Reverse Engineering

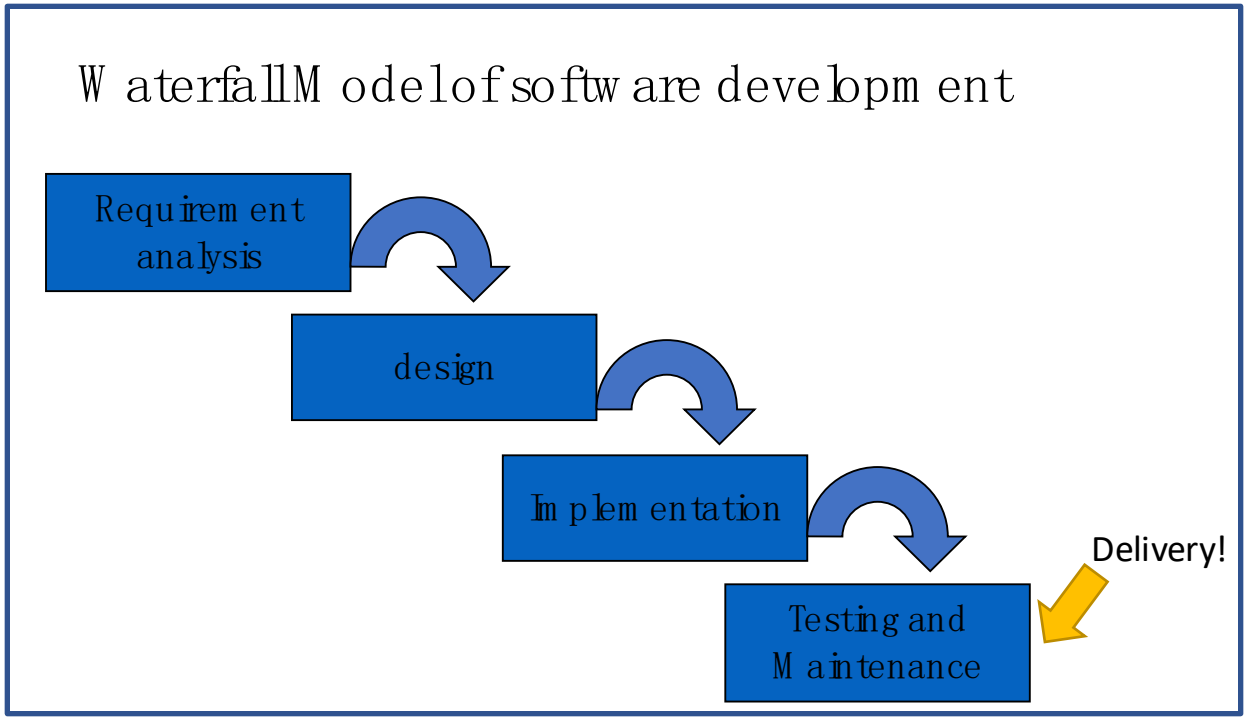
Shuai Wang

香 港 科 技 大 學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Forward Engineering

Forward Engineering:

- process of moving from high-level abstractions and logical designs to the physical implementation of a system.



Reverse Engineering vs. Forward Engineering

e.g., the deliverable of software development

Reverse Engineering:

- The process of taking something apart and analyzing its workings in detail, gradually recover its more "abstract" and high-level representation and intension.

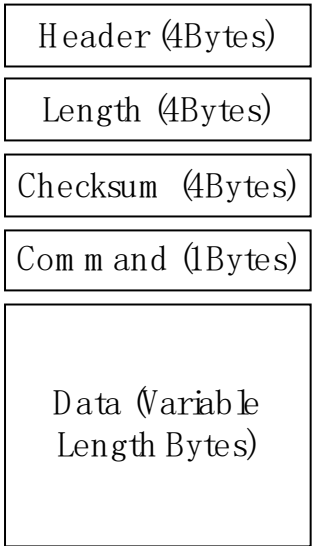
"*I have a hamburger patty in the fridge and I want to reverse engineer it to the cow it came from.*"
-- Usenet decompilation thread

Reverse Engineering in Cybersecurity

- Software reverse engineering
  - The basis of almost ALL software security missions
    - In security analysis scenarios, attacker/defender don't have source code!!
  - Also support various software re-engineering missions
- Protocol reverse engineering
  - Mostly well-established in network security.
- Model/algorithm reverse engineering
  - Emerging field; your reading materials today.
- Hardware reverse engineering
  - Your reading materials today.

Protocol Reverse Engineering

- Extracting the application/network level protocol.
  - Typically in a client-server based model
- Intercept the traffic → Reverse the protocol
  - Learn how it's structured



Package structure

```
00000000  42 49 4e 58                                       BINX
00000004  00 00 00 12                                       ....
00000008  00 00 05 d6                                       ....
0000000C  00                                                .
0000000D  04 74 65 73 74 0a 6b 6b  62 2d 75 62 75 6e 74 75  .test.kk b-ubuntu
0000001D  00                                                .
0000001E  00 00 00 1e                                       ....
00000022  00 00 09 f9                                       ....
00000026  03                                                .
00000027  04 74 65 73 74 17 54 68  69 73 20 69 73 20 61 20  .test.Th is is a
00000037  74 65 73 74 20 6d 65 73  73 61 67 65 21           test mes sage!
00000044  00 00 00 17                                       ....
00000048  00 00 07 57                                       ...W
0000004C  03                                                .
0000004D  04 74 65 73 74 10 48 65  6c 6c 6f 20 2d 20 74 65  .test.He llo - te
0000005D  73 74 69 6e 67 21                                 sting!
00000063  00 00 00 15                                       ....
00000067  00 00 06 8d                                       ....
0000006B  02                                                .
0000006C  13 49 27 6d 20 67 6f 69  6e 67 20 61 77 61 79 20  .I'm goi ng away
0000007C  6e 6f 77 21                                       now!
```

