

Reverse Engineering

Shuai Wang



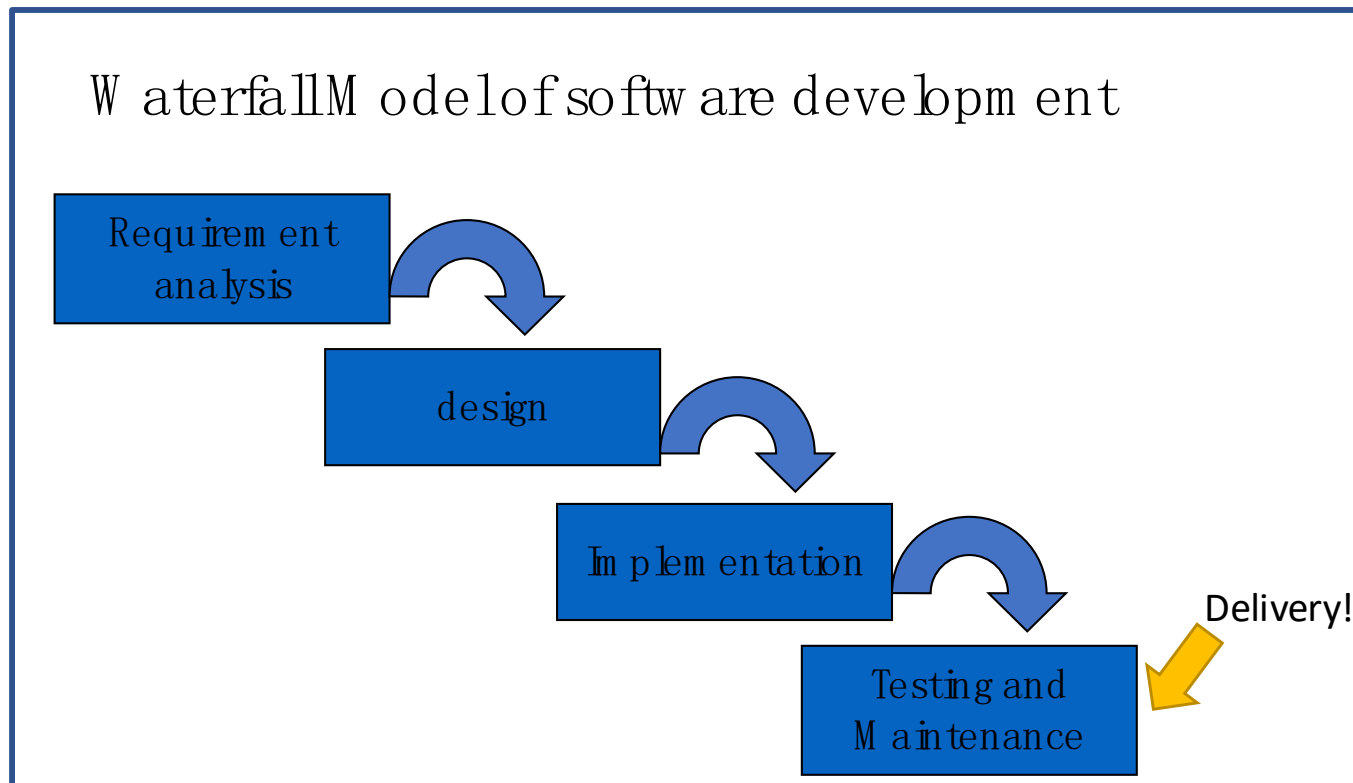
香港科技大學

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.

Malware analysis \Rightarrow understanding & examination of information necessary to respond to network instruction



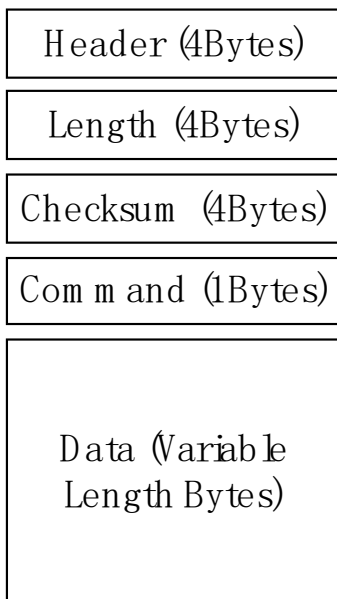
“I have a hamburger patty in the fridge and I want to reverse engineer it to the cow it came from.”
-- Usenet decompilation thread

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.This 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:

$(K[0], K[1], K[2], K[3]) = 128 \text{ bit key}$

$(L, R) = \text{plaintext (64-bit block)}$

$\text{delta} = 0x9e3779b9$



Magic number!

$\text{sum} = 0$

for $i = 1$ to 32

$\text{sum} += \text{delta}$

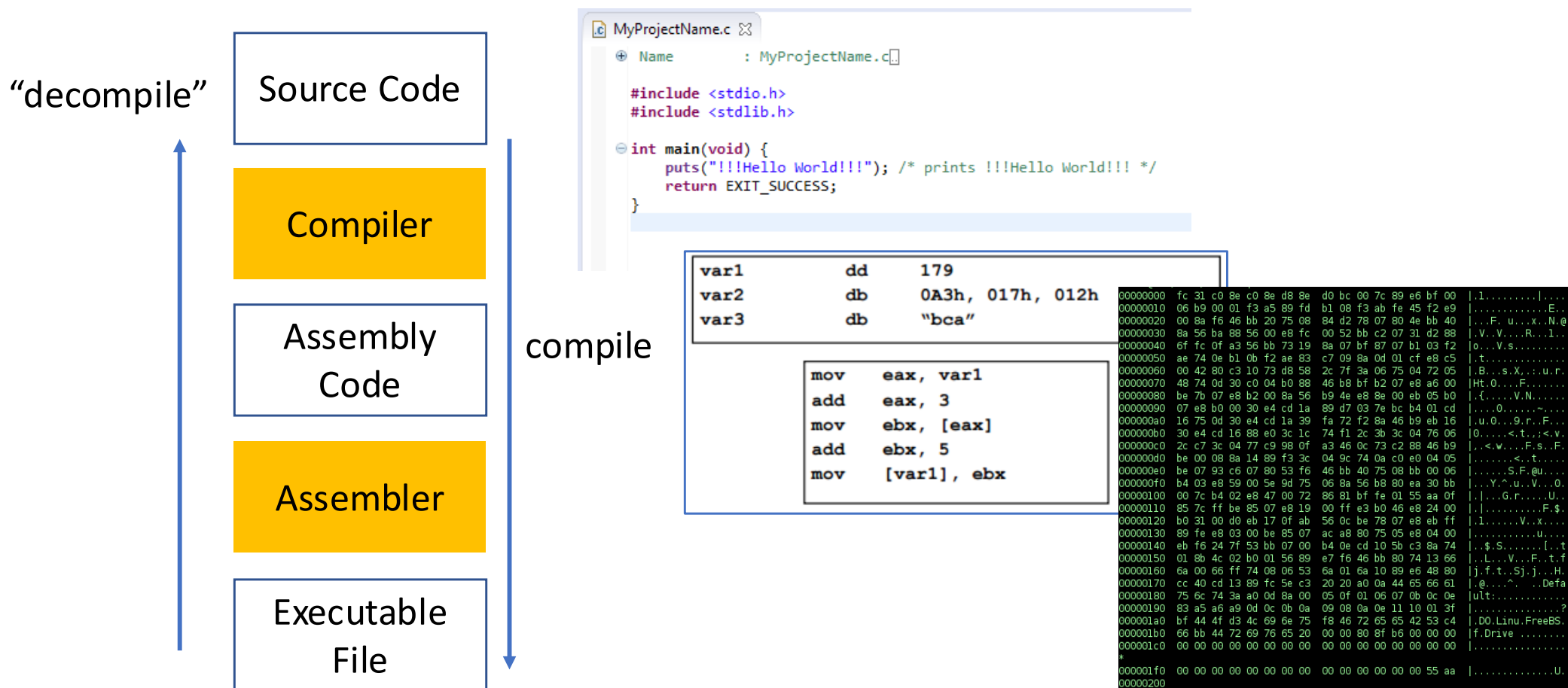
$L += ((R \ll 4) + K[0]) \oplus (R + \text{sum}) \oplus ((R \gg 5) + K[1])$

