6. Software Security

Jin Hong

jin.hong@uwa.edu.au

What you need today

- Kali VM
  - Install gdb-peda as per lab 3 instructions
- Windows VM
  - Download and install the OllyDBG debugger
    - https://www.ollydbg.de/download.htm
- Things we do today
  - Reverse engineering:
    - Codes
    - x86 programs

Reverse Engineering

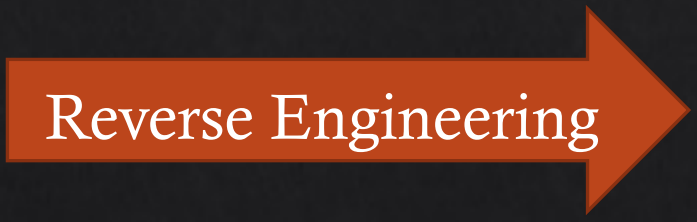
- What is Reversing?

- What is Reverse Engineering?

Program Abstractions

1. Computers understand binary code
2. Binary code can be written in hexadecimal
3. Hexadecimal code can be encoded in assembly language
4. Assembly language is human-readable but not as intuitive as source code
5. Decompilers convert assembly into an easier-to-read source code

11001111 10101 == CD21 == int 21



Security-related Reversing

- Malicious software
  - Malicious software (e.g., viruses and worms) spread much faster now that computers are connected to the Internet.
  - The infection process used to be slow (e.g., viruses spread by diskette sharing).
  - Today's worms can spread automatically to computers without any human intervention.
  - Developers of malware use RE to find vulnerabilities in OS and application software.
  - Developers of ant-virus software use RE to develop tools to protect users from malware.

- Reversing cryptographic algorithms
  - Restricted algorithms, where the encryption is in the algorithm, (e.g.,Caesar cipher) can be broken easily using RE.
  - Key-based algorithms, where the algorithms are public but the key is secret, are more difficult to break.
    - Vulnerabilities include obtaining the key (need luck)
    - trying all possible combinations until you get the key (need quantum computer)
    - Look for a flaw in the algorithm that exposes the key of the original message (need RE)

Security-related Reversing

- Digital Right Management (DMR)
  - Media content providers have developed or use technologies (DRM) that control the distribution of digital content (e.g.,music, movies).
  - DRM technologies determine if the content should be made available or not.
  - Crackers use RE to attempt to defeat DRM technologies. RE can help to:
    - reveal the secrets of DRM technology;
    - discover simple modifications that can be made to DRM technologies to disable the protection they offer.
- Auditing program binaries
  - Open source software feels safer to run because it has been inspected and approved by thousands of impartial software engineers.
  - RE is a viable (albeit limited) alternative for searching for security vulnerabilities when the source code of the program is not available.

Software Development-Reversing

- Achieving Interoperability with proprietary software
  - Proprietary software libraries or OS APIs often have insufficient documentation.
  - Solutions:
    - Try to get things to work via brute force.
    - Contact vendor for answers.
    - Use RE to figure out how the code works.
- Developing competing software
  - Even if the competitor has unpatented technology it is not cost-effective to RE the entire product.
  - The exception may be highly complex or unique designs/algorithms that are very difficult or costly to develop.
  - There are legal issues to consider when RE competing software.
- Evaluating software quality and robustness
  - It is possible to audit software to get an estimate of the general quality of the coding practices used in the program.
  - Microsoft's Windows source code is not publicly available, and the source code is released to only a few for evaluation purposes.
  - The rest have to either trust Microsoft or resort to RE.
  - Access to source code is more desirable that binary reversing, however – why?