Several network packages

Algorithm/Model Reverse Engineering

- Ways of reverse engineering algorithms/models
  - Reverse engineering its software/get the source code
  - Magic number
  - Statistically analyze a large number of pairs of queries
    - Reverse engineering deep neural network (DNN)…
    - Your reading materials today

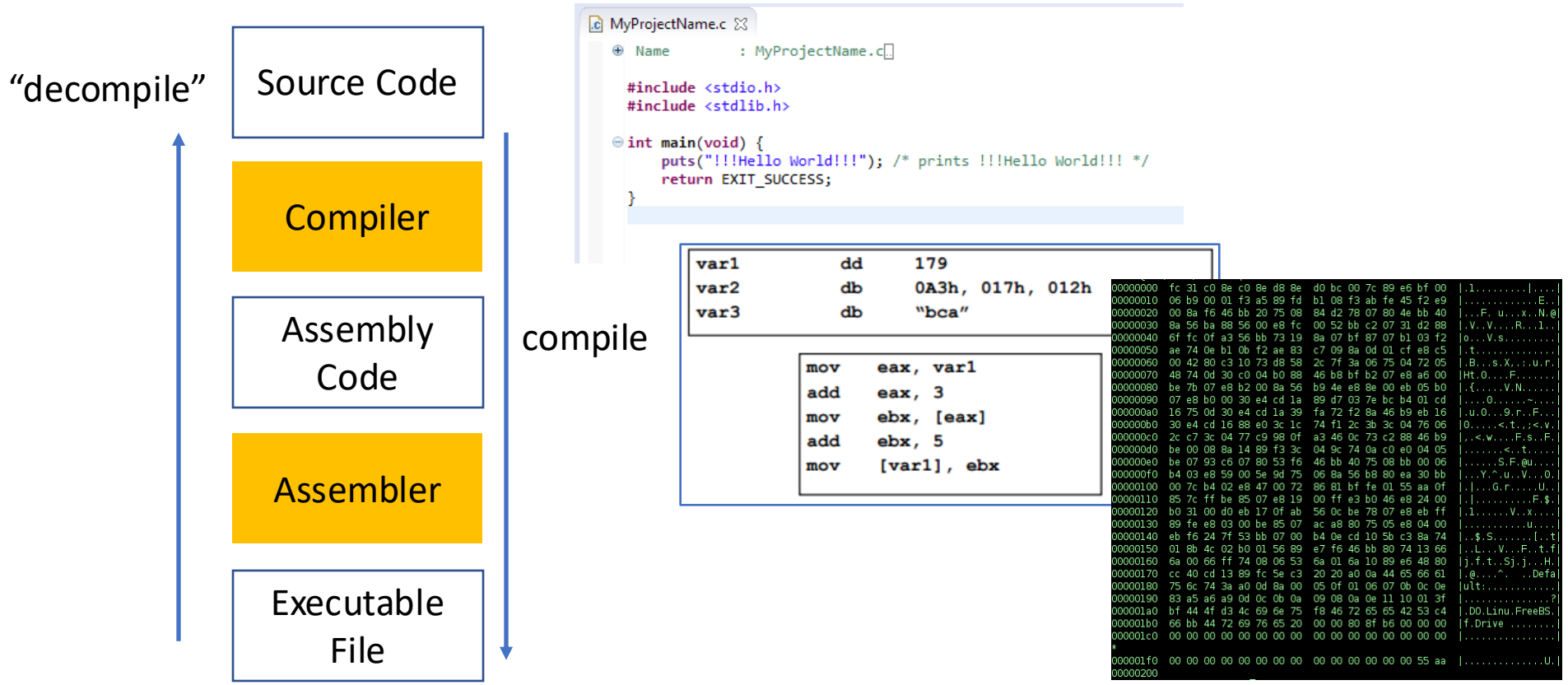
TEA Encryption

Assuming 32 rounds:

(K[0], K[1], K[2], K[3]) = 128 bit key
(L,R) = plaintext (64-bit block)
delta = 0x9e3779b9 ← Magic number!
sum = 0
for i = 1 to 32
    sum += delta
    L += ((R<<4)+K[0]) ⊕ (R+sum) ⊕ ((R>>5)+K[1])
    R += ((L<<4)+K[2]) ⊕ (L+sum) ⊕ ((L>>5)+K[3])
ciphertext = (L,R)

Software Reverse Engineering: The Big Picture

- Software reverse engineering converts executable files into source code.



Software Reverse Engineering Terminology

- Source code
  - Your program..
- Assembly code (including assembly instructions + data)
  - Compiler translates source code into assembly code
- Executable file
  - Translate assembly code into a “compact” and “executable” format. (.exe)
- Disassembler
  - Coverts executable files into CPU assembly instructions.
- Decompiler
  - **Convention**: converts executable files to source code
  - A “decompiler” would therefore contain a disassembler

Software Compilation: conceptually “one way function”

source code 1:
```
if (x != 0) x = 0;
```

source code 2:
```
x = 0;
```

source code 3:
```
if (x == 1) x = 0;
else if (x == 2) x = 0;
else if (x == 3) x = 0;
else if (x == 4) x = 0;
else if (x == 5) x = 0;
else if (x == 6) x = 0;
else x = 0;
```

assembly code:
```
movq    $0, -8(%rbp)
```

(part of the) machine code:
```
706:   48 c7 45 f8 00 00 00
```

Very difficulty to map back to the exactly piece of code.