$R += ((L \ll 4) + K[2]) \oplus (L + \text{sum}) \oplus ((L \gg 5) + K[3])$

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 difficult 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

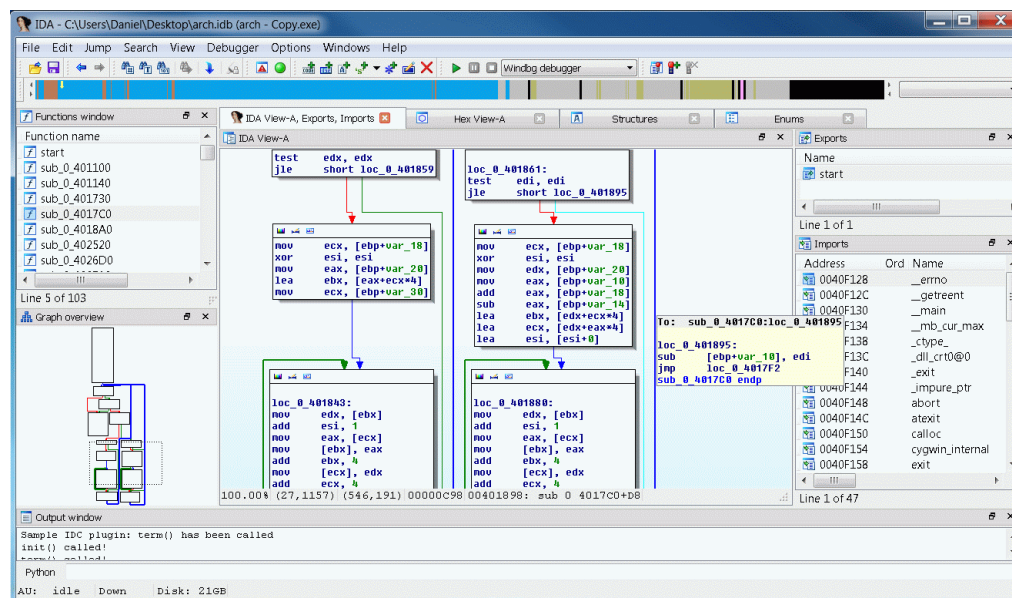
- The most fundamental component of software security and re-engineering missions
 - **Security**: how to launch an attack? “vulnerability” analysis
 - **Security**: I want to know whether my third-party library is malicious or not, or, I just want to steal other people’s code!
 - **Re-engineering**: 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.”*

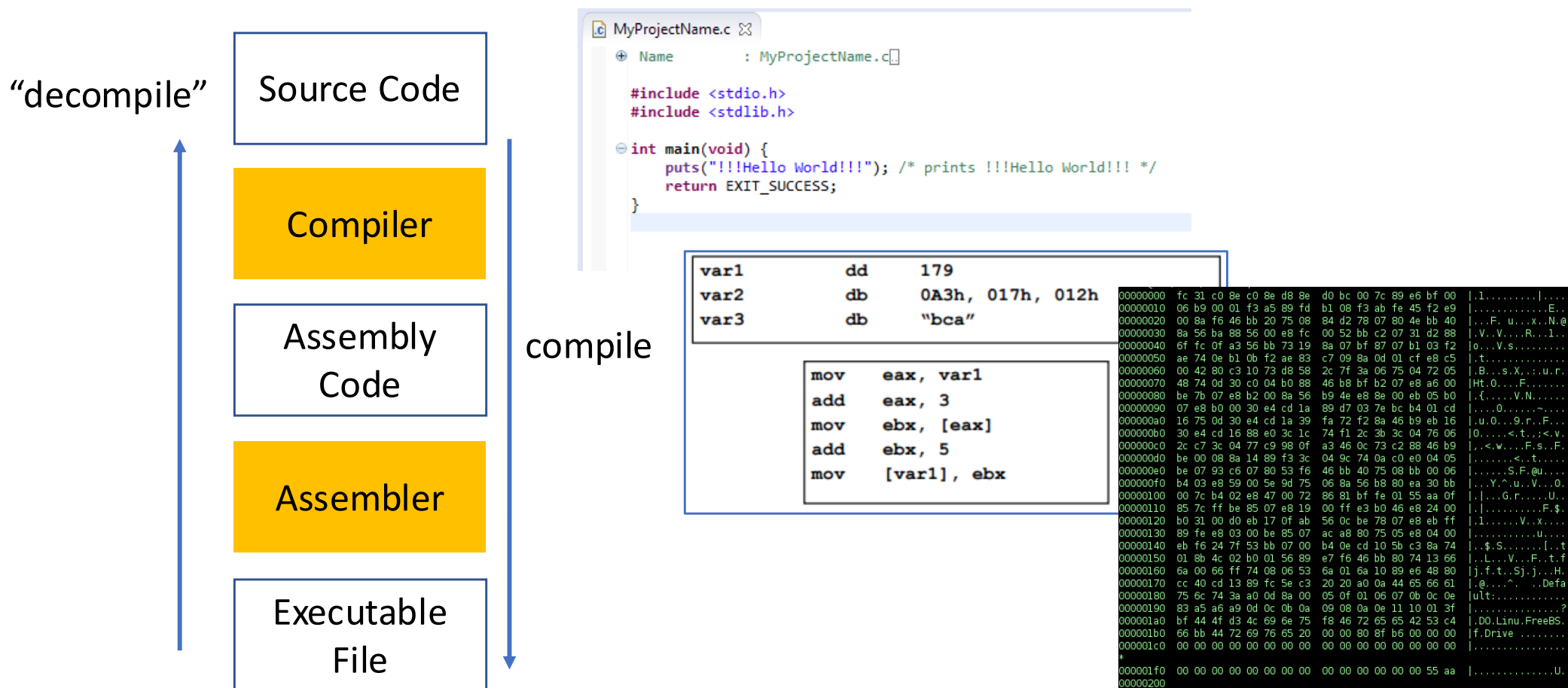


Then, to what extent we can “re-use” the recovered source code?

