be true
- Can Trudy patch exe so jz always holds?

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

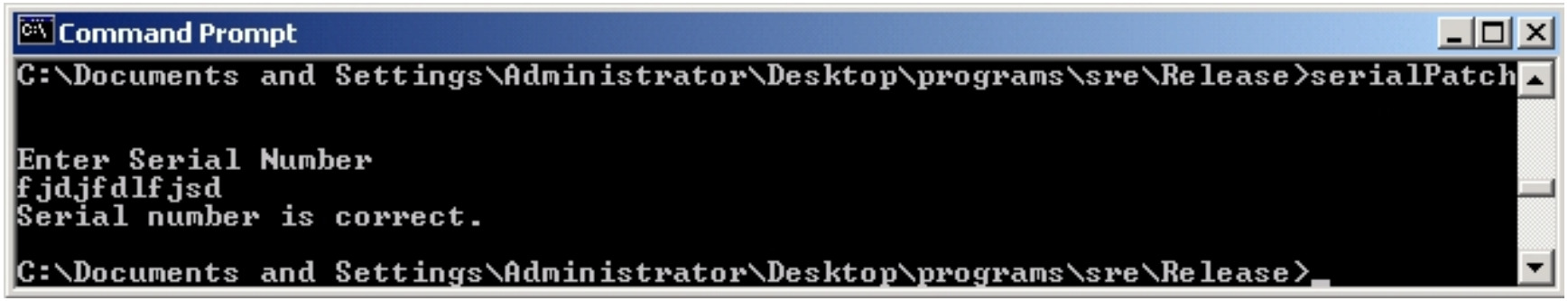
```
00001010h: 04 50 68 84 80 40 00 E8 7C 00 00 00 6A 08 8D 4C
00001020h: 24 10 68 78 80 40 00 51 E8 33 00 00 00 83 C4 18
00001030h: 85 C0 74 11 68 4C 80 40 00 E8 71 00 00 00 83 C4
00001040h: 04 83 C4 14 C3 68 30 80 40 00 E8 60 00 00 00 83
00001050h: C4 04 83 C4 14 C3 90 90 90 90 90 90 90 90 90 90
```

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



- **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

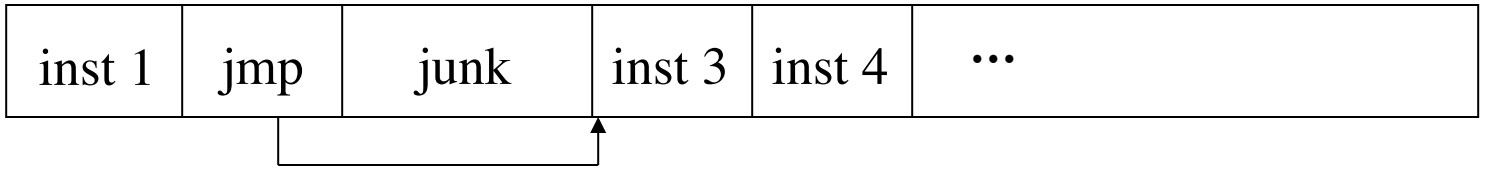
- **Impossible** to prevent SRE on open system
- But can make such attacks more difficult
- Anti-disassembly techniques
  - To confuse static view of code
- Anti-debugging techniques
  - To confuse dynamic view of code
- Tamper-resistance
  - Code checks itself to detect tampering
- Code obfuscation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

Anti-disassembly

- Anti-disassembly methods include
  - Encrypted or “packed” object code
  - False disassembly
  - Self-modifying code
  - Many other techniques
- Encryption **prevents** disassembly
  - But still need plaintext code to decrypt code!
  - Same problem as with polymorphic viruses

Anti-disassembly Example

- Suppose actual code instructions are



- What a "dumb" disassembler sees

- This is example of "false disassembly"
- But, clever attacker will figure it out