- But for comprehension or reuse, usually either one of these three source code is OK.

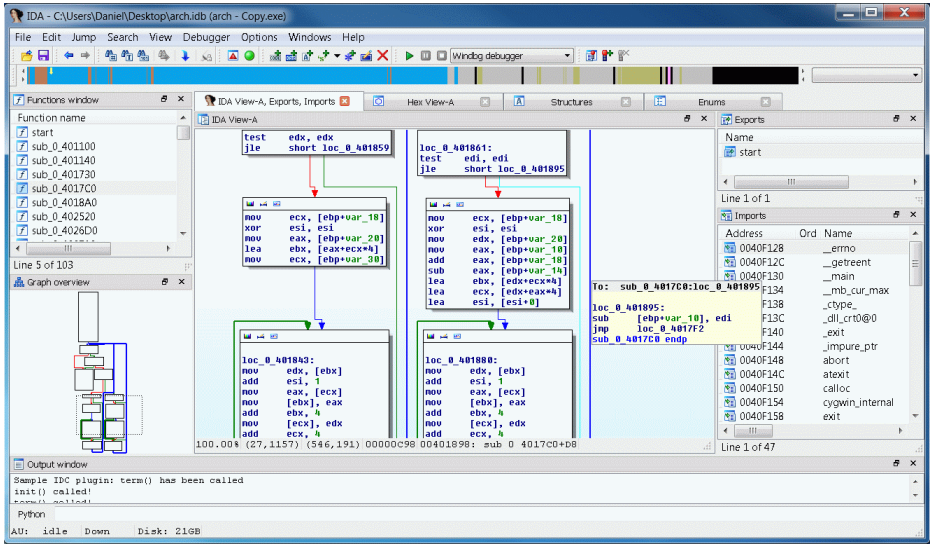
Software Reverse Engineering

- The most fundamental component of software security and re-engineering missions
  - Security: how to launch an attack? “vulnerability” analysis
  - Security: I want to know whether my third-party library is malicious or not, or, I just want to steal other people’s code!
  - Re-engineering: code reuse; code migration; add new features to executable files.
- What is the threat model here?
  - Attackers have exe but no source code.

Reverse engineering a Windows game (starcraft) and migrate to ARM.

Software reverse engineering is a real business

- Commercial reverse engineering tools are worth thousands of US dollars.
  - IDA-Pro; JEB3; Hopper;
- Many open source implementations and support.
  - NSA released Ghidra, an open source decompiler with the aim of "*training next generation of cyber defender*."



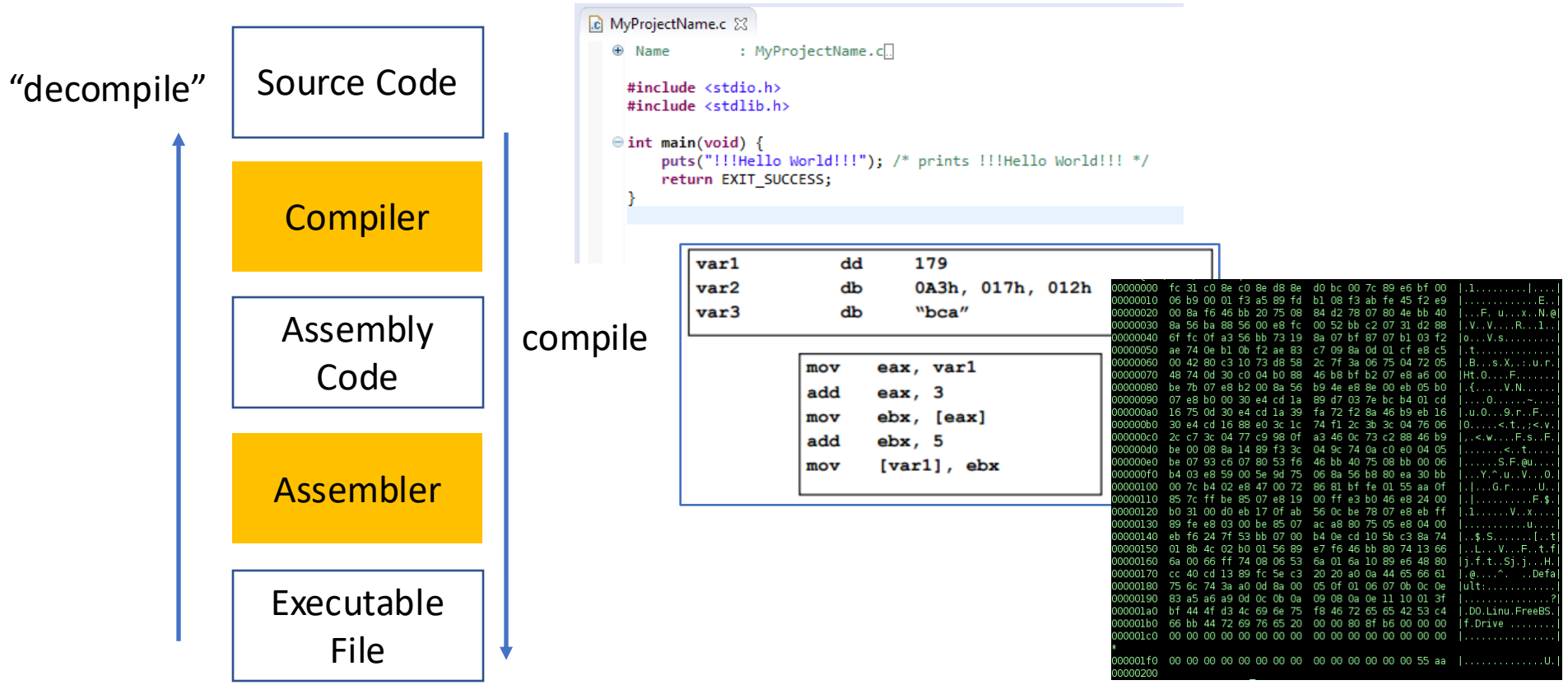
Then, to what extend we can "re-use" the recovered source code?