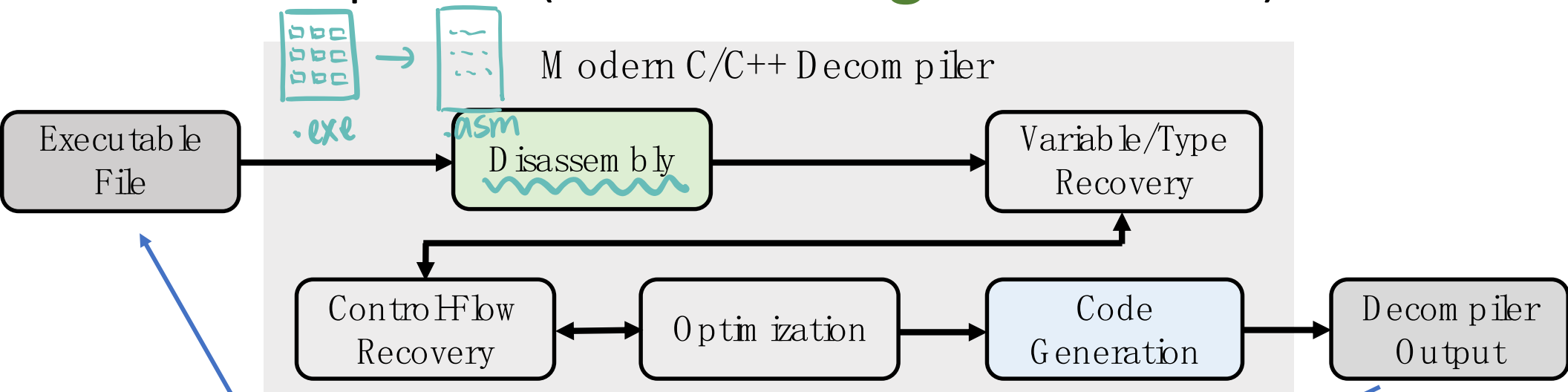
- The traditional attitude are indeed **quite pessimistic**
 - *“The recovered software are mostly used for **manual analysis**, and cannot be smoothly reused/recompiled as **a standard piece of C/C++ code**. ”*
 - But recent several years we have made fundamental improvement to **automatically** re-compile 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**)



```
00000000 fc 31 c0 8e c0 8e d8 8e d0 bc 00 7c 89 e6 bf 00 .l.....|
00000010 06 b9 00 01 f3 a5 89 fd b1 08 f3 ab fe 45 f2 e9 .....E.
00000020 00 8a f6 46 bb 20 75 08 84 d2 78 07 80 4e bb 40 .....F. u...x..N.@
00000030 8a 56 ba 88 56 00 e8 fc 00 52 bb c2 07 31 d2 88 .....V. V...R...l.
00000040 6f fc 0f a3 56 bb 73 19 8a 07 bf 87 07 b1 03 f2 .....o...V.s.....
00000050 ae 74 0e b1 0b f2 ae 83 c7 09 8a 0d 01 cf e8 c5 .....t.....
00000060 00 42 80 c3 10 73 d8 58 2c 7f 3a 06 75 04 72 05 .....B...s.X...u.r.
00000070 48 74 0d 30 c0 04 b0 88 46 b8 bf b2 07 e8 a6 00 .....Ht.O...F.....
00000080 be 7b 07 e8 b2 09 8a 56 b9 4a e8 9e 00 eb 05 b0 .....f...V.N.....
00000090 07 e8 b0 00 30 e4 cd 1a 89 d7 03 7e bc b4 01 cd .....o...O...
000000a0 16 75 0d 30 e4 cd 1a 39 fa 72 f2 8a 46 b9 eb 16 .....u.O...9.r.F.....
000000b0 30 e4 cd 16 88 e0 3c 1c 74 f1 2c 3b 3c 04 76 06 .....<...<.t...<.v.
000000c0 2c c7 3c 04 77 c9 98 0f a3 46 0c 73 c2 88 46 b9 .....<.w...<.F.s..F.
000000d0 be 00 08 8a 14 89 f3 3c 04 9e 74 0a c0 a0 04 05 .....<.t.....
000000e0 be 07 93 c6 07 80 53 f6 46 bb 40 75 08 bb 00 06 .....S.F.@.....
000000f0 b4 03 e8 59 00 5e 9d 75 06 8a 56 b8 80 ea 30 bb .....Y...u...V...O.
00000100 00 7c b4 02 e8 47 00 72 86 81 bf fe 01 55 aa 0f .....G.R...U.....
00000110 85 7c ff be 85 07 e8 19 00 ff e3 b0 46 e8 24 00 .....f...F.$.....
00000120 b0 31 00 d0 eb 17 0f ab 56 0c be 78 07 e8 eb ff .....l...V...x.....
00000130 89 fe e8 03 00 be 85 07 ac a8 80 75 05 e8 04 00 .....$.S.....[.t
00000140 eb f6 24 7f 53 bb 07 00 b4 0e cd 10 5b c3 8a 74 .....L...V...F...t.f
00000150 01 8b 4c 02 b0 01 56 89 e7 f6 46 bb 80 74 13 66 .....j.f.t...Sj...H.
00000160 6a 00 66 ff 74 08 06 53 6a 01 6a 10 89 e6 48 80 .....@...<...Defa
00000170 cc 40 cd 13 89 fc 5e c3 20 20 a0 0a 44 65 66 61 .....ult:...
00000180 75 6c 74 3a a0 0d 8a 00 05 0f 01 06 07 0b 0c 0e .....?
00000190 83 a5 a6 a9 0d 0c 0b 0a 09 08 0a 0e 11 10 01 3f .....D.O.Linu.FreeBS.
000001a0 bf 44 4f d3 4c 69 6e 75 f8 46 72 65 65 42 53 c4 .....f.Drive .....
000001b0 66 bb 44 72 69 76 65 20 00 00 80 8f b6 00 00 00 .....
000001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
000001f0 00 00 00 00 00 00 00 00 00 00 00 00 55 aa .....U.
00000200
```

```
MyProjectName.c
+ Name : MyProjectName.c

#include <stdio.h>
#include <stdlib.h>

- int main(void) {
    puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
    return EXIT_SUCCESS;
}
```

