Hong Kong Cyber Security New Generation

Capture the Flag (CTF) Challenge 2022

Webinar 2

CTF 2022

香港網絡保安新生代
Hong Kong Cyber Security New Generation
奪旗挑戰賽
Capture the Flag Challenge

Register now!

https://ctf.hkcert.org/

Sample Challenges:
https://training.hkcert22.pwnable.hk/

Discord channel:
https://discord.gg/V6QGvWCmDm

Speaker Bio

cire meat pop

- Pwn / RE player @ Black Bauhinia
- Security Software Engineer

b6a.black
@blackb6a
@blackb6a
/team/83678

Cap. 200 Crimes Ordinance

161.Access to computer with criminal or dishonest intent

(1) Any person who obtains access to a computer—

(a) with intent to commit an offence;

(b) with a dishonest intent to deceive;

(c) with a view to dishonest gain for himself or another; or

(d) with a dishonest intent to cause loss to another,

whether on the same occasion as he obtains such access or on any future occasion, commits an offence and is liable on conviction upon indictment to imprisonment for 5 years.

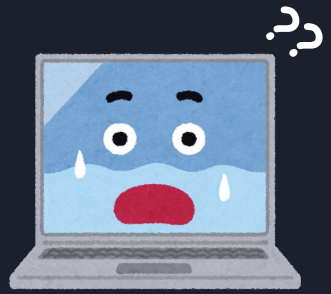
(2) For the purposes of subsection (1) gain (獲益) and loss (損失) are to be construed as extending not only to gain or loss in money or other property, but as extending to any such gain or loss whether temporary or permanent; and—

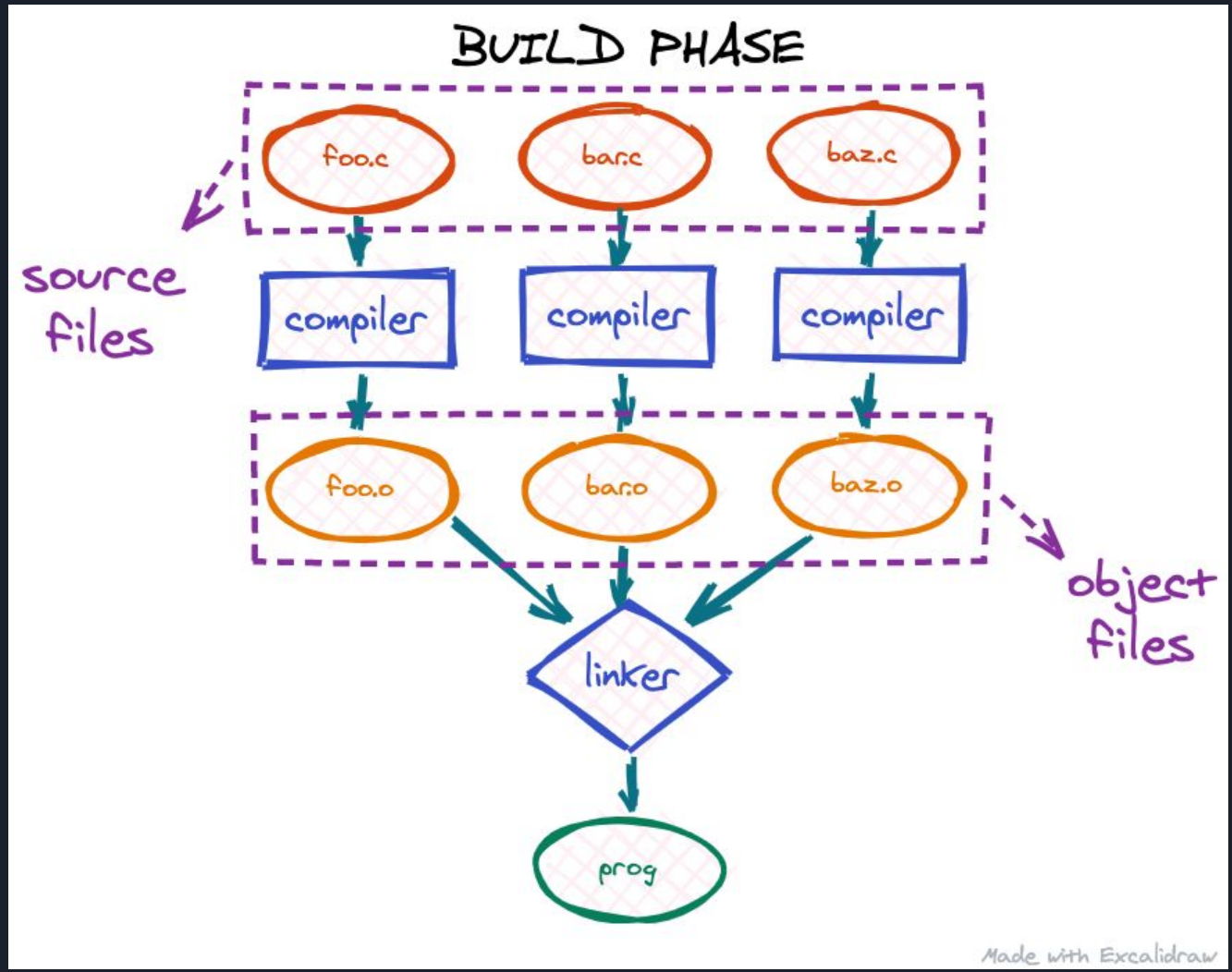
(a) gain (獲益) includes a gain by keeping what one has, as well as a gain by getting what one has not; and

(b) loss (損失) includes a loss by not getting what one might get, as well as a loss by parting with what one has.

How computers work?

Print “Hello World” on screen




BUILD PHASE
foo.c
bar.c
baz.c
source files
compiler
compiler
compiler
foo.o
bar.o
baz.o
object files
linker
prog
Made with Excalidraw

Program / Application

Pwn

Can I make the program do something unexpected?

Reverse Engineering

What will this program do?

Pwn & Reverse Engineering

- An important field in Computer Science, as well as in CTF
- Require low-level computer knowledge to master them

Reverse Engineering

逆向工程，又稱反向工程，是一種技術過程，即對一專案標 產品進行逆向分析及研究，從而演繹並得出該 產品的處理流程、組織結構、功能效能規格等設計要素，以製作出功能相近，但又不完全一樣的 產品。逆向工程源於商業及軍事領域中的硬體分析。其主要目的是，在無法輕易獲得必要的生 產資訊下，直接從成品的分析，推導 產品的設計原理。維基百科

Forward Enginnering

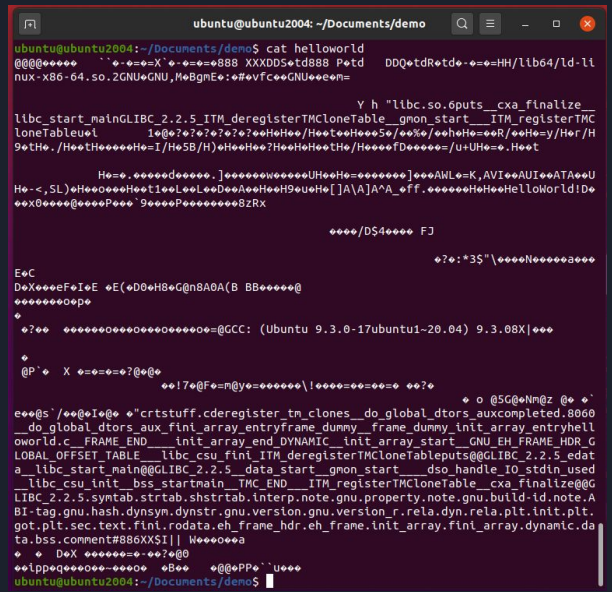
Source code

```c
#include <stdio.h>

void main() {
    printf("HelloWorld!\n");
}
```

Compile

Machine Code



Execute

```
ubuntu@ubuntu2004:~/Documents/demo$ ./helloworld
HelloWorld!
ubuntu@ubuntu2004:~/Documents/demo$
```

Reverse Enginnering

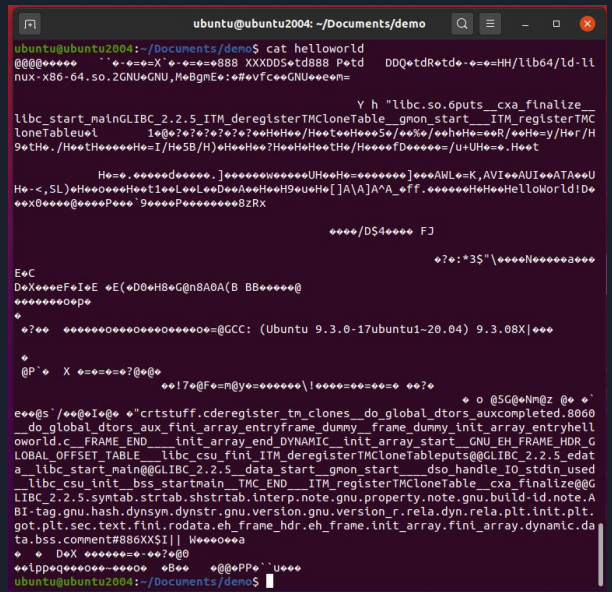
Source code

```
#include <stdio.h>

void main() {
    printf("HelloWorld!\n");
}
```

Reverse Engineering

Machine Code



Execute

```
ubuntu@ubuntu2004:~/Documents/demo$ ./helloworld
HelloWorld!
ubuntu@ubuntu2004:~/Documents/demo$
```

Why Reverse Engineering?

- Real world applications:
  - Code reconstruction / debugging
  - Malware analysis
  - Useful technique in future career

- Competitive Programming in Cybersecurity
  - Needed for reversing challenges
  - Needed for understanding program flow and execution of other category

Tools for Reverse Engineering

Famous:

- IDA
- Ghidra
- Binary Ninja

Others:

- Cutter
- Hopper

IDA

https://hex-rays.com/ida-free/

Download your IDA Free

The freeware version of IDA v7.6 comes with the following limitations:

- no commercial use is allowed
- lacks all features introduced in IDA > v7.6
- cloud-based decompiler lacks certain advanced commands
- lacks support for many processors, file formats, etc...
- comes without technical support

 IDA Freeware for Windows (71MB)

 IDA Freeware for Linux (63MB)

 IDA Freeware for Mac (59MB)

 IDA Freeware for Mac ARM (61MB)