- The traditional attitude are indeed quite pessimistic
  - *"The recovered software are mostly used for manual analysis, and cannot be smoothly reused/recompiled as a standard piece of C/C++ code."*
- But recent several years we have made fundamental improvement to automatically re-compile and re-use the recovered source code.
  - To date, re-compile/re-use recovered "well-formatted" source code is mostly an engineering effort (no research challenge).
    - Java bytecode executable, very easy
    - C/C++ binary executable, still subtle issues

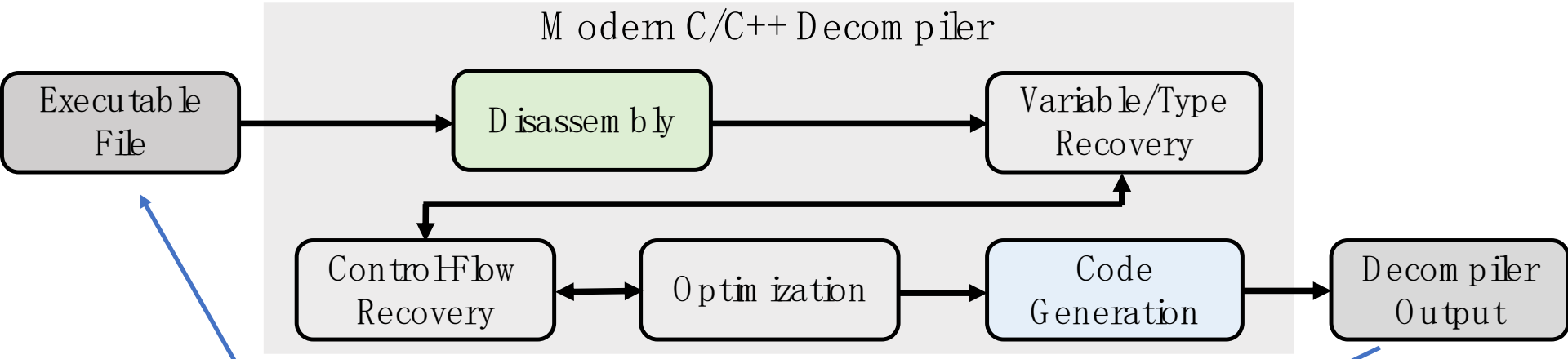
My Ph.D. thesis; first work in this line of research!!!

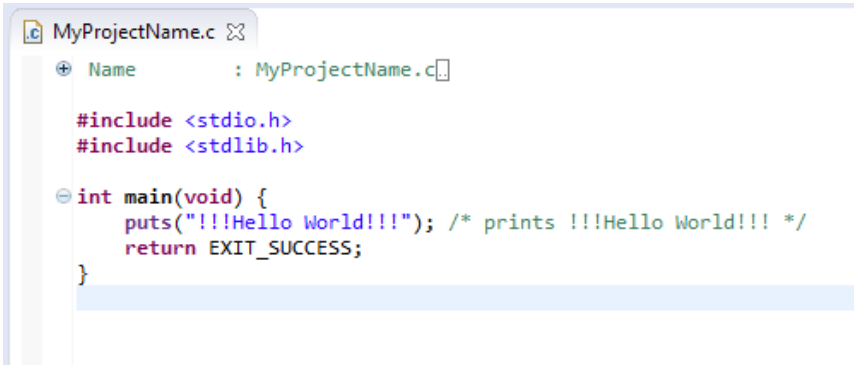
Case Study: Reverse Engineering C/C++ Executable Files

- For software reverse engineering, convert executable files compiled from C/C++ source code is the most challenging task.
  - Reverse engineering Java Bytecode is much easier…



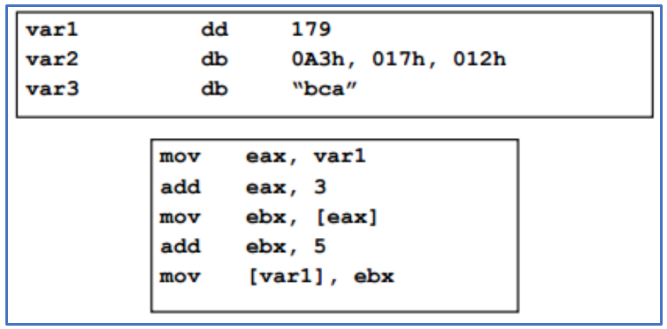
A Simplified Workflow of Modern C/C++ Decompilers (start from green boxes)





Disassembling

Executable file



Assembly instructions

Linear disassembling

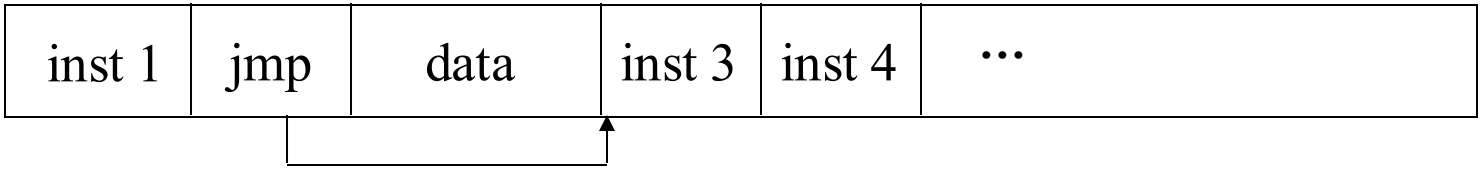
- Starts from the first byte in the code section of the executable file to decode each byte (map one or a sequence of bytes to its corresponding assembly instruction), until the end.