Disassembling

```
00000000 fc 31 c0 8e c0 8e d8 8e d0 bc 00 7c 89 e6 bf 00 [.l.....].  
00000010 06 b9 00 01 f3 a5 89 fd b1 08 f3 ab fe 45 f2 e9 [...F.u...x..N.e].  
00000020 00 8a f6 46 bb 20 75 08 84 d2 78 07 80 4e bb 40 [...V...R...l...].  
00000030 8a 56 ba 88 56 00 e8 fc 00 52 bb c2 07 31 d2 88 [...V.s.....].  
00000040 6f fc 0f a3 56 bb 73 19 8a 07 bf 87 07 b1 03 f2 [...t.....].  
00000050 ae 74 0e b1 0b f2 ae 83 c7 09 8a 0d 01 cf e8 c5 [...B...s.X...u.r].  
00000060 00 42 80 c3 10 73 d8 58 2c 7f 3a 06 75 04 72 05 [...Ht.O...F.....].  
00000070 48 74 0d 30 c0 04 b0 88 46 b8 bf b2 07 e8 a6 00 [...{.....V.N.....].  
00000080 be 7b 07 e8 b2 00 8a 56 b9 4e e8 8e 00 eb 05 b0 [...0.....].  
00000090 07 e8 b0 00 30 e4 cd 1a 89 d7 03 7e bc b4 01 cd [...u.O...9.m.F.....].  
000000a0 16 75 0d 30 e4 cd 1a 39 fa 72 f2 8a 46 b9 ab 16 [...0....<.t...<.v].  
000000b0 30 e4 cd 16 88 e0 3c 1c 74 f1 2c 3b 3c 04 76 06 [...<.W...F.s..F.....].  
000000c0 2c c7 3c 04 77 c9 98 0f a3 46 0c 73 c2 88 46 b9 [...<.....t.....].  
000000d0 be 00 08 8a 14 89 f3 3c 04 9c 74 0a c0 e0 04 05 [...S.F.@.....].  
000000e0 be 07 93 c6 07 80 53 f6 46 bb 40 75 08 bb 00 06 [...Y.^..u..V...U...].  
000000f0 b4 03 e8 59 00 5e 9d 75 06 8a 56 b8 80 ea 30 bb [...G.r.....F.$.....].  
00000100 00 7c b4 02 e8 47 00 72 86 81 bf fe 01 55 aa 0f [...l.....V.x.....].  
00000110 85 7c ff be 85 07 e8 19 00 ff e3 b0 46 e8 24 00 [........u.....].  
00000120 b0 31 00 d0 eb 17 0f ab 56 0c be 78 07 e8 eb ff [....$..S.....[.t].  
00000130 89 fe e8 03 00 be 85 07 ac a8 80 75 05 e8 04 00 [...L...V...F..t.f].  
00000140 eb f6 24 7f 53 bb 07 00 b4 0e cd 10 5b c3 8a 74 [...j.f.t..S..j...H.....].  
00000150 01 8b 4c 02 b0 01 56 89 e7 f6 46 bb 80 74 13 66 [...@.....^.....Defa].  
00000160 6a 00 66 ff 74 08 06 53 6a 01 6a 10 89 e6 48 80 [...ult:.....].  
00000170 cc 40 cd 13 89 fc 5e c3 20 20 a0 0a 44 65 66 61 [...?.....].  
00000180 75 6c 74 3a a0 0d 8a 00 05 0f 01 06 07 0b 0c 0e [...D0.Linu.FreeBS.....].  
00000190 83 a5 a6 a9 0d 0c 0b 0a 09 08 0a 0e 11 10 01 3f [...f.Drive.....].  
000001a0 bf 44 4f d3 4c 69 6e 75 f8 46 72 65 65 42 53 c4 [........].  
000001b0 66 bb 44 72 69 76 65 20 00 00 80 8f b6 00 00 00 [........].  
000001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [........U.....].  
000001f0 00 00 00 00 00 00 00 00 00 00 00 55 aa [........].  
00000200
```

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 instructions

Executable file

→ decoding 为对应的 assembly instruction.

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

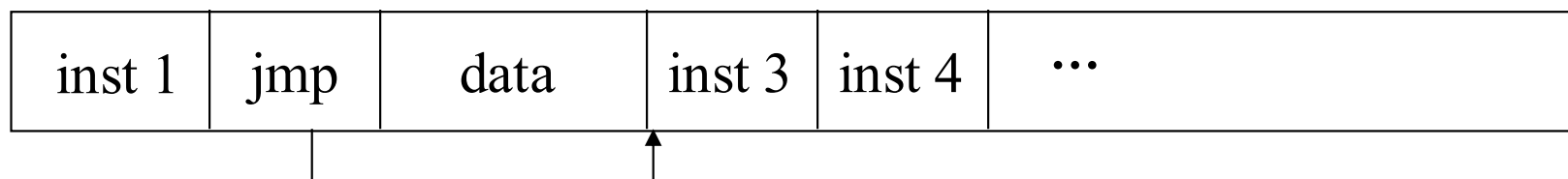
→ 模仿 CPU decode (jump over data bytes)

→ instruction missed (skipped)

↓
hybrid disassembling

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!

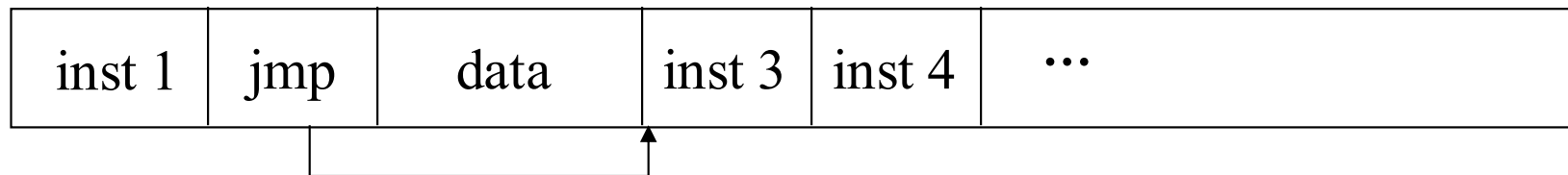
❑ What a “linear” disassembler sees



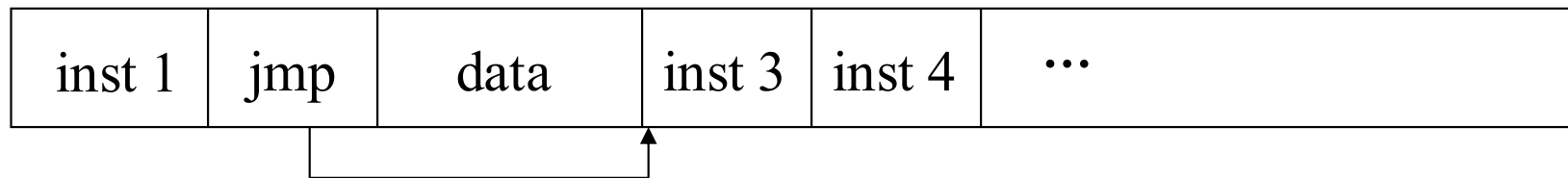
-
- So how do we disassemble this?
- Well, to take one step back, how can CPU correctly figure this out?

Recursive disassembling

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

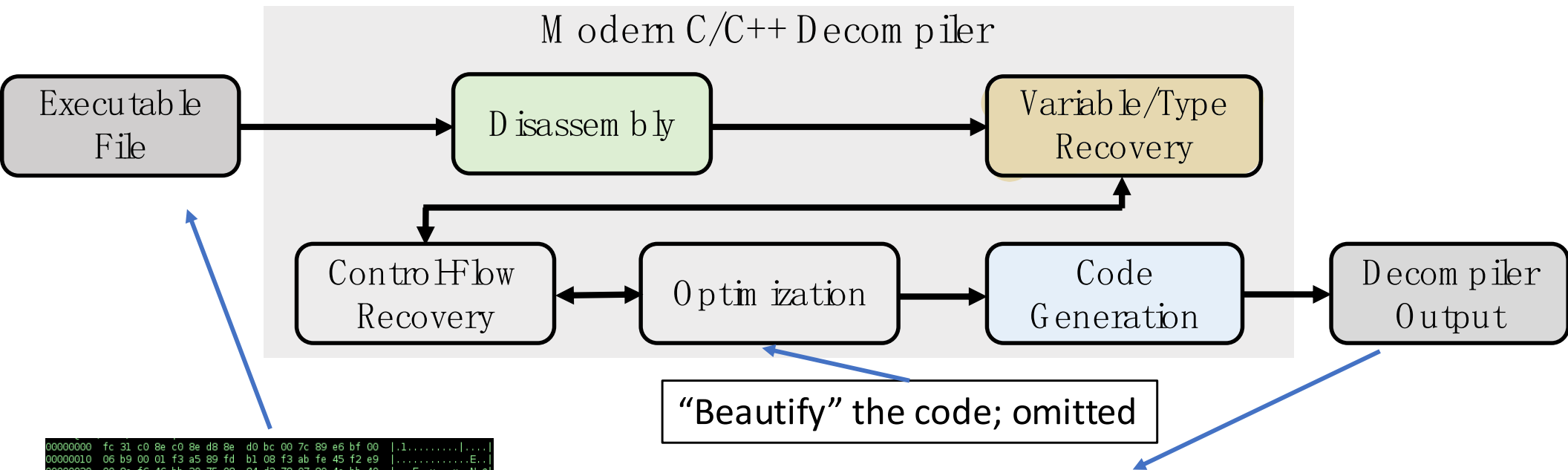


□ What a “recursive” disassembler sees:



Recursive disassembling follow the program control transfers to disassemble

The middle stage (orange boxes)



```
00000000 fc 31 c0 8e c0 8e d8 8e d0 bc 00 7c 89 e6 bf 00 .l.....|
00000010 06 b9 00 01 f3 a5 89 fd b1 08 f3 ab fe 45 f2 e9 .....E.
00000020 00 8a f6 46 bb 20 75 08 84 d2 78 07 80 4e bb 40 .....F. u...x..N.@
00000030 8a 56 ba 88 56 00 e8 fc 00 52 bb c2 07 31 d2 88 .....V. V...R...l.
00000040 6f fc 0f a3 56 bb 73 19 8a 07 bf 87 07 b1 03 f2 .....o. V.s.....
00000050 ae 74 0e b1 0b f2 ae 83 c7 09 8a 0d 01 cf e8 c5 .....t.....
00000060 00 42 80 c3 10 73 d8 58 2c 7f 3a 06 75 04 72 05 .....B...s.X...u.r.
00000070 48 74 0d 30 c0 04 b0 88 46 b8 bf b2 07 e8 a6 00 .....Ht.O...F.....
00000080 be 7b 07 e8 b2 09 8a 56 b9 4e a8 e8 00 eb 05 b0 .....f...V.N.....
00000090 07 e8 b0 00 30 e4 cd 1a 89 d7 03 7e bc b4 01 cd .....u.O...9.r.F..
000000a0 16 75 0d 30 e4 cd 1a 39 fa 72 f2 8a 46 b9 eb 16 .....u.O...<t...<v..
000000b0 30 e4 cd 16 88 e0 3c 1c 74 f1 2c 3b 3c 04 76 06 .....<w...F.s.F..
000000c0 2c c7 3c 04 77 c9 98 0f a3 46 0c 73 c2 88 46 b9 .....<f...<t.....
000000d0 be 00 08 8a 14 89 f3 3c 04 9e 74 0a c0 a0 04 05 .....S.F.@.....
000000e0 be 07 93 c6 07 80 53 f6 46 bb 40 75 08 bb 00 06 .....Y.Y...u.V...O.
000000f0 b4 03 e8 59 00 5e 9d 75 06 8a 56 b8 80 ea 30 bb .....G.R...U...
00000100 00 7c b4 02 e8 47 00 72 86 81 bf fe 01 55 aa 0f .....I...V.x.....
00000110 85 7c ff be 85 07 e8 19 00 ff e3 b0 46 e8 24 00 .....$.S.....[.t
00000120 b0 31 00 d0 eb 17 0f ab 56 0c be 78 07 e8 eb ff .....L...V...F..t.f
00000130 89 fe e8 03 00 be 85 07 ac a8 80 75 05 e8 04 00 .....j.f.t..Sj...H.
00000140 eb f6 24 7f 53 bb 07 00 b4 0e cd 10 5b c3 8a 74 .....e...^...Defa
00000150 01 8b 4c 02 b0 01 56 89 e7 f6 46 bb 80 74 13 66 .....ult.....
00000160 6a 00 66 ff 74 08 06 53 6a 01 6a 10 89 e6 48 80 .....?
00000170 cc 40 cd 13 89 fc 5e c3 20 20 a0 0a 44 65 66 61 .....D.O.Linu.FreeBS.
00000180 75 6c 74 3a a0 0d 8a 00 05 0f 01 06 07 0b 0c 0e .....f.Drive .....
00000190 83 a5 a6 a9 0d 0c 0b 0a 09 08 0a 0e 11 10 01 3f .....
000001a0 bf 44 4f d3 4c 69 6e 75 f8 46 72 65 65 42 53 c4 .....
000001b0 66 bb 44 72 69 76 65 20 00 00 80 8f b6 00 00 00 .....
000001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001f0 00 00 00 00 00 00 00 00 00 00 00 00 55 aa .....U.
00000200
```

“Beautify” the code; omitted

```
MyProjectName.c
+ Name : MyProjectName.c

#include <stdio.h>
#include <stdlib.h>

- int main(void) {
    puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
    return EXIT_SUCCESS;
}
```

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

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

    printf ("hello world!\n");

    return 0;
}
```

C code

Lack of:

int num, i;

type

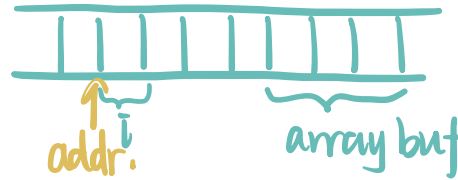
variable

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.*

A Holistic View of Variable

Process Memory Space:
Recovery



⇒ simulate execution of asm code

Program memory space

若有存在某个 addr. 多次被 access
则推测该 addr. 存在 var.

↓
recover var.

byte₁ byte₂ byte₃ byte₄ byte₅ byte₆ byte₇ byte₈ byte₉ ...

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

Program memory space

byte₁ byte₂ | byte₃ byte₄ byte₅ byte₆ | byte₇ byte₈ byte₉ ...

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...

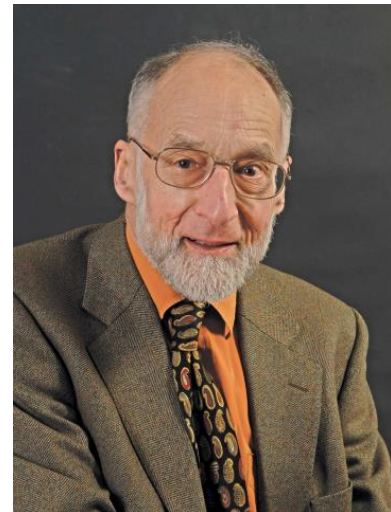
```
a = 0x8040200; // I don't know the type of a  
printf("%d\n", a); // print out an integer
```


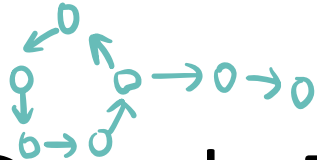
All right, now I know a must be an integer...

```
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.