Anti-debugging

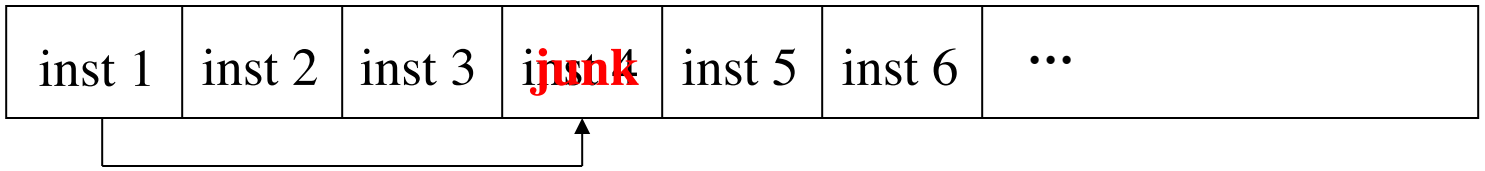
- IsDebuggerPresent()
- Can also monitor for
  - Use of debug registers
  - Inserted breakpoints
- Debuggers don't handle ***threads*** well
  - Interacting threads may confuse debugger
  - And therefore, confuse attacker
- Many other debugger-unfriendly tricks
  - See next slide for one example

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 if this segment of code executed more than once!
  - Also, code is very platform-dependent
- Again, clever attacker can 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 wastes 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 obfuscation
  - But it didn't work
- 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

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 Trudy can find this bit, she 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 can ultimately win

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.

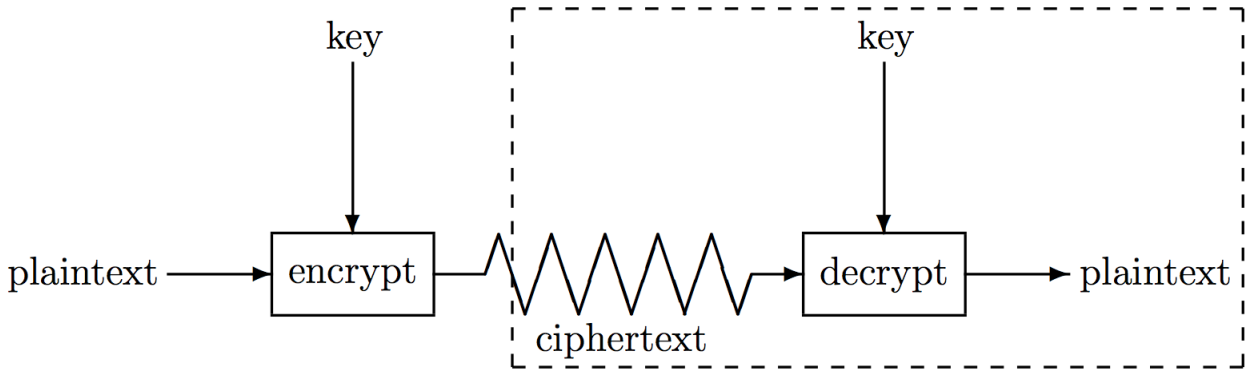
Persistent Protection

- "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

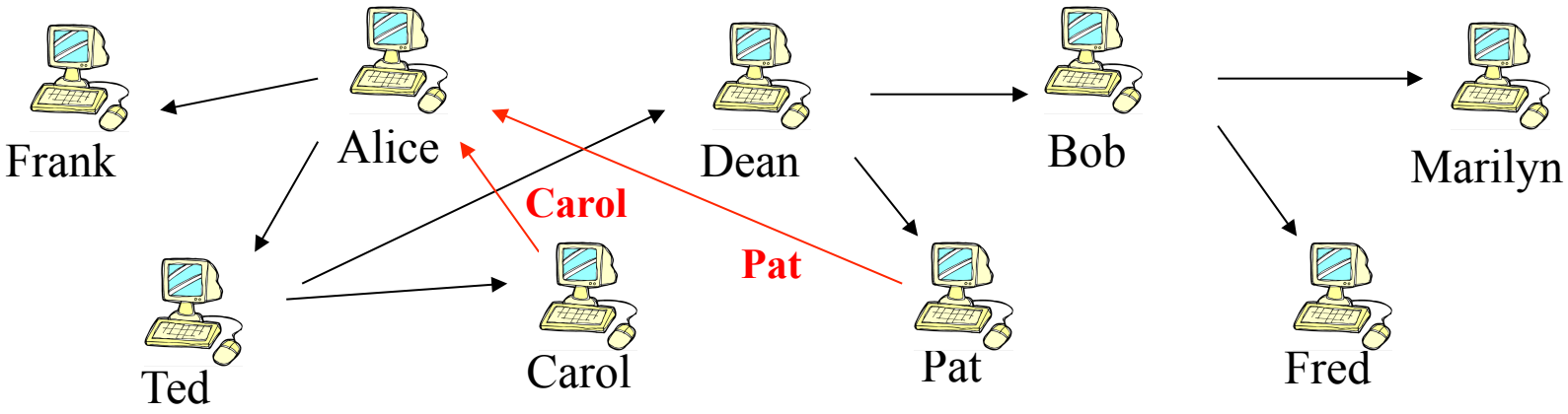
- Strong software-based DRM is impossible
- Why?
  - We can't really hide a secret in software
  - We cannot prevent SRE
  - User with full admin privilege can eventually break any anti-SRE protection
- Bottom line: **The** killer attack on software-based DRM is SRE