Recursive disassembling

- Explain later

Linear disassembling can be trapped

- Suppose an “embedding” case, where the actual code instructions could be



This is legit! Compiler can embed data bytes into code section!

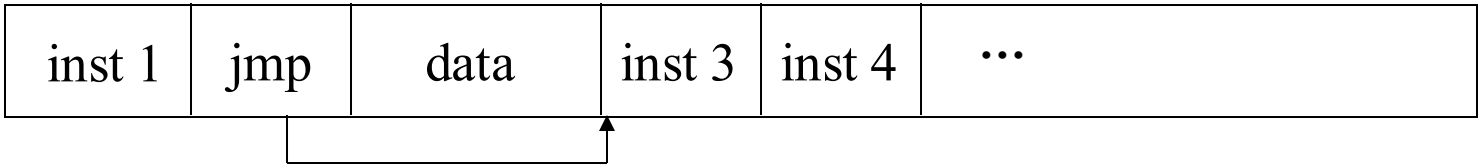
- What a “linear” disassembler sees

.

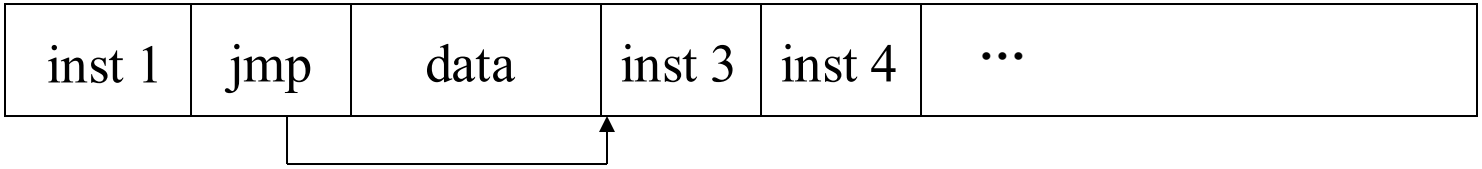
- So how do we disassemble this?
- Well, to take one step back, how can CPU correctly figure this out?

Recursive disassembling

- For the “embedding” case, actual code instructions should be

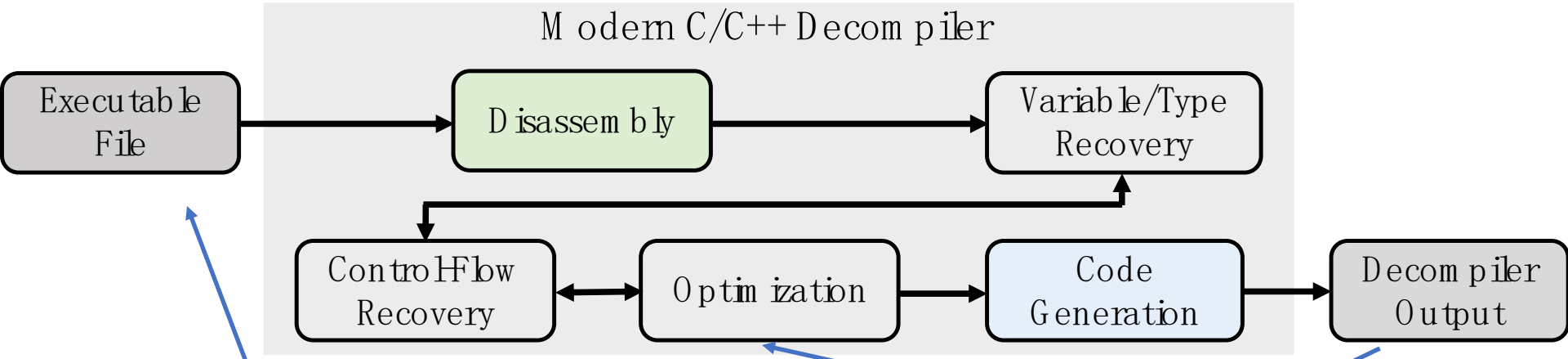


- What a “recursive” disassembler sees:



Recursive disassembling follow the program control transfers to disassemble

The middle stage (orange boxes)



“Beautify” the code; omitted

Variable/Type Recovery

```
var1        dd      179
var2        db      0A3h, 017h, 012h
var3        db      "bca"

        mov     eax, var1
        add     eax, 3
        mov     ebx, [eax]
        add     ebx, 5
        mov     [var1], ebx
```

Assembly code

```c
int main ()
{
  int diff, size = 8;
  char *buf1, *buf2;

  printf ("hello world!\n");

  return 0;
}
```

C code

Lack of: int num, i;

type variable

Variable: references of each program data.

Type: programs deal with data in a wide variety of forms such as numbers, strings (text), images, or lists of things. *Type is the annotation of data.*

Some general info; detailed procedure should be taught in typical PG courses like *program analysis* or *advanced compiler*.

A Holistic View of Variable Recovery

Program memory space

$byte_1$	$byte_2$	$byte_3$	$byte_4$	$byte_5$	$byte_6$	$byte_7$	$byte_8$	$byte_9$	…

the first two bytes are always used together, then potentially indicate a variable of two bytes.