.asm  →  [CFG] edges
basic block

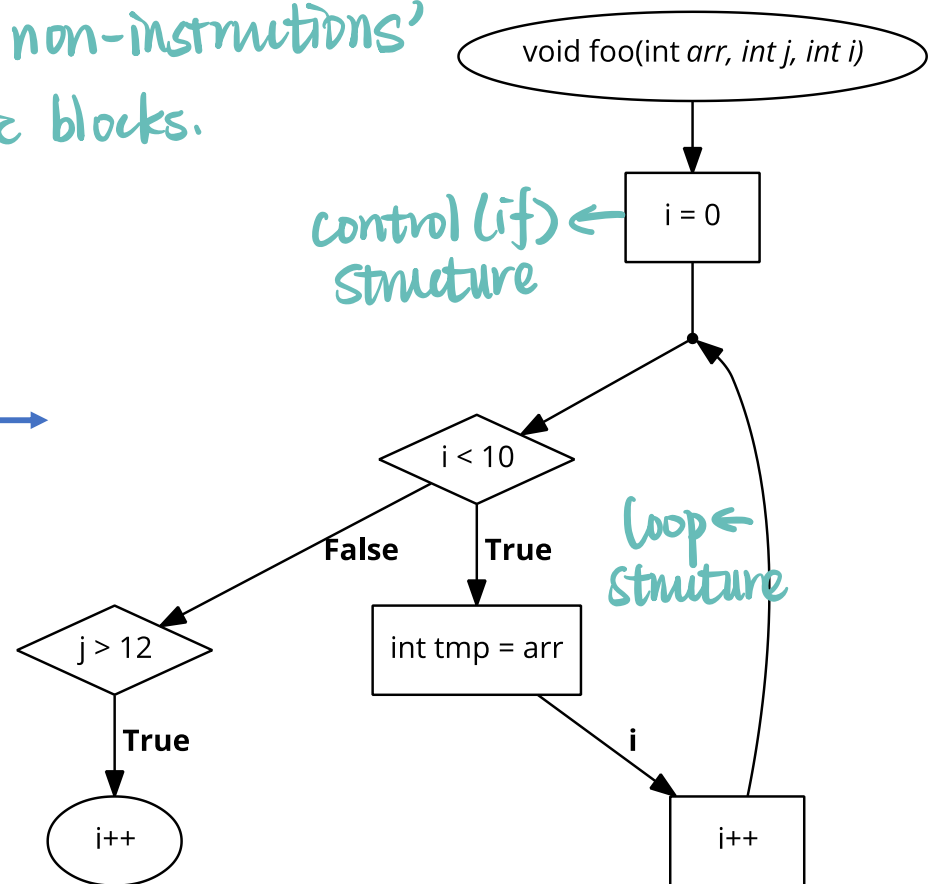
Control Flow Graph Recovery

① Identify jump instruction

② From edges & separate the other non-instructions' in to basic blocks.

Assembly programs

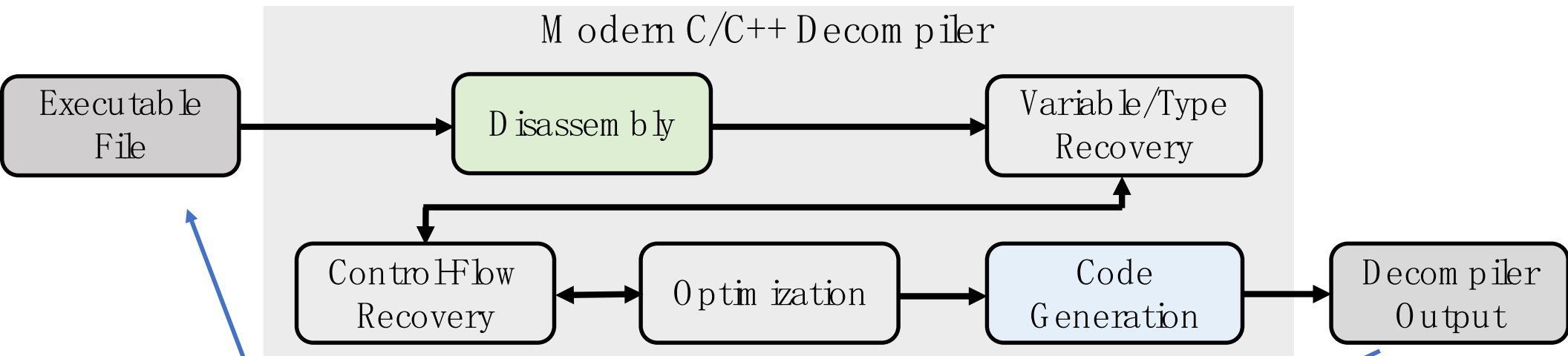
extract



Control-flow graph

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

The backend stage (blue boxes)



```
00000000 fc 31 c0 8e c0 8e d8 8e d0 bc 00 7c 89 e6 bf 00 [.I.....]
00000010 06 b9 00 01 f3 a5 89 fd b1 08 f3 ab fe 45 f2 e9 [...F.u...x..N.@]
00000020 00 8a f6 46 bb 20 75 08 84 d2 78 07 80 4e bb 40 [...V...R...l..]
00000030 8a 56 ba 88 56 00 e8 fc 00 52 bb c2 07 31 d2 88 [o...V.s.....]
00000040 6f fc 0f a3 56 bb 73 19 8a 07 bf 87 07 b1 03 f2 [t.....]
00000050 ae 74 0e b1 0b f2 ae 83 c7 09 8a 0d 01 cf e8 c5 [B...s.X...u.r.]
00000060 00 42 80 c3 10 73 d8 58 2c 7f 3a 06 75 04 72 05 [Ht.O...F.....]
00000070 48 74 0d 30 c0 04 b0 88 46 b8 bf b2 07 e8 a6 00 [f...V.N.....]
00000080 be 7b 07 e8 b2 09 8a 56 b9 4e a8 e8 00 eb 05 b0 [o...O...<.t...<.v.]
00000090 07 e8 b0 00 30 e4 cd 1a 89 d7 03 7e bc b4 01 cd [...<.w...<.f.s..F.]
000000a0 16 75 0d 30 e4 cd 1a 39 fa 72 f2 8a 46 b9 eb 16 [.....S.F.@....]
000000b0 30 e4 cd 16 88 e0 3c 1c 74 f1 2c 3b 3c 04 76 06 [..Y..u...V...O.]
000000c0 2c c7 3c 04 77 c9 98 0f a3 46 0c 73 c2 88 46 b9 [.....G.R.....U.]
000000d0 be 00 08 8a 14 89 f3 3c 04 9e 74 0a c0 a0 04 05 [I.....V..x....]
000000e0 0e 07 93 c6 07 80 53 f6 46 bb 40 75 08 bb 00 06 [..$.S.....[.t]
000000f0 b4 03 e8 59 00 5e 9d 75 06 8a 56 b8 80 ea 30 bb [..L...V...F..t.f]
00000100 00 7c b4 02 e8 47 00 72 86 81 bf fe 01 55 aa 0f [j.f.t..Sj...H.]
00000110 85 7c ff be 85 07 e8 19 00 ff e3 b0 46 e8 24 00 [e...<.u...Defa]
00000120 b0 31 00 d0 eb 17 0f ab 56 0c be 78 07 e8 eb ff [ult:.....]
00000130 89 fe e8 03 00 be 85 07 ac a8 80 75 05 e8 04 00 [.....?]
00000140 ef f6 24 7f 53 bb 07 00 b4 0e cd 10 5b c3 8a 74 [..D.O.Linu.FreeBS.]
00000150 01 8b 4c 02 b0 01 56 89 e7 f6 46 bb 80 74 13 66 [f.Drive.....]
00000160 6a 00 66 ff 74 08 06 53 6a 01 6a 10 89 e6 48 80 [.....]
00000170 cc 40 cd 13 89 fc 5e c3 20 20 a0 0a 44 65 66 61 [.....]
00000180 75 6c 74 3a a0 0d 8a 00 05 0f 01 06 07 0b 0c 0e [.....]
00000190 83 a5 a6 a9 0d 0c 0b 0a 09 08 0a 0e 11 10 01 3f [.....]
000001a0 bf 44 4f d3 4c 69 6e 75 f8 46 72 65 65 42 53 c4 [.....]
000001b0 66 bb 44 72 69 76 65 20 00 00 80 8f b6 00 00 00 [.....]
000001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [.....]
000001f0 00 00 00 00 00 00 00 00 00 00 00 00 55 aa [.....U.]
00000200
```