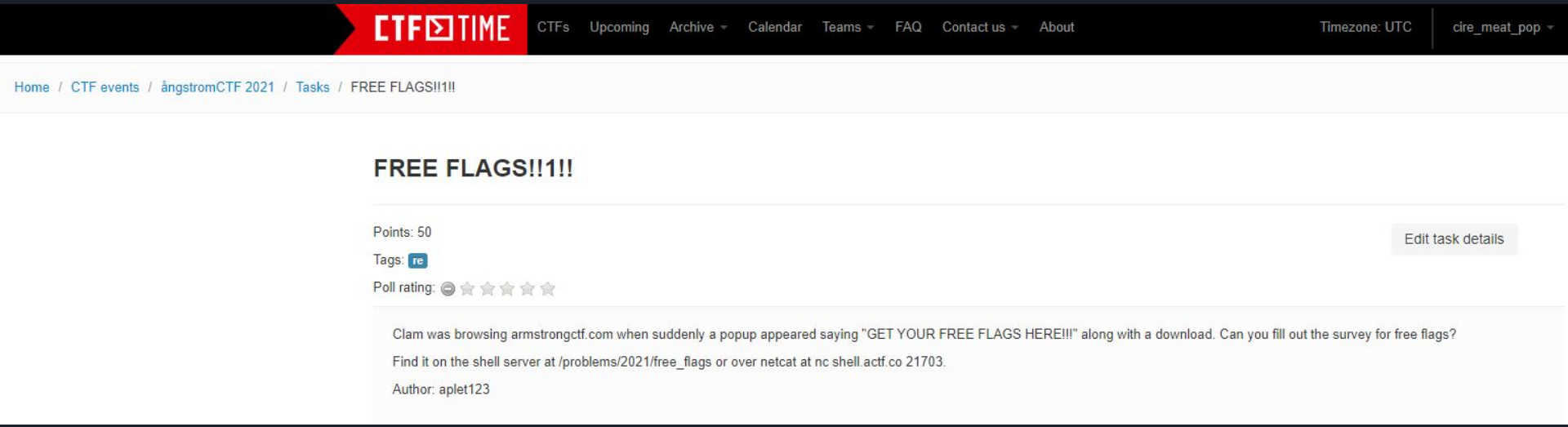
SHA1 checksums:

```
ae0d7266f59bbc36094dbb15e84d1fa1db61120f  arm_idafree76_mac.app.zip
317613188aa7250bfd98d8d2948614ed6dfa6d9a  idafree76_linux.run
95cd7b918bb2aa47fb10d05c606609a0e91de285  idafree76_mac.app.zip
c0a53f1a6841def6eacd8e67239b2122ef423985  idafree76_windows.exe
```

Free license

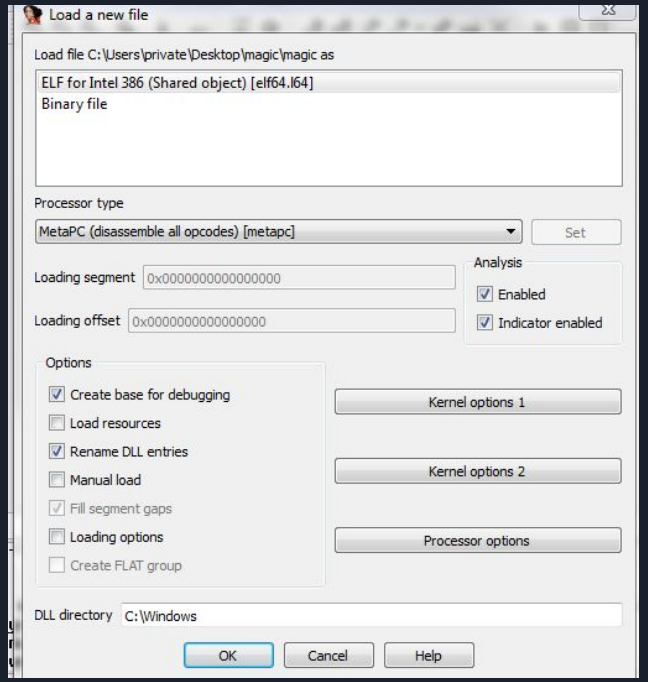
ångstromCTF 2021 - FREE FLAGS!!1!!

- free_flags: https://files.actf.co/6ddcb4e935b82c477140ee6833eceaf1149e0c732af1ba742a9e67db98693f88/free_flags

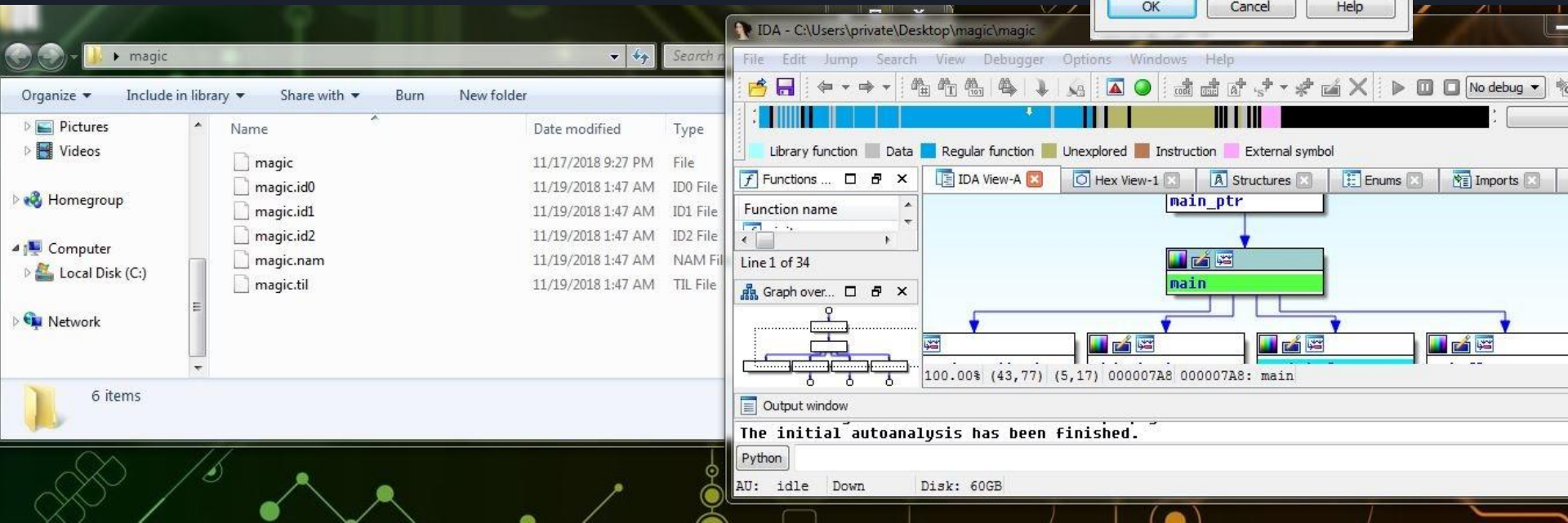


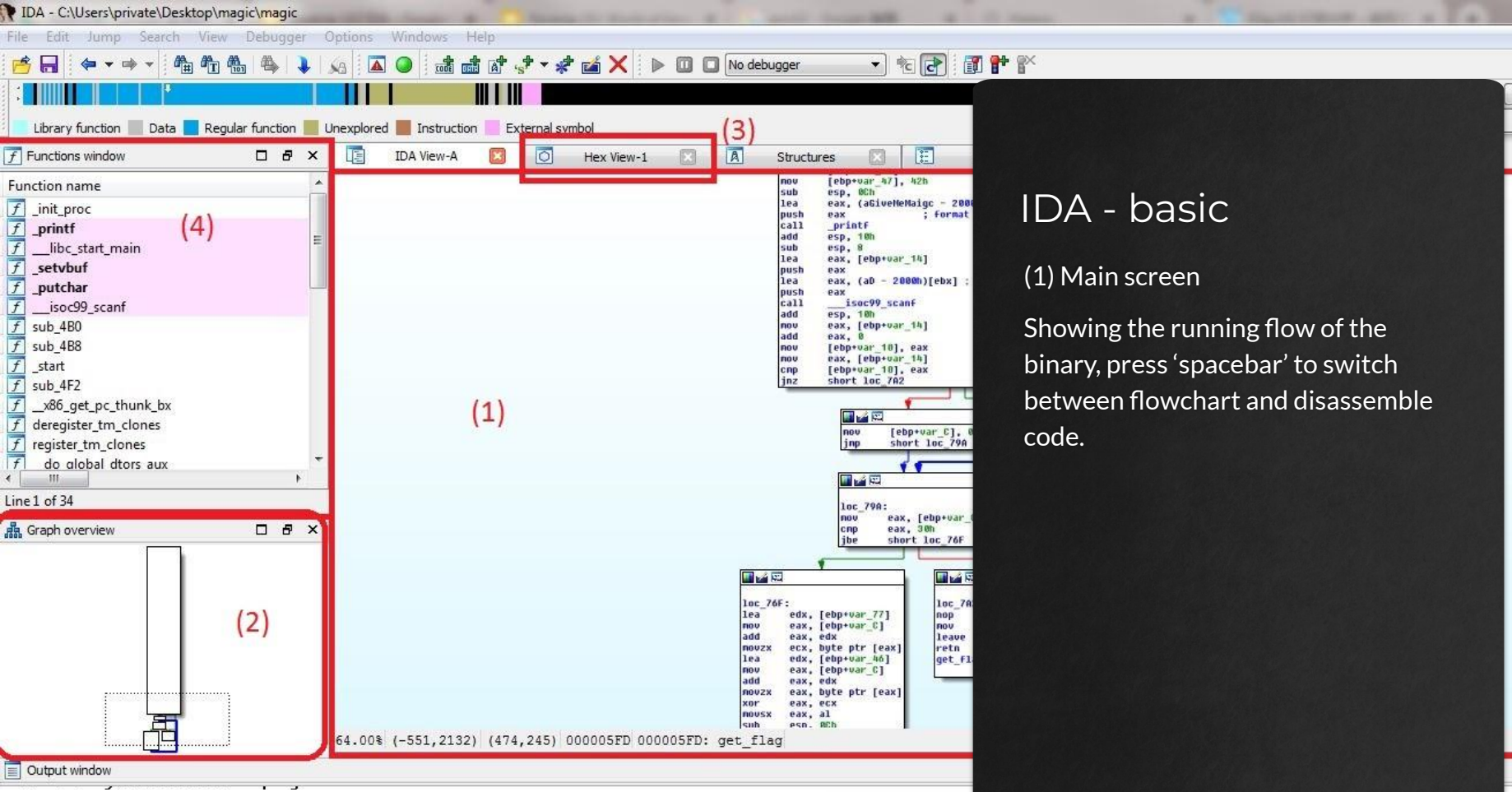
IDA - Basic

- Can load program
  - ELF (in Linux)
  - PE (in Windows) aka. .exe
- Pay attention on arch of the binary
  - Decompiler plugin limitation
  - AMD_i386, AMD_x64
  - ARM32, ARM64