Program memory space

$byte_1$	$byte_2$	$byte_3$	$byte_4$	$byte_5$	$byte_6$	$byte_7$	$byte_8$	$byte_9$	…
v1		**v2**				**v3**			

*From a **very holistic view**, variable recovery tries to divide the whole memory space into small regions, each small region will be deemed as one "variable".*

Type Recovery via Type Inference

Unfortunately variables of different types can have same length in memory

```
? a = 0x8040200; // I don't know the type of a
b = *a;          // pointer dereference
```

All right, now I know a must be a pointer…

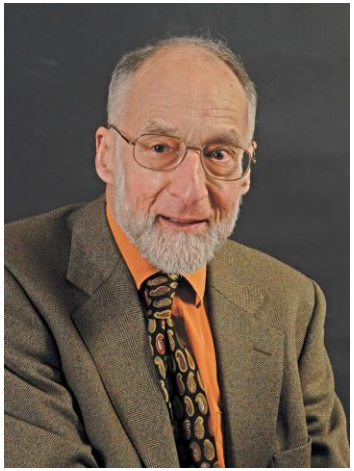
```
a = 0x8040200;   // I don't know the type of a
printf("%d\n", a); // print out an integer
```

All right, now I know a must be an integer…

```
fopen(a, "r"); // a is used as the input of fopen
```

All right, now I know a must be a string (file name)…

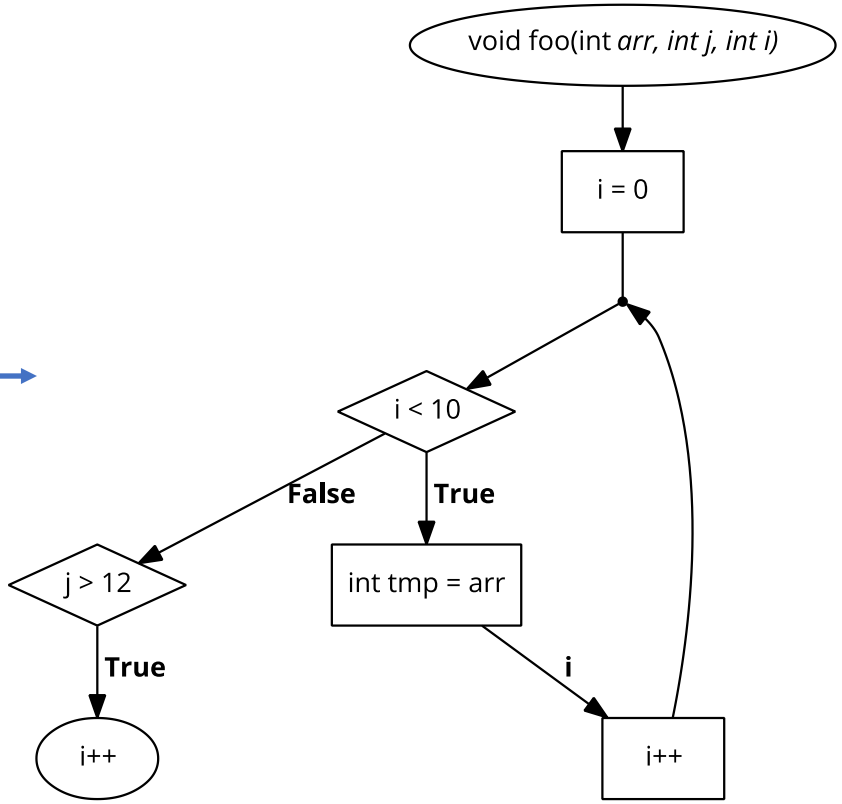
Type inference formulates a deductive proof procedure to gradually recover types of variables.