RE exercise

- General purpose registers:
  - `EAX`, `EBX`, `ECX`, `EDX`, `ESI`, `EDI`, `ESP`, `EBP`
  - `ESP` usually for stack pointer
  - `EBP` usually for base pointer
- `mov` – moves data between registers and memory from source to destination
- `mov` has variants – `movb` (1 byte), `movw` (2 bytes), `movl` (3 bytes)
- `$1` is the value 1
  - E.g., `movl` $1, `%edx` means move 32-bit integer value 1 to the address in `EDX`
- `jmp` – go to the label specified.
- `jne` – go to the label if not equal.
- `imul` – multiply two, and save onto the second operand.
- `decl` – decrement by 1
- `Ret` - return

Note: this is based on a specific OS on X86 arch. The assembly code will differ based on the OS, CPU arch and the compiler used.

```
1       .file   "factorial.c"
2       .text
3   .globl factorial
4       .type   factorial, @function
5   factorial:
6       testl   %eax, %eax
7       jne     .L2
8       movl    $1, %edx
9       jmp     .L4
10  .L2:
11      movl    $1, %edx
12  .L5:
13      imull   %eax, %edx
14      decl    %eax
15      jne     .L5
16  .L4:
17      movl    %edx, %eax
18      ret
19      .size   factorial, .-factorial
20      .ident  "GCC: (GNU) 4.1.2 20060715 (prerelease) (Debian 4.1.1-9)"
21      .section    .note.GNU-stack,"",@progbits
```

RE exercise

```
factorial:
    if ((eax & eax) != 0){
        go to .L2;
    }
    else{
        edx = 1;
        go to .L4;
    }
.L2:
    edx = 1;
.L5:
    edx = eax * edx;
    eax = eax – 1;
    if(eax != 0){
        go to .L5;
    }
.L4:
    eax = edx;
    return;
```

```
1       .file   "factorial.c"
2       .text
3   .globl factorial
4       .type   factorial, @function
5   factorial:
6       testl   %eax, %eax
7       jne     .L2
8       movl    $1, %edx
9       jmp     .L4
10  .L2:
11      movl    $1, %edx
12  .L5:
13      imull   %eax, %edx
14      decl    %eax
15      jne     .L5
16  .L4:
17      movl    %edx, %eax
18      ret
19      .size   factorial, .-factorial
20      .ident   "GCC: (GNU) 4.1.2 20060715 (prerelease) (Debian 4.1.1-9)"
21      .section    .note.GNU-stack,"",@progbits
```

Reverse engineer this code!

Is RE legal?

- What social and economic impact does RE have on society?

Legal Issues:

Interoperability

- Exposing software interfaces facilitates the development of software that runs on top of the exposed platform.
  - Good for increasing the sales of the platform.
  - Bad if the competition offers a better product on the same platform.
- US court in legal case of Sega (original) versus Accolade (copied) ruled in favor of Accolade.
  - Accolade did not violate code copyright of Sega.
  - Competition in the market benefited customers.
  - Court essentially authorized RE for the purpose of interoperability.

Legal Issues:

Competition

- RE for interoperability has societal benefits, but when RE is used to develop competing products the situation is more complicated.
- Opponents of RE claim that reversing stifles innovation.
- Illegal (easy to prove): Directly stealing code.
- Illegal (hard to prove): Decompiling a program and recompiling it to generate a different binary with the same functionality.
- Legal: RE small parts of a product to gather information, not code. Then develop code independently.

Legal Issues:

Copyright Law

- A violation of copyright law is to directly copy protected code sequences from the competitor's product into you own product.
- Some have claimed that intermediate copies during RE decompilation violates copyright.
  - These claims did not hold in court (e.g., Sega versus Accolade) because intermediate copies are created when a program is installed from a CD onto a hard disk.
- If the final product does not contain anything that was directly copied from the original product, copying is considered fair use.

Legal Issues:

Trade Secrets and Patents

- **Patent**:
  - Owner controls the invention for up to 20 years (or so, depending on the jurisdiction).
  - Details of the patent are publicly disclosed, hence RE of a patented technology does not make sense.
  - Invention becomes public after patent expires.
  - Patent protects owner from a competitor who independently invents the same technology.
- **Trade secret**:
  - Granted automatically if product is kept a secret and significant effort was put into its development.
  - Protects against rogue employee selling secrets to the competition.
  - Does not protect product from RE.