IDA - basic

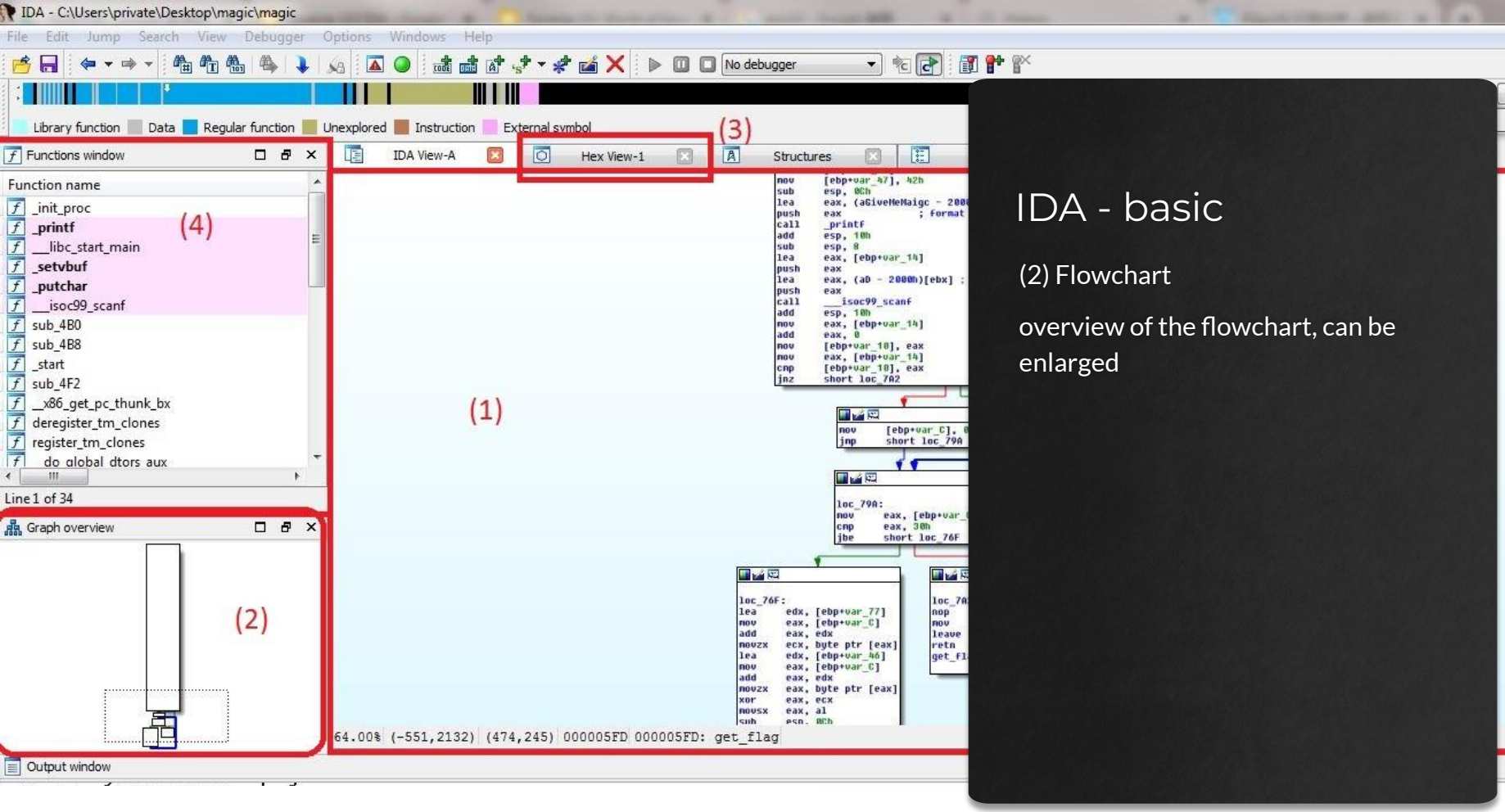

Save database
IDA will save all changes to the disk.
Don't pack database
Pack database (Store)
Pack database (Deflate)
Collect garbage
DON'T SAVE the database
OK
Cancel
Help
magic
Organize
Include in library
Share with
Burn
New folder
Pictures
Videos
Homegroup
Computer
Local Disk (C:)
Network
Name
Date modified
Type
magic
magic.id0
magic.id1
magic.id2
magic.nam
magic.til
11/17/2018 9:27 PM
11/19/2018 1:47 AM
11/19/2018 1:47 AM
11/19/2018 1:47 AM
11/19/2018 1:47 AM
11/19/2018 1:47 AM
File
ID0 File
ID1 File
ID2 File
NAM File
TIL File
6 items
IDA - C:\Users\private\Desktop\magic\magic
File Edit Jump Search View Debugger Options Windows Help
No debug
Library function
Data
Regular function
Unexplored
Instruction
External symbol
Functions ...
Function name
Line 1 of 34
Graph over...
IDA View-A
Hex View-1
Structures
Enums
Imports
main_ptr
main
100.00% (43,77) (5,17) 000007A8 000007A8: main
Output window
The initial autoanalysis has been finished.
Python
AU: idle
Down
Disk: 60GB



IDA - basic

(1) Main screen

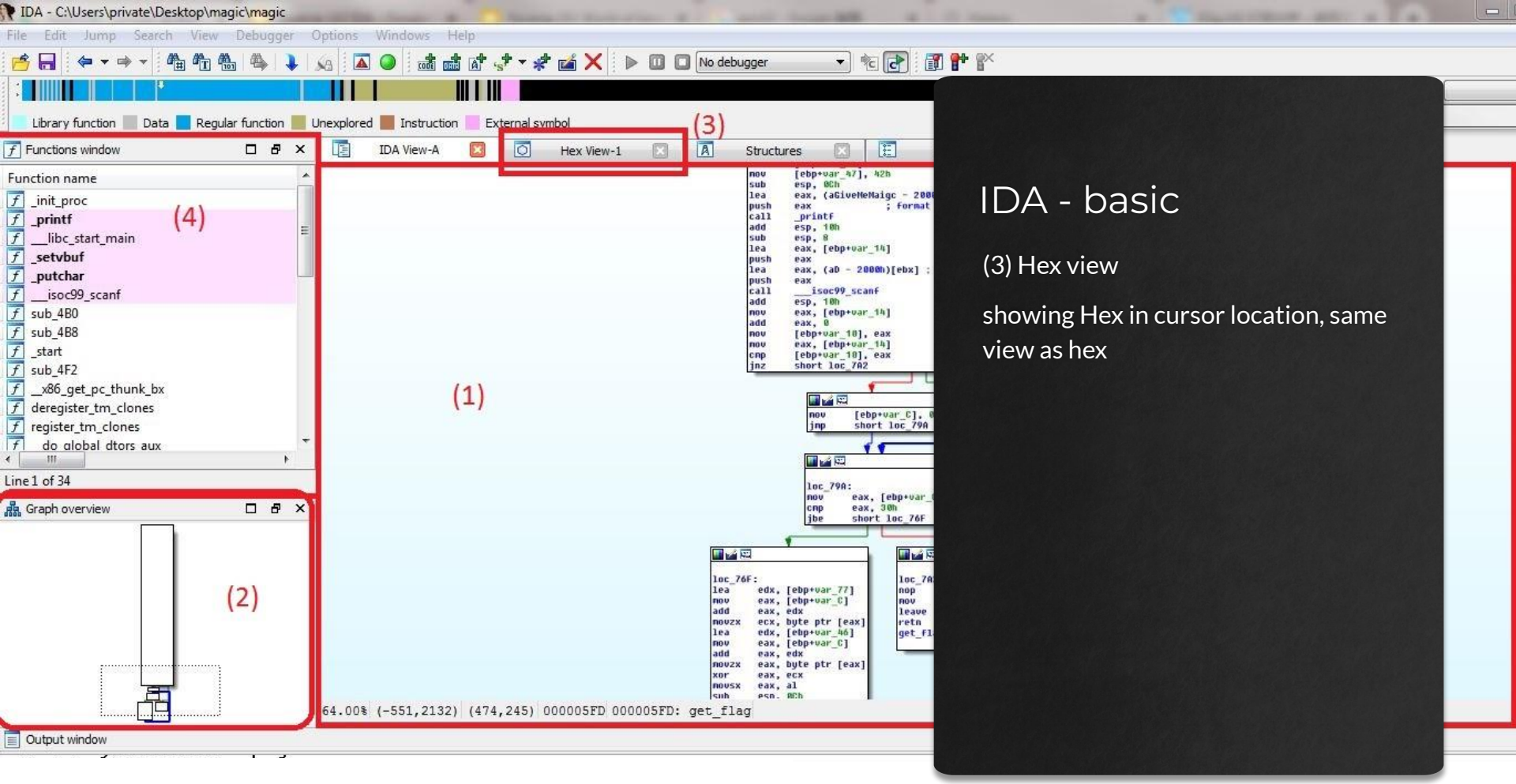
Showing the running flow of the binary, press ‘spacebar’ to switch between flowchart and disassemble code.



IDA - basic

(2) Flowchart

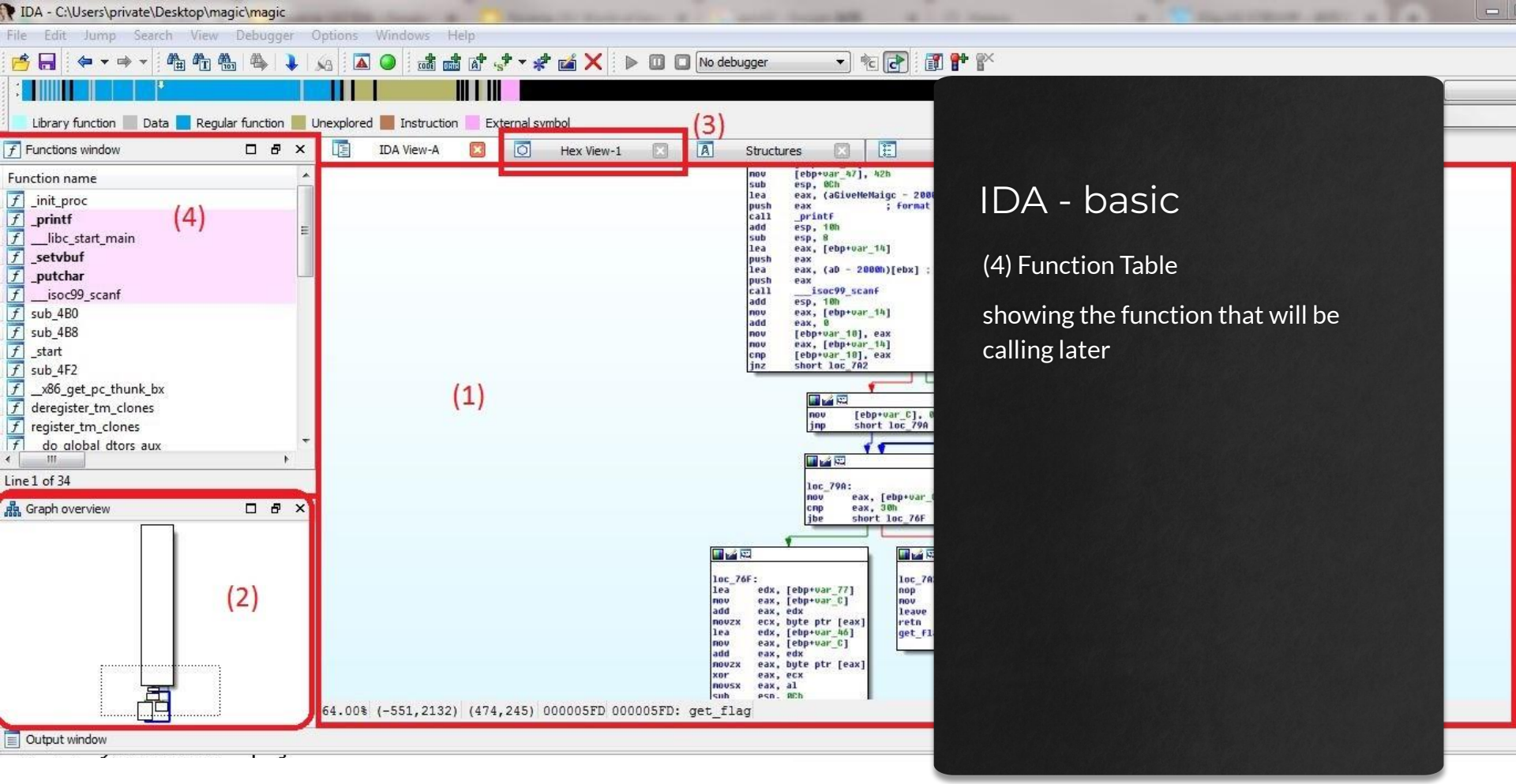
overview of the flowchart, can be enlarged