Control Flow Graph Recovery

Assembly programs

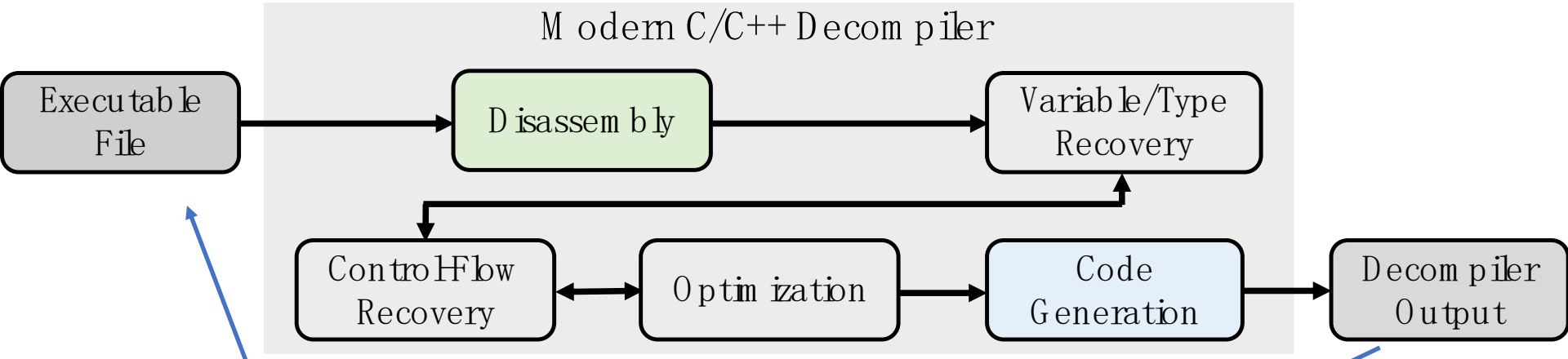
extract

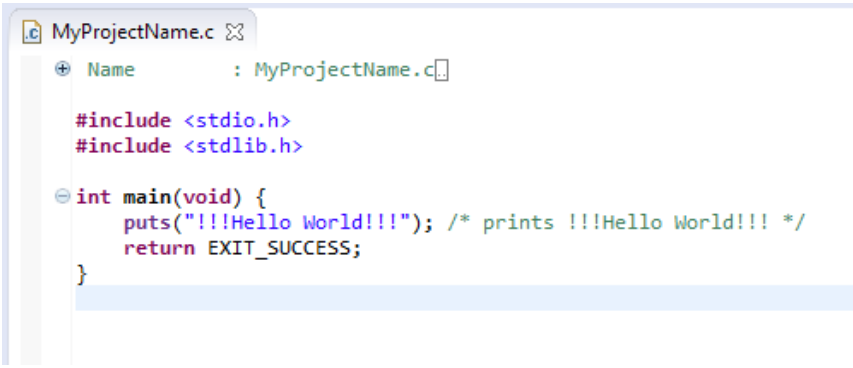


Control-flow graph

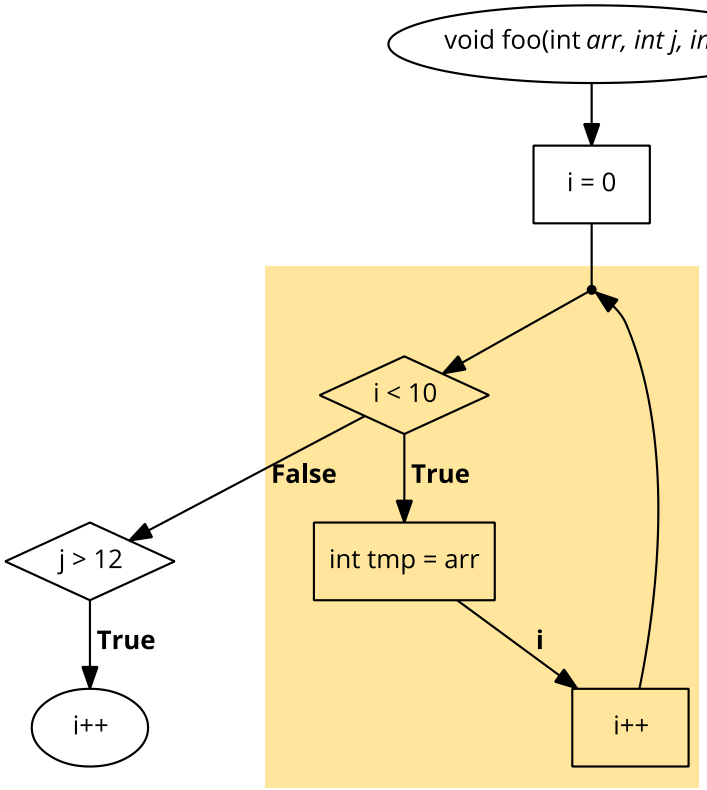
Pre-requisite to recover C/C++ control structures (while;for;if)

The backend stage (blue boxes)





Code Generation with Pattern Matching and Concretize Code Templates



should be a ”for” loop

```
void foo(int *arr, int j, int i)
{
    for(i = 0; i < 10; i++) {
      int tmp = arr[i];
    }

    if (j > 12) {
      i++;
    }
}
```

Practical challenges (bug) during C/C++ decompilation

```
1. int i = 0;
2. for (i = 6; i < -12; i -= 6) {
3.   ... // not reachable
4. }
```

C code generated by CSmith

```
1. unsigned int i = 0;
2. for (i = 6; i < -12; i -= 6) {
3.   ... // reachable
4. }
```

C code decompiled by A CommercialTool

(a)

```
1. uint32_t a = 0xffff0001;
2. a = (uint8_t)a >> (uint8_t)6;
```

C code generated by CSmith

```
1. uint32_t a = 0xffff0001;
2. a = ((unsigned char)(a >> 6));
```

C code decompiled by A CommercialTool

(b)

```
1. int32_t a = 0xaaaaffff;
2. a = a + 4;
```

C code generated by CSmith

```
1. int16_t a = 0xaaaaffff;
2. a = a + 4;
```

C code decompiled by A CommercialTool

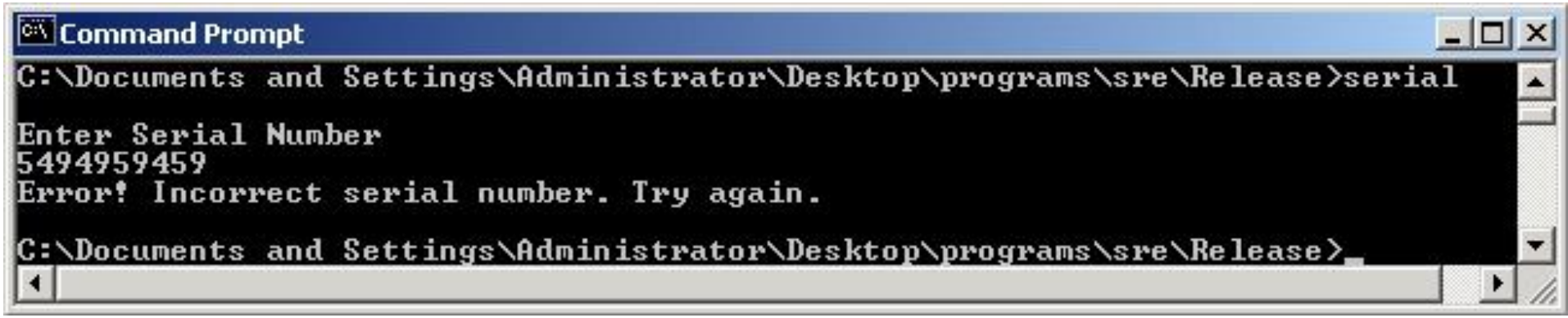
(c)

Note: all the decompiled code has been simplified for readability.
For case (a), the `for` loop in the decompiled output will be iterated for many times.
For case (b) and (c), variable a would have different values in the decompiled output.