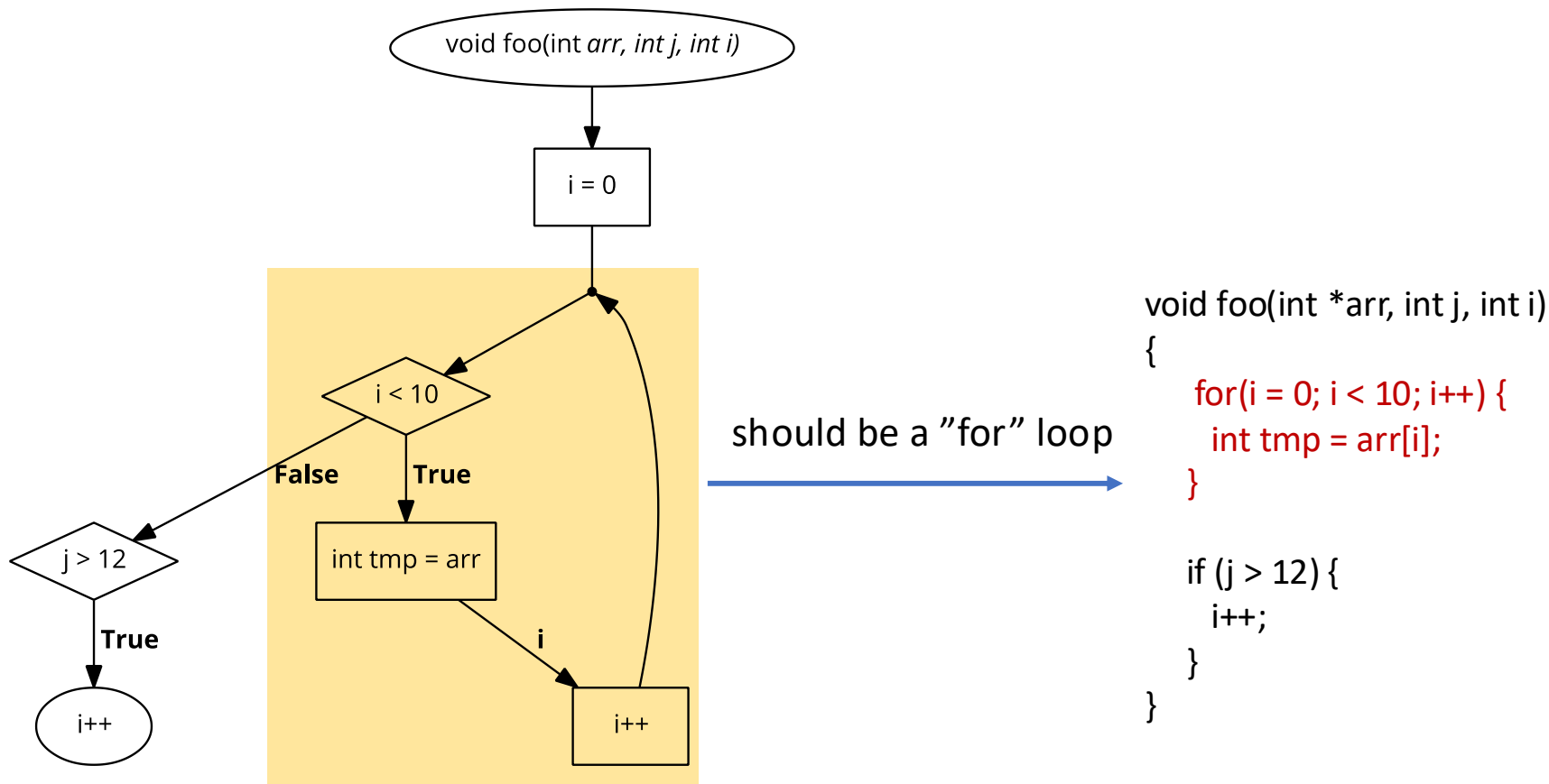
```
MyProjectName.c

+ Name      : MyProjectName.c

#include <stdio.h>
#include <stdlib.h>

- int main(void) {
    puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
    return EXIT_SUCCESS;
}
```

Code Generation with Pattern Matching and Concretize Code Templates



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 C Smith

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

C code decompiled by A Commercial Tool

(a)

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

C code generated by C Smith

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

C code decompiled by A Commercial Tool

(b)

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

C code generated by C Smith

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

C code decompiled by A Commercial Tool

(c)

Note: all the decompiled code has been simplified for readability.

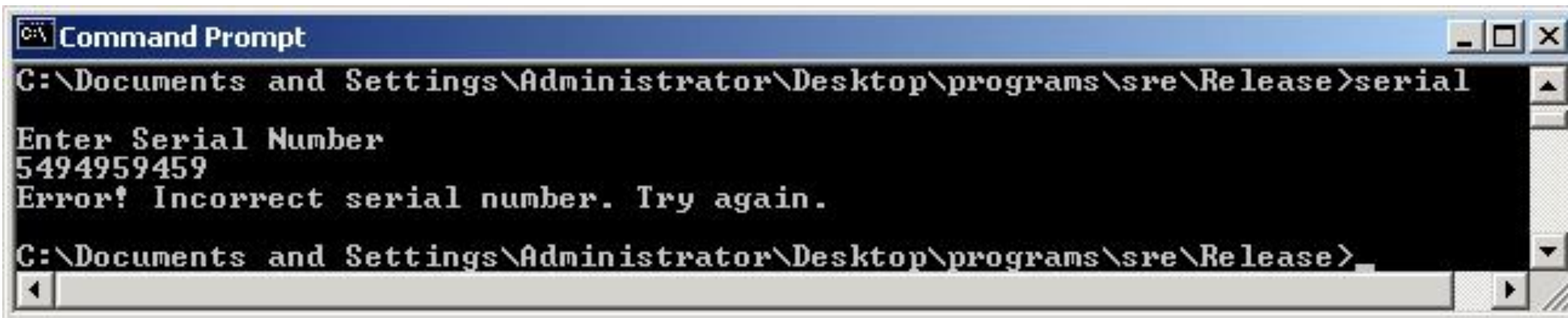
For case (a), the **for** loop in the decompiled output will be iterated from any times.

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...




```
Command Prompt
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 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, "????") = ) {  
    printf("Error! Incorrect series number. Try again.");  
}
```



code is simplified.