IDA - basic

(3) Hex view

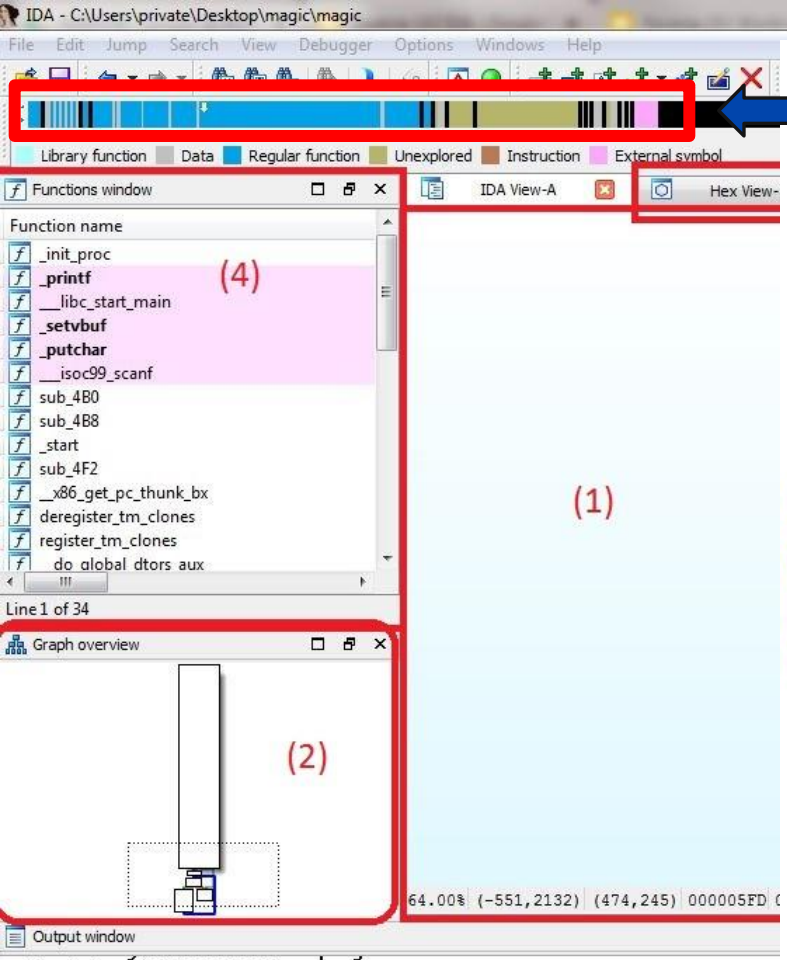
showing Hex in cursor location, same view as hex

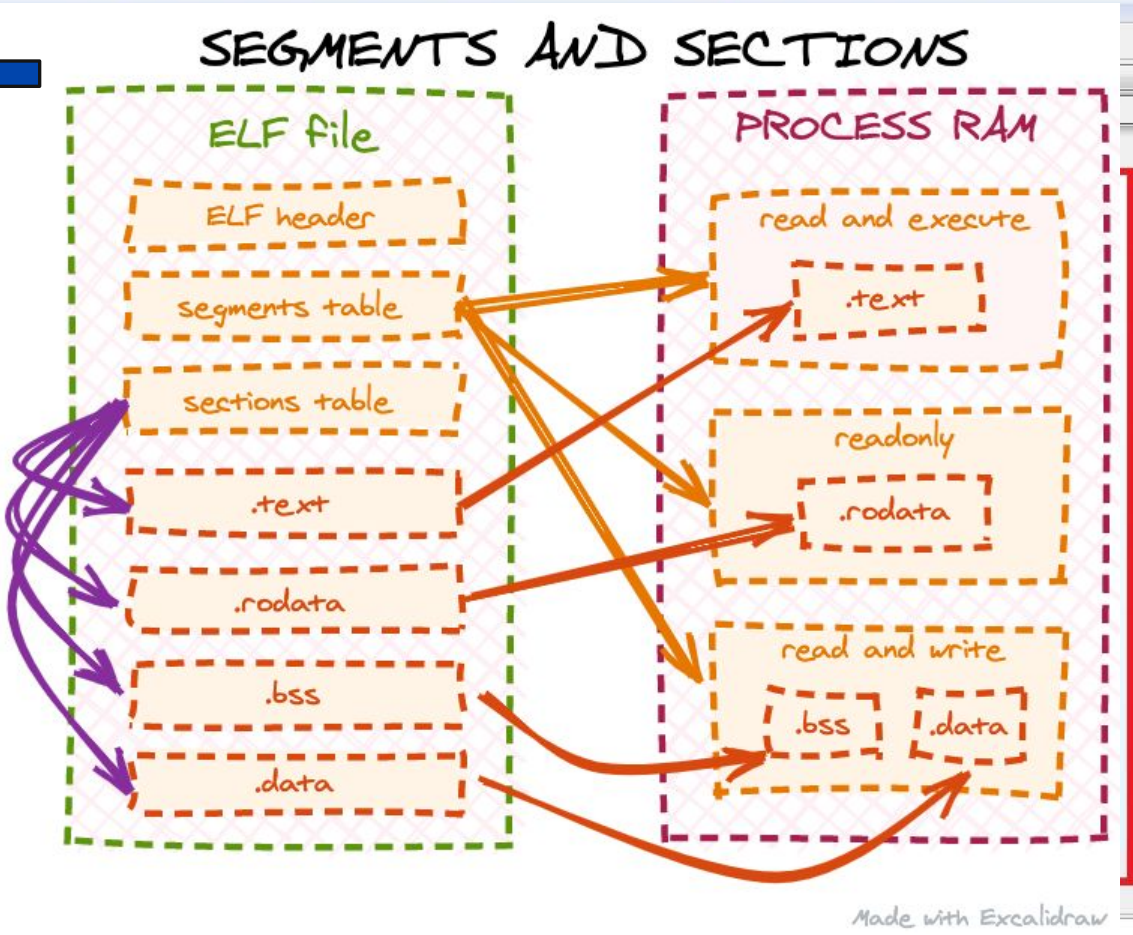


IDA - basic

(4) Function Table

showing the function that will be calling later


IDA - C:\Users\private\Desktop\magic\magic
File Edit Jump Search View Debugger Options Windows Help
Library function Data Regular function Unexplored Instruction External symbol
Functions window
Function name
_init_proc
_printf
__libc_start_main
_setvbuf
_putchar
__isoc99_scanf
sub_4B0
sub_4B8
_start
sub_4F2
__x86_get_pc_thunk_bx
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
(4)
Line 1 of 34
IDA View-A
Hex View-
(1)
Graph overview
(2)
64.00% (-551,2132) (474,245) 000005FD
Output window


SEGMENTS AND SECTIONS
ELF file
ELF header
segments table
sections table
.text
.rodata
.bss
.data
PROCESS RAM
read and execute
.text
readonly
.rodata
read and write
.bss
.data
Made with Excalidraw

Entry point

- Run it
  - Check the program's behaviours
- Main function
  - First function executed by the program
  - ctrl + e
- String Table (short-cut: shift+F12)
  - Strings appear in program

Decompile

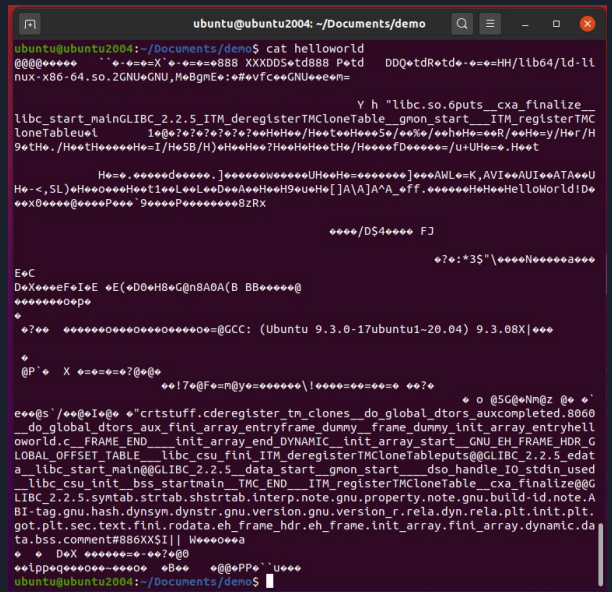
Source code

```c
#include <stdio.h>

void main() {
    printf("HelloWorld!\n");
}
```

Compile

Machine code

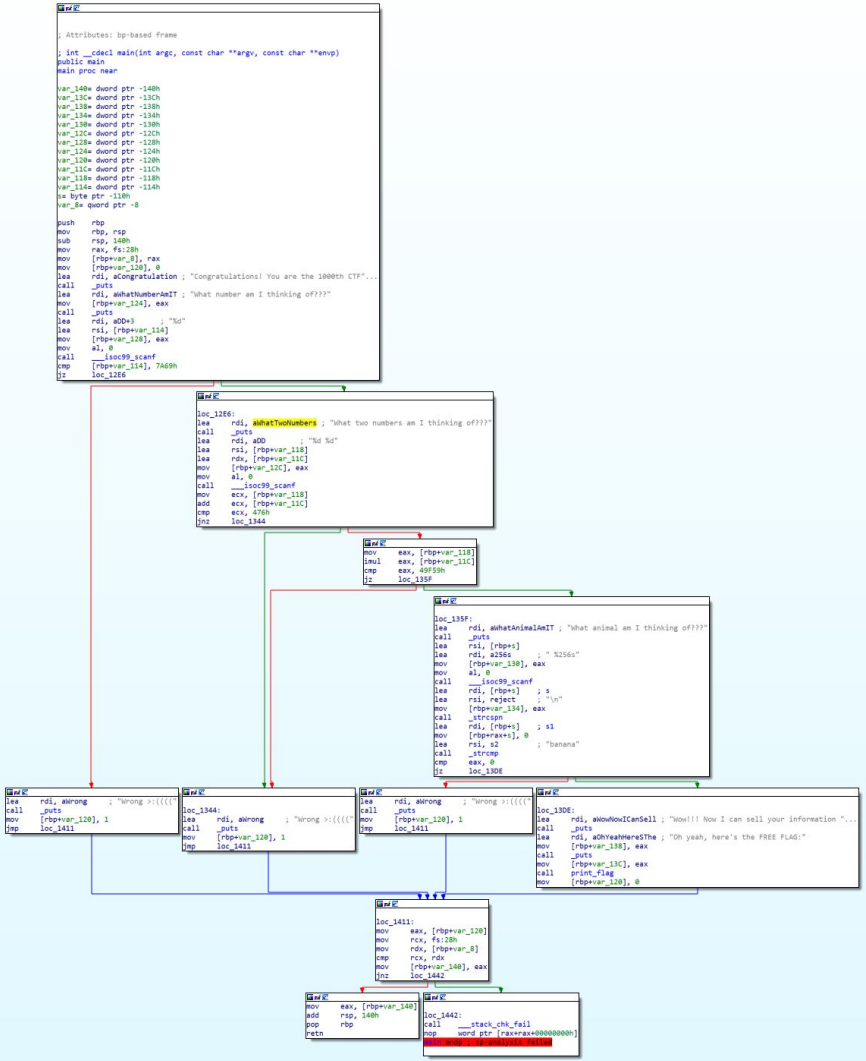


Guessed Source code

```c
#include <stdio.h>

void main() {
    printf("HelloWorld!\n");
}
```