Legal Issues:

The Digital Millennium Copyright Act (DMCA)

- DMCA legally protects *copyright protection systems* from circumvention.
- DMCA is considered *anti-RE legislation* because such circumvention almost always involves reversing.
- DMCA only applies to copyright protection systems, so most RE applications are not affected by it.
- Two high-profile cases of DMCA violation:
  - Felten versus RIAA (audio recordings protection)
  - US versus Skylarov (Adobe eBook file format)

Legal Issues:

The Digital Millennium Copyright Act (DMCA) – What is not allowed?

1. Circumvention of copyright protection systems.
   - E.g., using a *keygen* program.
2. The development of circumvention technologies (DRM Digital Rights Management).
   - E.g., developing a *keygen* program.

- A *keygen* program generates a serial number on-the-fly for programs that require a serial number during installation.

Legal Issues:

The Digital Millennium Copyright Act (DMCA) - What is allowed?

- Interoperability
- Encryption research on encryption products
- Security testing
- Educational institutions and public libraries can violate DMCA in order to evaluate a copyrighted work prior to purchasing it.
- Government investigation
- Regulation
- Protection of privacy

Legal Issues:

License Agreement Considerations

- Other than DMCA, there are no laws that directly prohibit or restrict RE.
- DMCA only applies to DRM products.
- Vendors add anti-RE clauses to software license agreements.
- In the US it is not clear that these clauses are enforceable.
- In the EU decompilation for the purposes of interoperability is permissible under the “Directive of the Legal Protection of Computer Programs”.

The RE Process

- System-level reversing:
  - Techniques that help determine the general structure of the program.
  - Most of the information comes from the OS and, hence, OS monitoring utilities.
- Code-level reversing:
  - Techniques provide detailed information on a selected code segment.
  - Involves extracting design concepts and algorithms from binary code.

RE Tools

- System monitoring tools:
  - Network activity, file access, register access.
- Disassemblers:
  - Translate binary code to assembly language code.
- Debuggers:
  - Reversers use debuggers in disassembly mode to set breakpoints and step through a program's execution (with on-the-fly disassembly).
- Decompilers:
  - Attempt to produce high-level code (e.g., C) from an executable binary file. **A reverse compiler!**
  - Perfect decompilation impossible for most platforms.
  - Decompile a binary programs into readable source code.
  - Replace all binary code that could not be decompiled with assembly code.

Reverse Engineering Tool Categories

- Hex Editors
- Decompilers
- **Disassemblers/Debuggers**

Reverse Engineering Prevention Tools

- Code Obfuscators

Disassemblers/Debuggers

- Convert binary code into its assembly equivalent.
- Extract ASCII strings and used libraries.
- View memory, stack and CPU registers.
- Run the program (with breakpoints).
- Edit the assembly code at runtime.

Product	Dis-Assembly	Processor options	Debugger	String extraction	Disk Hex editor	Memory Hex editor	Memory Dumper	Library's used	Decryptor
IDAPro	x	x		x	x			x	
OllyDbg	x	x	x	x	x	x	x	x	
W32Dasm	x	x	x	x	x	x	x	x	
BORG	x	x					x		x

GDB

- Demo
- You should be able to use your Kali

You need an AMD64 machine…

wget https://github.com/uwacyber/cits3006/raw/2023S2/cits3006-labs/files/test_gdb