Variable/type recovery is still difficult for de facto commercial decompiler…

Application: Software Crack

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



- Can the attacker get serial number from exe?

This software Crack example is from Mark Stamp.

Serial number checking

Simple if condition by taking the user input and a hardcoded serial number for comparison

```
printf("Enter Series Number");
scanf(x);
If(is_equal(x, "????") = [redacted]) {
   printf("Error! Incorrect series number. Try again.");
}
```

code is simplified.

Reverse Engineering the (Victim) Software

- 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
```

print

read_input

cmp

print

- Looks like serial number is S123N456

Serial number checking

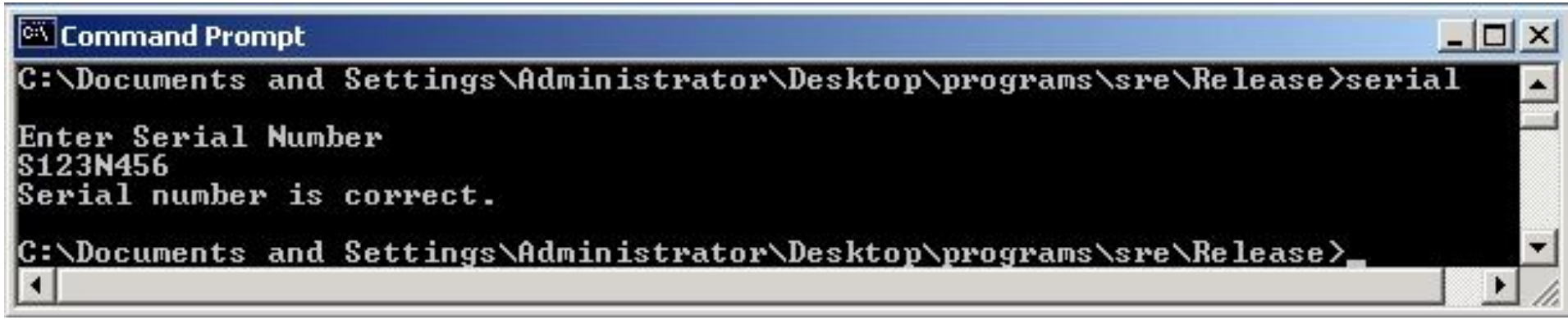
Simple if condition by taking the user input and a hardcoded serial number for comparison

```
printf("Enter Series Number");
scanf(x);
If(is_equal(x, S123N456) == False) {
   printf("Error! Incorrect series number. Try again.");
}
```

code is simplified.

SRE Example

- Try the serial number S123N456



- It works!
- Can we do better?

Serial number checking

Simple if condition by taking the user input and a hardcoded serial number for comparison

```
printf(“Enter Series Number”);
scanf(x);
If(is_equal(x, “????”) = [redacted]) {
   printf(“Error! Incorrect series number. Try again.”);
}
```

code is simplified.

SRE Example

- Again, 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

- Can edit serial.exe with hex editor

serial.exe

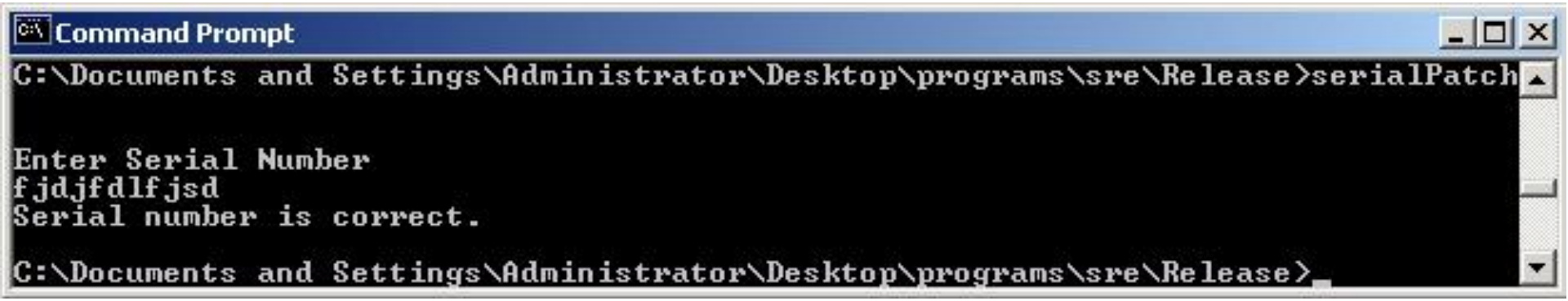
```
00001010h: 04 50 68 84 80 40 00 E8 7C 00 00 00 6A 08 8D 4C
00001020h: 24 10 68 78 80 40 00 51 E8 33 00 00 00 83 C4 18
00001030h: 85 C0 74 11 68 4C 80 40 00 E8 71 00 00 00 83 C4
00001040h: 04 83 C4 14 C3 68 30 80 40 00 E8 60 00 00 00 83
00001050h: C4 04 83 C4 14 C3 90 90 90 90 90 90 90 90 90 90
```

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 from now on…

SRE Example

- Back to 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
```

Software Reverse Engineering

- **Impossible** to prevent SRE on open system
- Can we make such attacks more difficult?
- Anti-disassembly techniques
  - Embed data with code.
- Code obfuscation
  - Make code more difficult to understand
  - Encryption (packer and de-packer)
  - We will talk about that…