Decompile



IDA - F5

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int result; // eax
4    int v4; // [rsp+20h] [rbp-120h]
5    int v5; // [rsp+24h] [rbp-11Ch] BYREF
6    int v6; // [rsp+28h] [rbp-118h] BYREF
7    int v7; // [rsp+2Ch] [rbp-114h] BYREF
8    char s[264]; // [rsp+30h] [rbp-110h] BYREF
9    unsigned __int64 v9; // [rsp+138h] [rbp-8h]
10
11   v9 = __readfsqword(0x28u);
12   puts("Congratulations! You are the 1000th CTFer!!! Fill out this short survey to get FREE FLAGS!!!");
13   puts("What number am I thinking of???");
14   __isoc99_scanf("%d", &v7);
15   if ( v7 == 31337 )
16   {
17     puts("What two numbers am I thinking of???");
18     __isoc99_scanf("%d %d", &v6, &v5);
19     if ( v5 + v6 == 1142
20       && v5 * v6 == 302937
21       && (puts("What animal am I thinking of???"),
22           __isoc99_scanf(" %256s", s),
23           s[strcspn(s, "\n")] = 0,
24           !strcmp(s, "banana")) )
25     {
26       puts("Wow!!! Now I can sell your information to the Russian government!!!");
27       puts("Oh yeah, here's the FREE FLAG:");
28       print_flag();
29       v4 = 0;
30     }
31     else
32     {
33       puts("Wrong >:((((");
34       v4 = 1;
35     }
36   }
37   else
38   {
39     puts("Wrong >:((((");
40     v4 = 1;
41   }
42   if ( __readfsqword(0x28u) == v9 )
43     return v4;
44   _libc_csu_init();
45   return result;
46 }
```

Reversing...

What can you do?

- Understand the code => Guess the logic and doing Math
- Find unknown function => Google the function name
- Mark you analysis result => Rename function && Commenting


Demo

Q&A

Another Example

flag_generator: https://drive.google.com/file/d/1sxVr0dMH9ylS5d7cn2Tevzy4G5eSqruv/view

- Run flag_generator
- Open flag_generator in IDA:

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    char v4; // [rsp+3h] [rbp-1Dh]
4    int i; // [rsp+4h] [rbp-1Ch]
5    __int64 seconds; // [rsp+8h] [rbp-18h]
6
7    srand(0x31337u);
8    seconds = 1LL;
9    for ( i = 0; i < strlen(secret); ++i )
10   {
11     v4 = rand();
12     putchar(v4 ^ (unsigned __int8)secret[i]);
13     fflush(stdout);
14     sleep(seconds);
15     seconds *= 2LL;
16   }
17   putchar(10);
18   return 0;
19 }
```

Patching

Patching : Modify Program behaviour

✓ : Modify

✗ : Insert/Delete

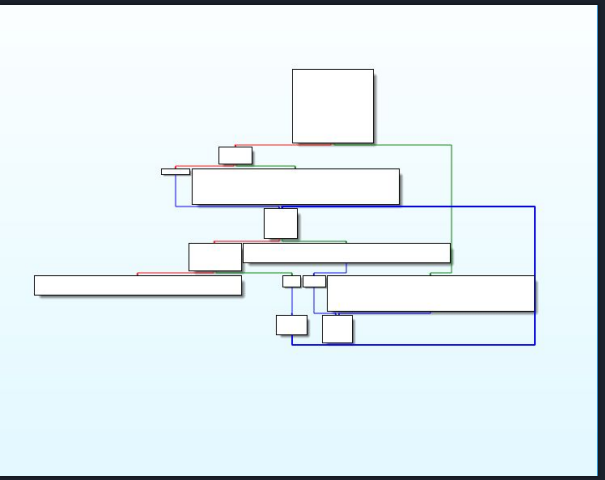
```
text:000000000000123D 48 89 C7                mov     rdi, rax        ; stream
text:0000000000001240 E8 8B FE FF FF          call    _fflush
text:0000000000001245 48 8B 45 E8             mov     rax, [rbp+seconds]
text:0000000000001249 89 C7                   mov     edi, eax        ; seconds
text:000000000000124B E8 90 FE FF FF          call    _sleep
text:0000000000001250 48 D1 65 E8             shl     [rbp+seconds], 1
text:0000000000001254 83 45 E4 01             add     [rbp+var_1C], 1
```

```
text:000000000000123D 48 89 C7                mov     rdi, rax        ; stream
text:0000000000001240 E8 8B FE FF FF          call    _fflush
text:0000000000001245 48 8B 45 E8             mov     rax, [rbp+seconds]
text:0000000000001249 89 C7                   mov     edi, eax        ; seconds
text:000000000000124B 90                      nop
text:000000000000124C 90                      nop
text:000000000000124D 90                      nop
text:000000000000124E 90                      nop
text:000000000000124F 90                      nop
text:0000000000001250 48 D1 65 E8             shl     [rbp+seconds], 1
text:0000000000001254 83 45 E4 01             add     [rbp+var_1C], 1
```

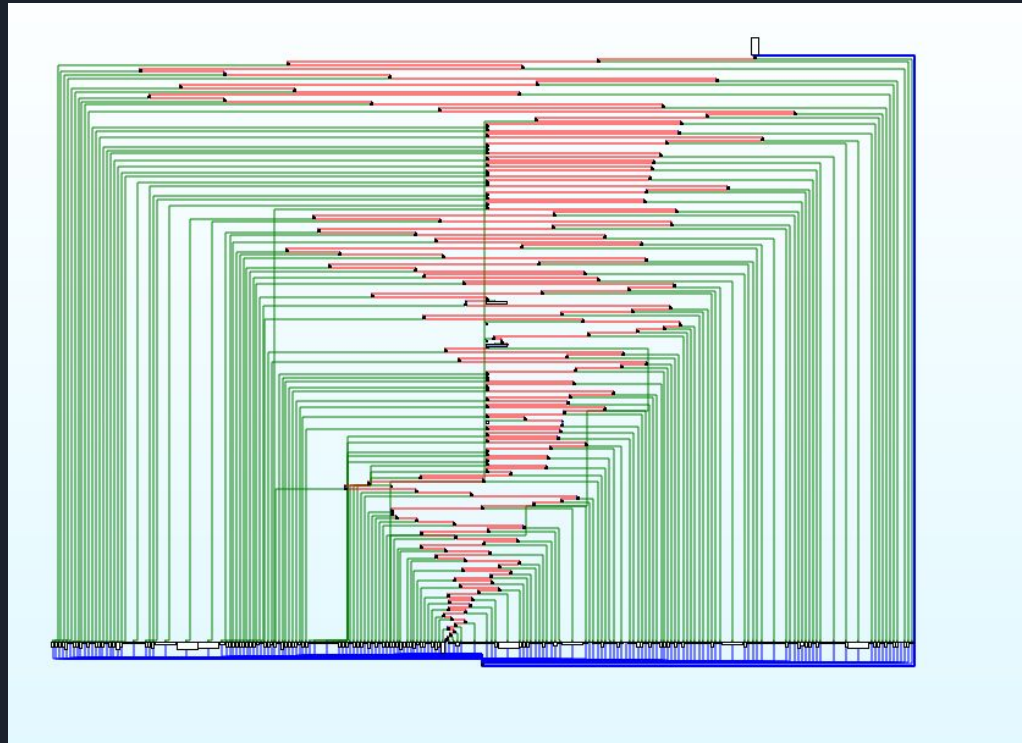

Demo

Reverse engineering = F5 ?

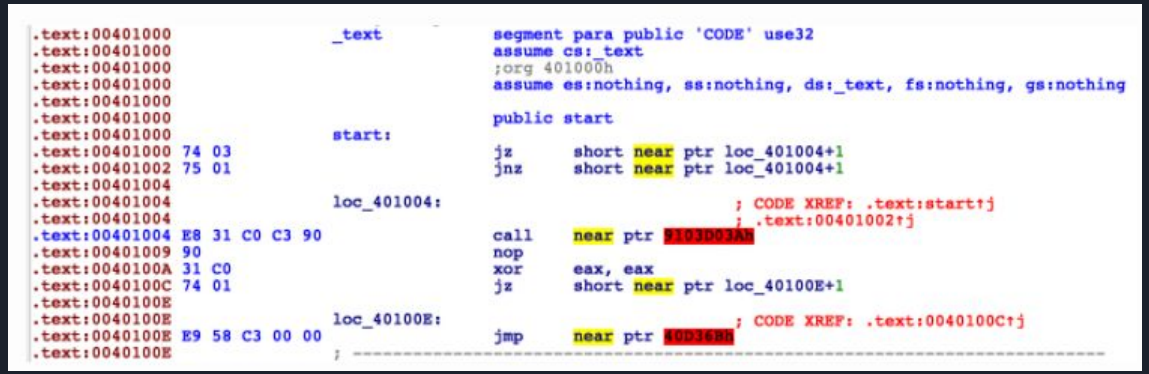
Anti-Reverse engineering – Obfuscate



obfuscate

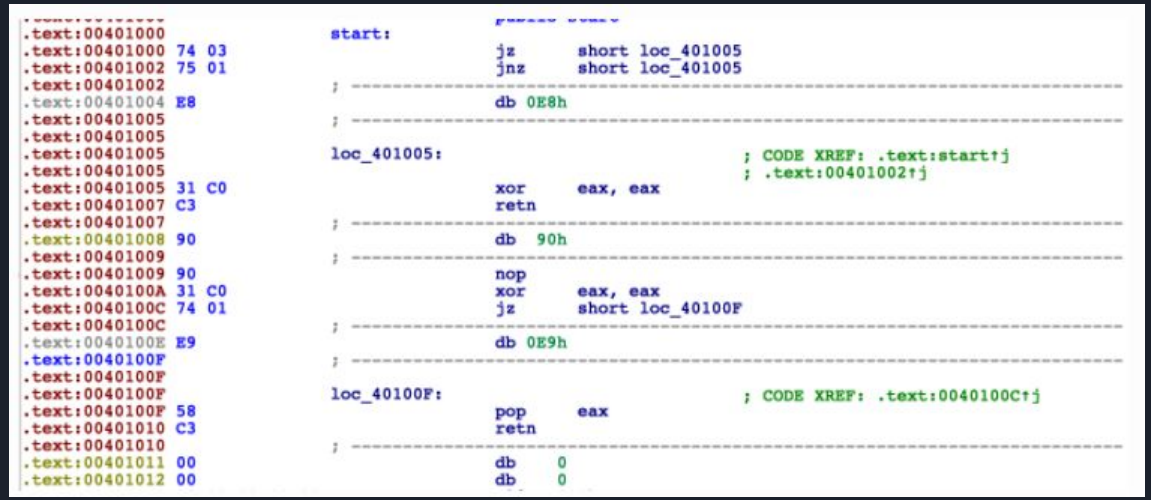


Anti-Reverse engineering – Indirect Jump



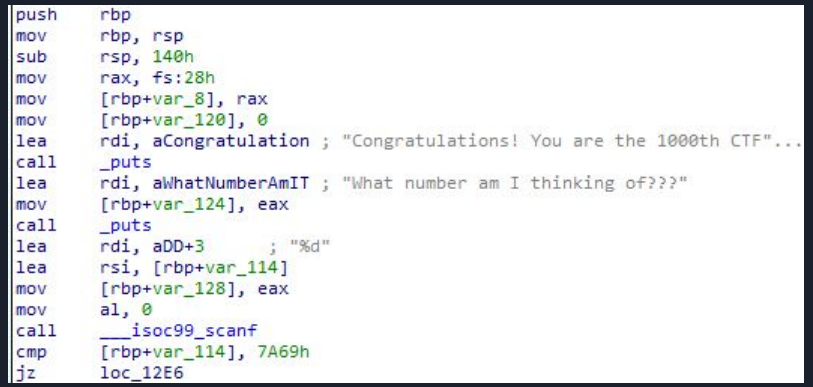
```
.text:00401000                  _text           segment para public 'CODE' use32
.text:00401000                                  assume cs:_text
.text:00401000                                  ;org 401000h
.text:00401000                                  assume es:nothing, ss:nothing, ds:_text, fs:nothing, gs:nothing
.text:00401000
.text:00401000                                  public start
.text:00401000                  start:
.text:00401000 74 03                            jz      short near ptr loc_401004+1
.text:00401002 75 01                            jnz     short near ptr loc_401004+1
.text:00401004
.text:00401004                  loc_401004:                             ; CODE XREF: .text:start↑j
.text:00401004                                                          ; .text:00401002↑j
.text:00401004 E8 31 C0 C3 90                   call    near ptr 9103D03ABh
.text:00401009 90                               nop
.text:0040100A 31 C0                            xor     eax, eax
.text:0040100C 74 01                            jz      short near ptr loc_40100E+1
.text:0040100E
.text:0040100E                  loc_40100E:                             ; CODE XREF: .text:0040100C↑j
.text:0040100E E9 58 C3 00 00                   jmp     near ptr 40D36Bh
.text:0040100E                  ; ---------------------------------------------------------------------------
```