Disassemblers/Debuggers: objdump

- objdump -d [file]

```
0000122d <validate_password>:
    122d:       f3 0f 1e fb             endbr32
    1231:       55                      push   %ebp
    1232:       89 e5                   mov    %esp,%ebp
    1234:       e8 f6 00 00 00          call   132f <__x86.get_pc_thunk.ax>
    1239:       05 93 2d 00 00          add    $0x2d93,%eax
    123e:       8b 45 08                mov    0x8(%ebp),%eax
    1241:       0f b6 00                movzbl (%eax),%eax
    1244:       3c 68                   cmp    $0x68,%al
    1246:       75 3b                   jne    1283 <validate_password+0x56>
    1248:       8b 45 08                mov    0x8(%ebp),%eax
    124b:       83 c0 01                add    $0x1,%eax
    124e:       0f b6 00                movzbl (%eax),%eax
    1251:       3c 65                   cmp    $0x65,%al
    1253:       75 2e                   jne    1283 <validate_password+0x56>
    1255:       8b 45 08                mov    0x8(%ebp),%eax
    1258:       83 c0 02                add    $0x2,%eax
    125b:       0f b6 00                movzbl (%eax),%eax
    125e:       3c 6c                   cmp    $0x6c,%al
    1260:       75 21                   jne    1283 <validate_password+0x56>
    1262:       8b 45 08                mov    0x8(%ebp),%eax
    1265:       83 c0 03                add    $0x3,%eax
    1268:       0f b6 00                movzbl (%eax),%eax
    126b:       3c 6c                   cmp    $0x6c,%al
    126d:       75 14                   jne    1283 <validate_password+0x56>
    126f:       8b 45 08                mov    0x8(%ebp),%eax
    1272:       83 c0 04                add    $0x4,%eax
    1275:       0f b6 00                movzbl (%eax),%eax
    1278:       3c 6f                   cmp    $0x6f,%al
    127a:       75 07                   jne    1283 <validate_password+0x56>
    127c:       b8 00 00 00 00          mov    $0x0,%eax
    1281:       eb 05                   jmp    1288 <validate_password+0x5b>
    1283:       b8 ff ff ff ff          mov    $0xffffffff,%eax
    1288:       5d                      pop    %ebp
    1289:       c3                      ret
```

Disassemblers/Debuggers: strings

- strings [file]

```
(base) jinhong@uwacyber-s1:~/test$ strings test_gdb
td0
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
__isoc99_scanf
puts
printf
strlen
__cxa_finalize
__libc_start_main
GLIBC_2.7
GLIBC_2.1.3
GLIBC_2.0
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
<hu;
<eu.
<lu!
[^_]
Enter password:
Success!
Failure!
9*2$"
GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.7623
__do_global_dtors_aux_fini_array_entry
frame_dummy
__frame_dummy_init_array_entry
test.c
__FRAME_END__
__init_array_end
_DYNAMIC
__init_array_start
__GNU_EH_FRAME_HDR
_GLOBAL_OFFSET_TABLE_
__libc_csu_fini
_ITM_deregisterTMCloneTable
__x86.get_pc_thunk.bx
validate_password
```

Disassemblers/Debuggers: GDB

- You will do more in the labs

```
(gdb) info func
All defined functions:

Non-debugging symbols:
0x00001000  _init
0x00001090  __cxa_finalize@plt
0x000010a0  printf@plt
0x000010b0  puts@plt
0x000010c0  strlen@plt
0x000010d0  __libc_start_main@plt
0x000010e0  __isoc99_scanf@plt
0x000010f0  _start
0x00001130  __x86.get_pc_thunk.bx
0x00001140  deregister_tm_clones
0x00001180  register_tm_clones
0x000011d0  __do_global_dtors_aux
0x00001220  frame_dummy
0x00001229  __x86.get_pc_thunk.dx
0x0000122d  validate_password
0x0000128a  main
0x0000132f  __x86.get_pc_thunk.ax
0x00001340  __libc_csu_init
0x000013b0  __libc_csu_fini
0x000013b5  __x86.get_pc_thunk.bp
```