DRM for 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

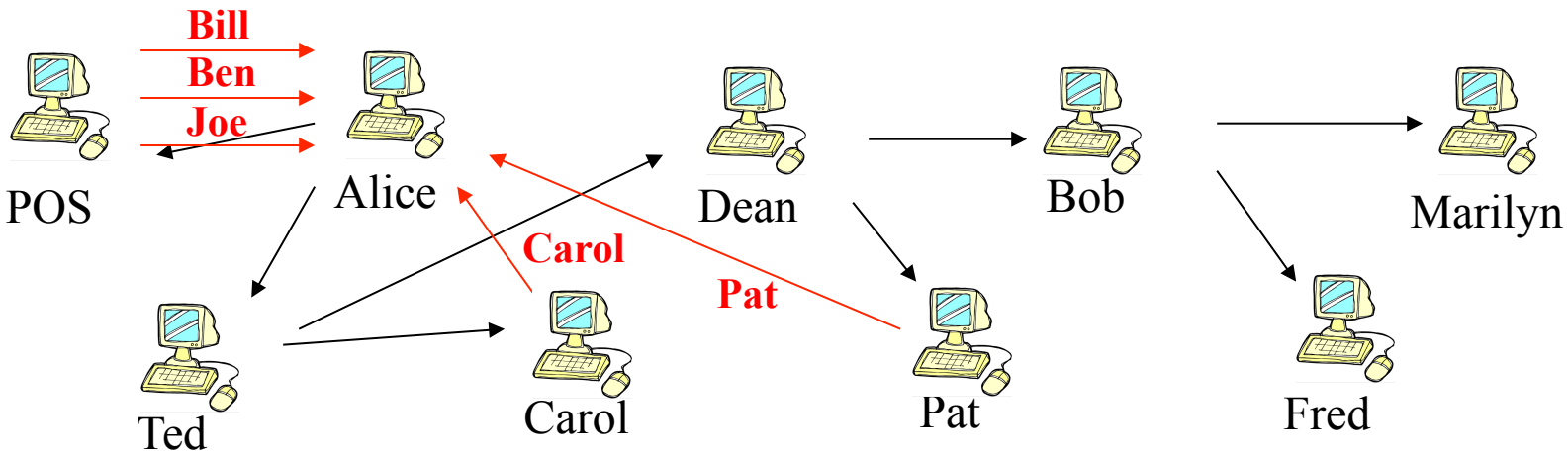
- Suppose Alice requests “Hey Jude”
- **Black** arrows: query flooding
- **Red** arrows: positive responses



- 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**, & **Joe** have DRM protected content

POS

- Bill, Ben and Joe must appear normal to Alice
- If “victim” (Alice) clicks POS response
  - DRM protected 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 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!

PyMusique

- iTunes was not available on Linux.
- DRM was applied on the client.
- PyMusique (later SharpMusique) purchased and downloaded songs, but did not apply the DRM.
- Apple very quickly released a new version & forced its users to upgrade.

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