Anti-Reverse engineering – embedded data



```
.text:00401000                  start:
.text:00401000 74 03                    jz      short loc_401005
.text:00401002 75 01                    jnz     short loc_401005
.text:00401002                  ; ---------------------------------------------------------------------------
.text:00401004 E8                       db 0E8h
.text:00401005                  ; ---------------------------------------------------------------------------
.text:00401005
.text:00401005                  loc_401005:                             ; CODE XREF: .text:start↑j
.text:00401005                                                          ; .text:00401002↑j
.text:00401005 31 C0                    xor     eax, eax
.text:00401007 C3                       retn
.text:00401007                  ; ---------------------------------------------------------------------------
.text:00401008 90                       db  90h
.text:00401009                  ; ---------------------------------------------------------------------------
.text:00401009 90                       nop
.text:0040100A 31 C0                    xor     eax, eax
.text:0040100C 74 01                    jz      short loc_40100F
.text:0040100C                  ; ---------------------------------------------------------------------------
.text:0040100E E9                       db 0E9h
.text:0040100F                  ; ---------------------------------------------------------------------------
.text:0040100F
.text:0040100F                  loc_40100F:                             ; CODE XREF: .text:0040100C↑j
.text:0040100F 58                       pop     eax
.text:00401010 C3                       retn
.text:00401010                  ; ---------------------------------------------------------------------------
.text:00401011 00                       db    0
.text:00401012 00                       db    0
```

Assembly (ASM)



```
push    rbp
mov     rbp, rsp
sub     rsp, 140h
mov     rax, fs:28h
mov     [rbp+var_8], rax
mov     [rbp+var_120], 0
lea     rdi, aCongratulation ; "Congratulations! You are the 1000th CTF"...
call    _puts
lea     rdi, aWhatNumberAmIT ; "What number am I thinking of???"
mov     [rbp+var_124], eax
call    _puts
lea     rdi, aDD+3      ; "%d"
lea     rsi, [rbp+var_114]
mov     [rbp+var_128], eax
mov     al, 0
call    ___isoc99_scanf
cmp     [rbp+var_114], 7A69h
jz      loc_12E6
```

Leaked Secret
forensics, ★☆☆☆☆

Scratch Passcode
misc, ★☆☆☆☆

Enormous Little Blog II
crypto, ★★★☆☆

CrackMe Revenge
reverse, ★★☆☆☆

CTF Training Questionnaire
web, ★☆☆☆☆

Admin panel
web, ★☆☆☆☆

Shellcode Runner
pwn, ★☆☆☆☆

Post-it
reverse, ★★☆☆☆

To Modify the Past
pwn, ★☆☆☆☆

Infant Browser
web, pwn, ★☆☆☆☆

Scattered
reverse, ★★☆☆☆

Sanity Check 2
web, ★★☆☆☆

All Missing
pwn, ★★☆☆☆

JPG as key
misc, ★★☆☆☆

DNS Record
misc, ★☆☆☆☆

Jack Botnet Service 1
forensics, ★★☆☆☆

Long Story Short
crypto, ★★☆☆☆

Freedom
crypto, ★★☆☆☆

Shuffle

Info 基本資料

harrier | reverse | ★☆☆☆☆

Description 描述

I heard perfect shuffle is reproducible...

Attachment: shuffle_0f1b57766b8350360719184ccdf870d2.zip

Hint:

- What is `.pyc`? Are there some tools for reverting pyc to some readable source (maybe back to python script)?
- Next you have to revert the algorithm for flag, i.e. given the output, find the corresponding input (which is flag)
- Understanding random module should help a lot...
- Why do this always produces same result (for same input) but not randomly differ each time? Can you make use of this to revert back to flag?

hkcert2x{...}

Magic Number

https://en.wikipedia.org/wiki/List_of_file_signatures

7F 45 4C 46	DELELF	0		Executable and Linkable Format

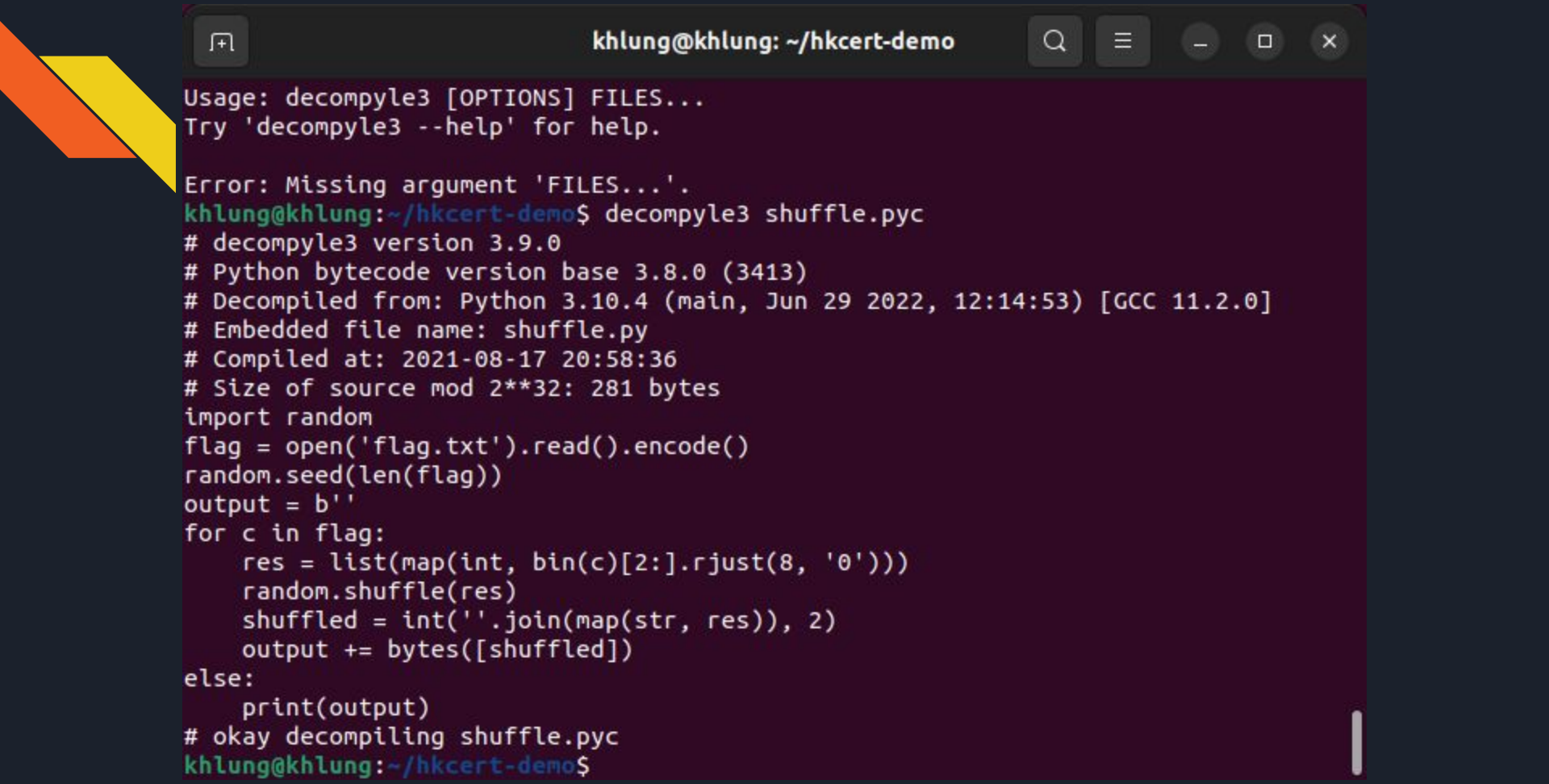
FF D8 FF DB	ÿØÿÛ	0	jpg jpeg	JPEG raw or in the JFIF or Exif file format[14]
FF D8 FF E0 00 10 4A 46 49 46 00 01	ÿØÿàNULDLEJFIFNULSOH			
FF D8 FF EE	ÿØÿî			
FF D8 FF E1 ?? ?? 45 78 69 66 00 00	ÿØÿá??ExifNULNUL			
FF D8 FF E0	ÿØÿà	0	jpg	JPEG raw or in the JFIF or Exif file format[14]

file

```
dileep@dileep-Inspiron-5558:~$ file *
array.class:      compiled Java class data, version 55.0
array.java:       C source, ASCII text
compile commands: ASCII text
Desktop:          directory
dileep:           directory
Documents:        directory
Download:         directory
Downloads:        directory
email.py:         ASCII text
exam.ods:         OpenDocument Spreadsheet
geeks file:       UTF-8 Unicode text
Invoice.pdf:      PDF document, version 1.4
java:             directory
javaprograms:     directory
Music:            directory
name.jpeg:        JPEG image data, JFIF standard 1.01, aspect ratio, density 1x1, segment length 16, progressive, precision 8, 1040x780, frame
s 3
oops:             directory
os.pdf:           PDF document, version 1.7
Pictures:         directory
Public:           directory
Templates:        directory
Videos:           directory
videosong.mp4:    ISO Media, MP4 v2 [ISO 14496-14]
workspace:        directory
wrapper.java:     C source, ASCII text
dileep@dileep-Inspiron-5558:~$
```

Other decompiler

- .apk (Android application package)
  - Online Decompiler (May not useful)
  - APKTools
    - https://medium.com/dwarsoft/how-to-reverse-engineer-an-android-application-in-3-easy-steps-dwarsoft-mobile-880d268bdc90
    - https://ibotpeaches.github.io/Apktool/documentation/
- .pyc (Python Compiled Format)
  - uncompyle6
  - decompyle3 - better for python 3.7 and higher
    - https://github.com/rocky/python-decompile3


khlung@khlung: ~/hkcert-demo
Usage: decompyle3 [OPTIONS] FILES...
Try 'decompyle3 --help' for help.
Error: Missing argument 'FILES...'.
khlung@khlung:~/hkcert-demo$ decompyle3 shuffle.pyc
# decompyle3 version 3.9.0
# Python bytecode version base 3.8.0 (3413)
# Decompiled from: Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0]
# Embedded file name: shuffle.py
# Compiled at: 2021-08-17 20:58:36
# Size of source mod 2**32: 281 bytes
import random
flag = open('flag.txt').read().encode()
random.seed(len(flag))
output = b''
for c in flag:
res = list(map(int, bin(c)[2:].rjust(8, '0')))
random.shuffle(res)
shuffled = int(''.join(map(str, res)), 2)
output += bytes([shuffled])
else:
print(output)
# okay decompiling shuffle.pyc
khlung@khlung:~/hkcert-demo$


Demo

Break

https://www.online-stopwatch.com/countdown/

Register now!

https://ctf.hkcert.org/

Sample Challenges:
https://training.hkcert22.pwnable.hk/

Discord channel:
https://discord.gg/V6QGvWCmDm

PWN

- Pwn, 是一個駭客語法的俚語詞, 自 "own"這個字引申出來的, 這個詞的含意在於, 玩家在整個遊戲對戰中處在勝利的優勢, 或是 說明競爭對手處在完全慘敗的情形下, 這個詞習慣上在網路遊戲文化主要用於嘲笑競爭對手在整個遊戲對戰中已經完全被擊敗。過去式的拼寫可以拼成pwnd, pwn3d, pwnt或是powned。維基百科

Last Year

- https://ctf.hkcert.org/wp-content/uploads/HKCERT_CTF_2021_Webinar_02.pdf p.41

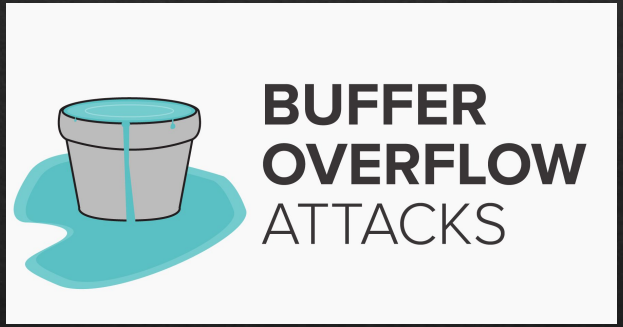
Buffer Overflow (BOF)

an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.