Reverse Engineering the (Victim) Software

- disassembly

```
.text:00401003
.text:00401008
.text:0040100D
.text:00401011
.text:00401012
.text:00401017
.text:0040101C
.text:0040101E
.text:00401022
.text:00401027
.text:00401028
.text:0040102D
.text:00401030
.text:00401032
.text:00401034
.text:00401039


push    offset aEnterSerialNum ; "\nEnter Serial Number\n"
call    sub_4010AF               ← print
lea     eax, [esp+18h+var_14]
push    eax
push    offset aS                ; "%s"
call    sub_401098               ← read_input
push    8
lea     ecx, [esp+24h+var_14]
push    offset aS123n456 ; "S123N456"
push    ecx
call    sub_401060               ← cmp
add     esp, 18h
test    eax, eax
jz      short loc_401045
push    offset aErrorIncorrect ; "Error! Incorrect serial number."
call    sub_4010AF               ← 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

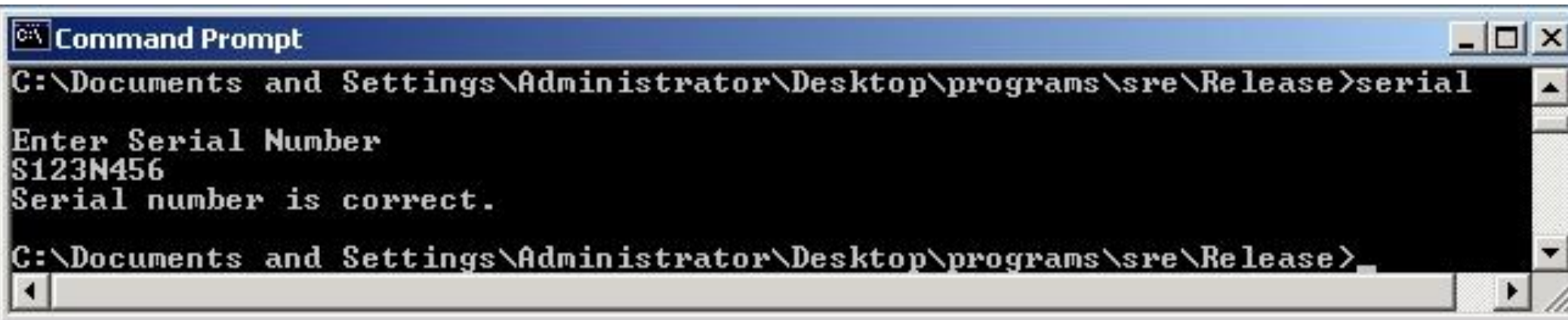
```
printf("Enter Series Number");  
scanf(x);  
If(is_equal(x, S123N456) == False) {  
    printf("Error! Incorrect series number. Try again.");  
}
```



code is simplified.

SRE Example

- Try the serial number S123N456


A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The command prompt shows the current directory as "C:\Documents and Settings\Administrator\Desktop\programs\sre\Release". The user has entered the command "serial". The program responds with "Enter Serial Number", followed by the input "S123N456", and then the message "Serial number is correct." The prompt returns to the command line.


```
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 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, "????") = ) {  
    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 C6 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

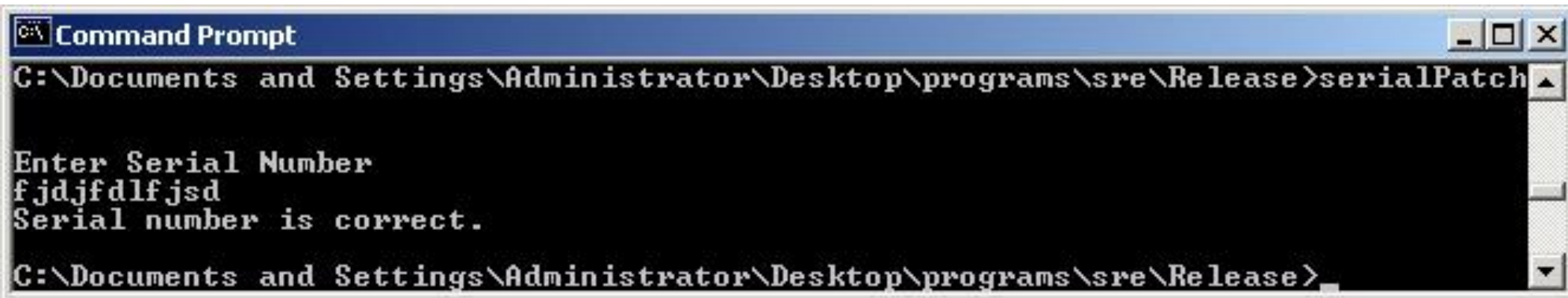
```
00001010h: 04 50 68 84 80 40 00 E8 7C 00 00 00 6A 08 8D 4C
00001020h: 24 10 68 78 80 40 00 51 E8 33 00 00 00 83 C4 18
00001030h: 85 C0 74 11 68 4C 80 40 00 E8 71 00 00 00 83 C4
00001040h: 04 83 C4 14 C3 68 30 80 40 00 E8 60 00 00 00 83
00001050h: C4 04 83 C4 14 C3 90 90 90 90 90 90 90 90 90
```

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
```

□ Save as serialPatch.exe

SRE Example



A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The command prompt shows the current directory as "C:\Documents and Settings\Administrator\Desktop\programs\sre\Release". The user has entered the command "serialPatch". The program then prompts "Enter Serial Number", the user enters "fjdjfdlfjsd", and the program outputs "Serial number is correct." The prompt returns to "C:\Documents and Settings\Administrator\Desktop\programs\sre\Release>".

```
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 from now on...

SRE Example

- Back to disassembly...

serial.exe

```
.text:00401003  
.text:00401008  
.text:0040100D  
.text:00401011  
.text:00401012  
.text:00401017  
.text:0040101C  
.text:0040101E  
.text:00401022  
.text:00401027  
.text:00401028  
.text:0040102D  
.text:00401030  
.text:00401032  
.text:00401034  
.text:00401039
```

```
push    offset aEnterSerialNum ; "\nEnter Serial Number\n"  
call    sub_4010AF  
lea     eax, [esp+18h+var_14]  
push    eax  
push    offset aS               ; "%5"  
call    sub_401098  
push    8  
lea     ecx, [esp+24h+var_14]  
push    offset aS123n456 ; "S123N456"  
push    ecx  
call    sub_401060  
add     esp, 18h  
test    eax, eax  
jz      short loc_401045  
push    offset aErrorIncorrect ; "Error! Incorrect serial number."  
call    sub_4010AF
```

serialPatch.exe

```
.text:00401003  
.text:00401008  
.text:0040100D  
.text:00401011  
.text:00401012  
.text:00401017  
.text:0040101C  
.text:0040101E  
.text:00401022  
.text:00401027  
.text:00401028  
.text:0040102D  
.text:00401030  
.text:00401032  
.text:00401034  
.text:00401039
```

```
push    offset aEnterSerialNum ; "\nEnter Serial Number\n"  
call    sub_4010AF  
lea     eax, [esp+18h+var_14]  
push    eax  
push    offset aS               ; "%5"  
call    sub_401098  
push    8  
lea     ecx, [esp+24h+var_14]  
push    offset aS123n456 ; "S123N456"  
push    ecx  
call    sub_401060  
add     esp, 18h  
xor     eax, eax  
jz      short loc_401045  
push    offset aErrorIncorrect ; "Error! Incorrect serial number."  
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...