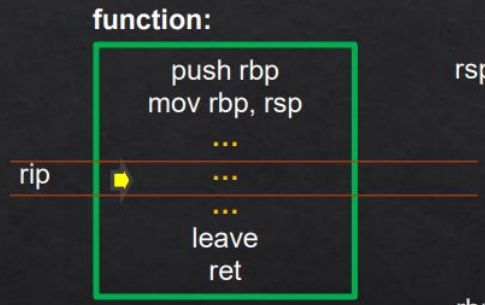
source: Wikipédia

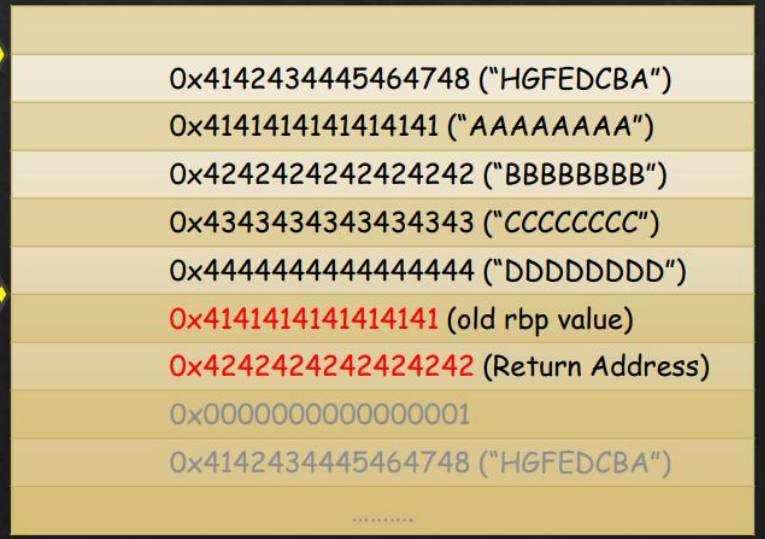
標對程式設計缺陷，向程式輸入緩衝區寫入使之溢位的內容（通常是超過緩衝區能儲存的最大數據量的數據），從而破壞程式執行、趁著中斷之際並取得程式乃至系統的控制權

來源：維基百科



BOF Attack





User input 40+16 characters:
"HGFEDCBAAAAAAAAABBBBBBBBCCC
CCCCCDDDDDDDD" +
"AAAAAAAABBBBBBBB"

Solving Pwn

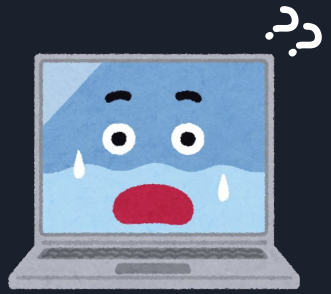
4. Achieve the target (difficulty: ★ ★ ☆ ☆ ☆)
   - Get the shell
     - system(“/bin/sh”)
     - execve(“/bin/sh”, [“/bin/sh”, null], null)
   - Print out the flag
     1. fd = open(“flag.txt”)
     2. read(fd, buf, 100)
     3. write(1, buf, 100)
   - Run the backdoor function
   - one gadget rce (won’t mention here)

```
5   /* Call this function! */
6   void win(void) {
7     char *args[] = {"/bin/sh", NULL};
8     execve(args[0], args, NULL);
9     exit(0);
10  }
```

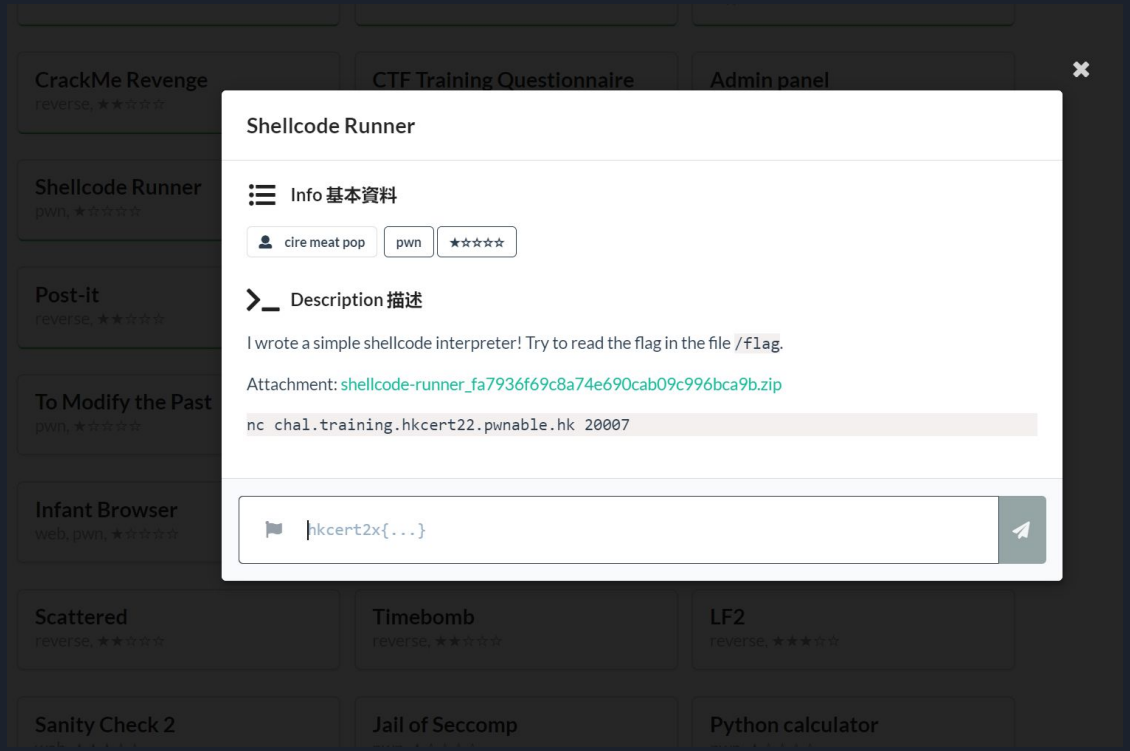
Write a backdoor function

```
void win(void) {
  char *args[] = {"/bin/sh", NULL};
  execve(args[0], args, NULL);
  exit(0);
}
```

```
48 89 C7          mov     rdi, rax        ; stream
E8 8B FE FF FF    call    _fflush
48 8B 45 E8       mov     rax, [rbp+seconds]
89 C7             mov     edi, eax        ; seconds
E8 90 FE FF FF    call    _sleep
48 D1 65 E8       shl     [rbp+seconds], 1
83 45 E4 01       add     [rbp+var_1C], 1
```



Shellcode


CrackMe Revenge
reverse, ★★☆☆☆
CTF Training Questionnaire
Admin panel
Shellcode Runner
pwn, ★☆☆☆☆
Post-it
reverse, ★★☆☆☆
To Modify the Past
pwn, ★☆☆☆☆
Infant Browser
web, pwn, ★☆☆☆☆
Scattered
reverse, ★★☆☆☆
Timebomb
reverse, ★★☆☆☆
LF2
reverse, ★★★☆☆
Sanity Check 2
Jail of Seccomp
Python calculator
Shellcode Runner
Info 基本資料
cire meat pop
pwn
★☆☆☆☆
Description 描述
I wrote a simple shellcode interpreter! Try to read the flag in the file /flag.
Attachment: shellcode-runner_fa7936f69c8a74e690cab09c996bca9b.zip
nc chal.training.hkcert22.pwnable.hk 20007
hkcert2x{...}

Write a backdoor function

```
48 89 C7
E8 8B FE FF FF
48 8B 45 E8
89 C7
E8 90 FE FF FF
48 D1 65 E8
83 45 E4 01
```

=

```
mov     rdi, rax        ; stream
call    _fflush
mov     rax, [rbp+seconds]
mov     edi, eax        ; seconds
call    _sleep
shl     [rbp+seconds], 1
add     [rbp+var_1C], 1
```

1. Write asm codes
2. Convert to shellcode
3. Input shellcode to the program
4. Change RIP pointing to shellcode
5. Get flag !!

Syscall

- execve(“/bin/sh”, [“/bin/sh”, null], null)
- https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86_64-64_bit

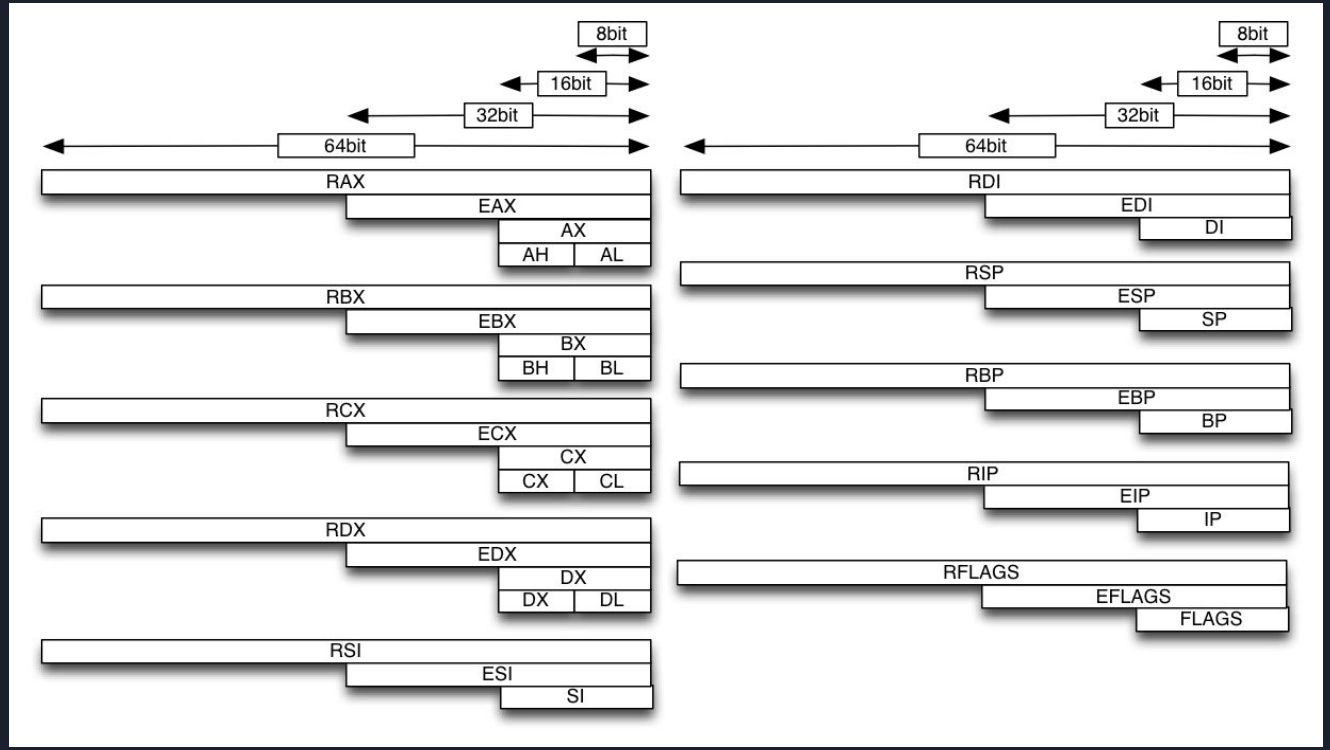
NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
0	read	man/ cs/	0x00	unsigned int fd	char *buf	size_t count	-	-	-
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count	-	-	-
					...				
59	execve	man/ cs/	0x3b	const char *filename	const char *const *argv	const char *const *envp	-	-	-

Syscall

execve(“/bin/sh”, [“/bin/sh”, null], null)

- rax: 0x3b
- rdi: address pointing to “/bin/sh”
- rsi: 0x0 or address pointing to [“/bin/sh”, null]
- rdx: 0x0
- syscall <= asm instruction

Register


8bit
16bit
32bit
64bit
RAX
EAX
AX
AH
AL
RBX
EBX
BX
BH
BL
RCX
ECX
CX
CX
CL
RDX
EDX
DX
DX
DL
RSI
ESI
SI
8bit
16bit
32bit
64bit
RDI
EDI
DI
RSP
ESP
SP
RBP
EBP
BP
RIP
EIP
IP
RFLAGS
EFLAGS
FLAGS

Basic Instrustion

- mov
  - mov rax, 123
- add/sub
  - add rax, rbx
- and/or/xor
  - and rax, 3
- push/pop
  - push rax
  - pop rbx
- jmp/call
  - call 0x400123
- leave/ret
- syscall

Mov

mov A, B = move B's value to A

rax = 0 , rbx = 1

- mov rax, rbx => result : rax = 1, rbx =1
- mov rax, 0x12345678 => result : rax = 0x12345678

A and B must be same in size

X mov rax, ebx

add/sub/and/or/xor

add A, B => A = A + B

rax = 3, rbx = 2

- add rax, 1 => result : rax = 4
- add rax, rbx => result : rax = 5

A and B must be same in size

memory operands []

Basically means dereference

rax = 0xdead1230

- mov rbx, QWORD PTR [rax] => result : rbx = 0x4

arithmetics can be handled inside emory operands

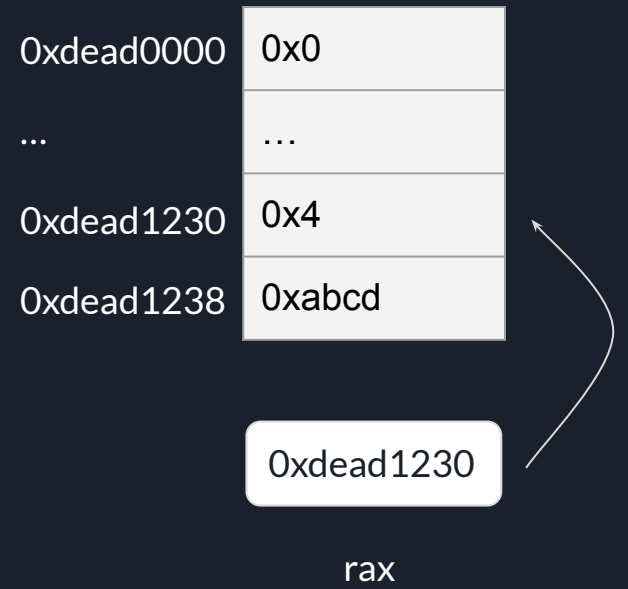
- mov rbx, QWORD PTR [rax+8]=> result : rbx = 0xabcd

QWORD -> 8 bytes

DWORD -> 4 bytes

WORD -> 2 bytes

BYTE-> 1 byte



push/pop

push/pop A

push rax

- (equivalent to) sub rsp, 8 ; mov [rsp], rax

pop rbx

- (equivalent to) mov rbx, [rsp] ; add rsp, 8

Example

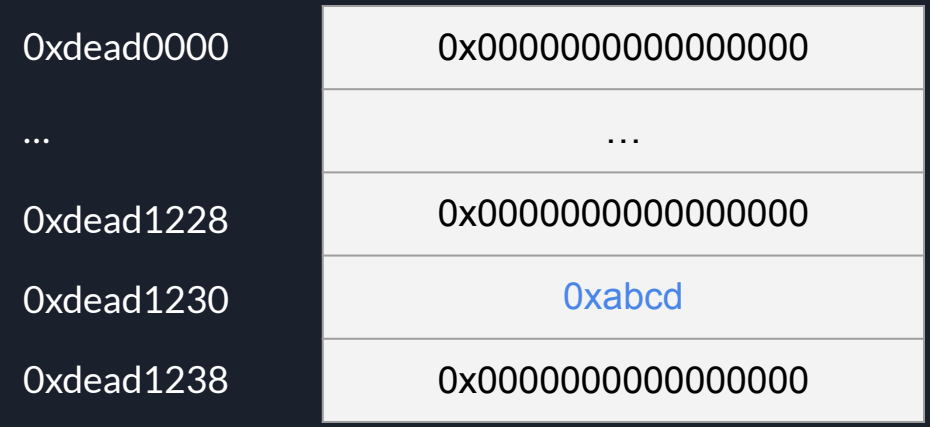
```
push rax
push 0xdead0000
pop rcx
mov QWORD PTR [rcx], rbx
```

0xdead0000	0x0000000000000000
…	…
0xdead1228	0x0000000000000000
0xdead1230	0x0000000000000000
0xdead1238	0x0000000000000000

rax	rbx	rcx	rsp	rbp
0xabcd	0x55	0x123	0xdead1238	0xdead1238

Example

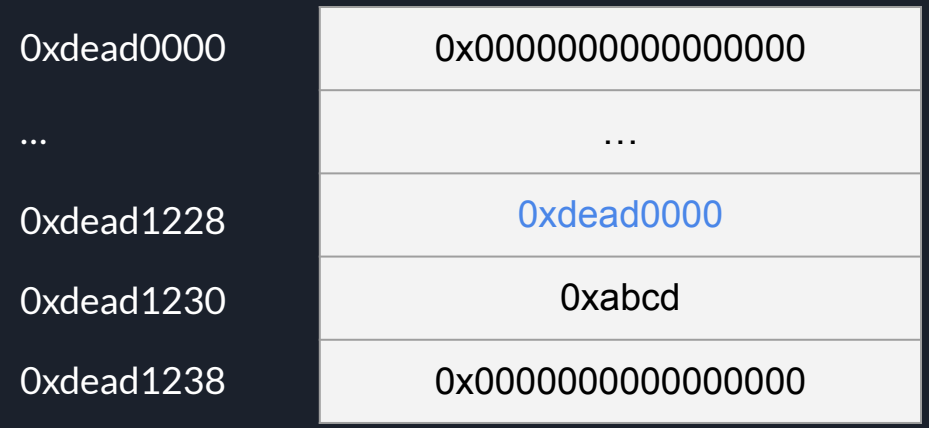
```
push rax
push 0xdead0000
pop rcx
mov QWORD PTR [rcx], rbx
```

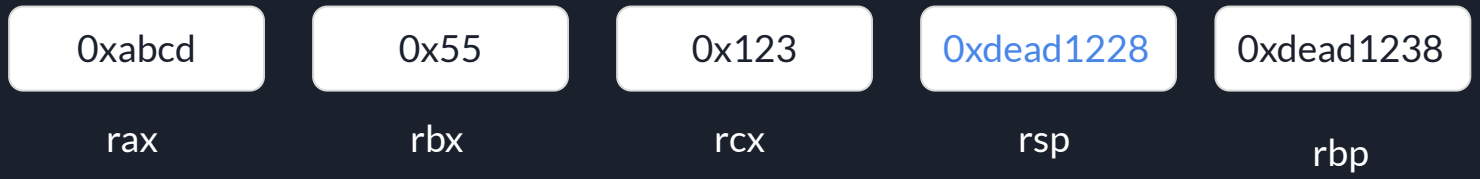


0xabcd	0x55	0x123	0xdead1230	0xdead1238
rax	rbx	rcx	rsp	rbp

Example

```
push rax
push 0xdead0000
pop rcx
mov QWORD PTR [rcx], rbx
```





Example

```
push rax
push 0xdead0000
pop rcx
mov QWORD PTR [rcx], rbx
```

0xdead0000	0x0000000000000000
…	…
0xdead1228	0xdead0000
0xdead1230	0xabcd
0xdead1238	0x0000000000000000

rax	rbx	rcx	rsp	rbp
0xabcd	0x55	0xdead0000	0xdead1230	0xdead1238

Example

```
push rax
push 0xdead0000
pop rcx
mov QWORD PTR [rcx], rbx
```

0xdead0000	0x55
…	…
0xdead1228	0xdead0000
0xdead1230	0xabcd
0xdead1238	0x0000000000000000

rax	rbx	rcx	rsp	rbp
0xabcd	0x55	0xdead0000	0xdead1230	0xdead1238

jmp/call/ret

jmp/call/ret <address>

jmp 0x400412

- (equivalent to) rip = 0x400412

call <function address>

- push <next instruction address> #store the return address
- jmp <function address>

ret

- (equivalent to) pop rip
- call when exit function

Syscall

execve(“/bin/sh”, [“/bin/sh”, null], null)

- rax: 0x3b
- rdi: address pointing to “/bin/sh”
- rsi: 0x0 or address pointing to [“/bin/sh”, null]
- rdx: 0x0
- syscall <= asm instruction

Pwntools

```
from pwn import *


assembly = '''
   mov rax, 0x3b
   xor rsi, rsi
   ....
   syscall
'''


shellcode = asm(assembly, arch='amd64')
```

1. Write asm codes
2. Convert to shellcode
3. Input shellcode to the program
4. Change RIP pointing to shellcode
5. Get flag !!


Demo

DEP(NX)

- Data Execution Prevention
- Checksec

Shellcode Challenge

- Seccomp bypass
  - Seccomp tools: https://github.com/david942j/seccomp-tools
  - seccomp-tools dump <program name>

```
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x09 0x40000003  if (A != ARCH_I386) goto 0011
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x15 0x07 0x00 0x000000ad  if (A == rt_sigreturn) goto 0011
 0004: 0x15 0x06 0x00 0x00000077  if (A == sigreturn) goto 0011
 0005: 0x15 0x05 0x00 0x000000fc  if (A == exit_group) goto 0011
 0006: 0x15 0x04 0x00 0x00000001  if (A == exit) goto 0011
 0007: 0x15 0x03 0x00 0x00000005  if (A == open) goto 0011
 0008: 0x15 0x02 0x00 0x00000003  if (A == read) goto 0011
 0009: 0x15 0x01 0x00 0x00000004  if (A == write) goto 0011
 0010: 0x06 0x00 0x00 0x00050026  return ERRNO(38)
 0011: 0x06 0x00 0x00 0x7fff0000  return ALLOW
```


- Shellcode restriction
  - Prime number only
    - primepwn (34C3 CTF)
  - Even number only
  - Odd number only
  - Alphabet letters + digits only

Q&A

Pwner Tools

- gef: more powerful gdb (debugger)

- ida/ghidra: decompile binary to pseudo code

- Pwntools: useful python module for ctf

Tips

- Each challenge cost time on learning/solving
- Google
- Goal: solve 1 challenge that is new to you

Training Platform

https://training.hkcert22.pwnable.hk/