Disassemblers/Debuggers: GDB

```
gdb-peda$ disas validate_password
Dump of assembler code for function validate_password:
   0x5655622d <+0>:     endbr32
   0x56556231 <+4>:     push   ebp
   0x56556232 <+5>:     mov    ebp,esp
   0x56556234 <+7>:     call   0x5655632f <__x86.get_pc_thunk.ax>
   0x56556239 <+12>:    add    eax,0x2d93
   0x5655623e <+17>:    mov    eax,DWORD PTR [ebp+0x8]
   0x56556241 <+20>:    movzx  eax,BYTE PTR [eax]
   0x56556244 <+23>:    cmp    al,0x68
   0x56556246 <+25>:    jne    0x56556283 <validate_password+86>
   0x56556248 <+27>:    mov    eax,DWORD PTR [ebp+0x8]
   0x5655624b <+30>:    add    eax,0x1
   0x5655624e <+33>:    movzx  eax,BYTE PTR [eax]
   0x56556251 <+36>:    cmp    al,0x65
   0x56556253 <+38>:    jne    0x56556283 <validate_password+86>
   0x56556255 <+40>:    mov    eax,DWORD PTR [ebp+0x8]
   0x56556258 <+43>:    add    eax,0x2
   0x5655625b <+46>:    movzx  eax,BYTE PTR [eax]
   0x5655625e <+49>:    cmp    al,0x6c
   0x56556260 <+51>:    jne    0x56556283 <validate_password+86>
   0x56556262 <+53>:    mov    eax,DWORD PTR [ebp+0x8]
   0x56556265 <+56>:    add    eax,0x3
   0x56556268 <+59>:    movzx  eax,BYTE PTR [eax]
   0x5655626b <+62>:    cmp    al,0x6c
   0x5655626d <+64>:    jne    0x56556283 <validate_password+86>
   0x5655626f <+66>:    mov    eax,DWORD PTR [ebp+0x8]
   0x56556272 <+69>:    add    eax,0x4
   0x56556275 <+72>:    movzx  eax,BYTE PTR [eax]
   0x56556278 <+75>:    cmp    al,0x6f
   0x5655627a <+77>:    jne    0x56556283 <validate_password+86>
   0x5655627c <+79>:    mov    eax,0x0
   0x56556281 <+84>:    jmp    0x56556288 <validate_password+91>
   0x56556283 <+86>:    mov    eax,0xffffffff
   0x56556288 <+91>:    pop    ebp
   0x56556289 <+92>:    ret
End of assembler dump.
gdb-peda$
```

Disassemblers/Debuggers: GDB

```
Breakpoint 7, 0x56556241 in validate_password ()
gdb-peda$ si
[----------------------------------registers-----------------------------------]
EAX: 0x72 ('r')
EBX: 0x56558fcc --> 0x3ed4
ECX: 0x30cb
EDX: 0x0
ESI: 0xf7f8a000 --> 0x1e7d6c
EDI: 0xf7f8a000 --> 0x1e7d6c
EBP: 0xffffcf58 --> 0xffffcf88 --> 0x0
ESP: 0xffffcf58 --> 0xffffcf88 --> 0x0
EIP: 0x56556244 (<validate_password+23>:        cmp    al,0x68)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x56556239 <validate_password+12>:   add    eax,0x2d93
   0x5655623e <validate_password+17>:   mov    eax,DWORD PTR [ebp+0x8]
   0x56556241 <validate_password+20>:   movzx  eax,BYTE PTR [eax]
=> 0x56556244 <validate_password+23>:   cmp    al,0x68
   0x56556246 <validate_password+25>:
    jne    0x56556283 <validate_password+86>
   0x56556248 <validate_password+27>:   mov    eax,DWORD PTR [ebp+0x8]
   0x5655624b <validate_password+30>:   add    eax,0x1
   0x5655624e <validate_password+33>:   movzx  eax,BYTE PTR [eax]
[-------------------------------------stack-------------------------------------]
0000| 0xffffcf58 --> 0xffffcf88 --> 0x0
0004| 0xffffcf5c --> 0x565562f3 (<main+105>:    add    esp,0x10)
0008| 0xffffcf60 --> 0xffffcf7b ("rewrw")
0012| 0xffffcf64 --> 0xffffcf7b ("rewrw")
0016| 0xffffcf68 --> 0x56558fcc --> 0x3ed4
0020| 0xffffcf6c --> 0x565562a5 (<main+27>:     add    ebx,0x2d27)
0024| 0xffffcf70 --> 0x2
0028| 0xffffcf74 --> 0xffffd034 --> 0xffffd21d ("/home/jinhong/test/test_gdb")
[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x56556244 in validate_password ()
gdb-peda$
```


echo -e "\x68"

OllyDbg

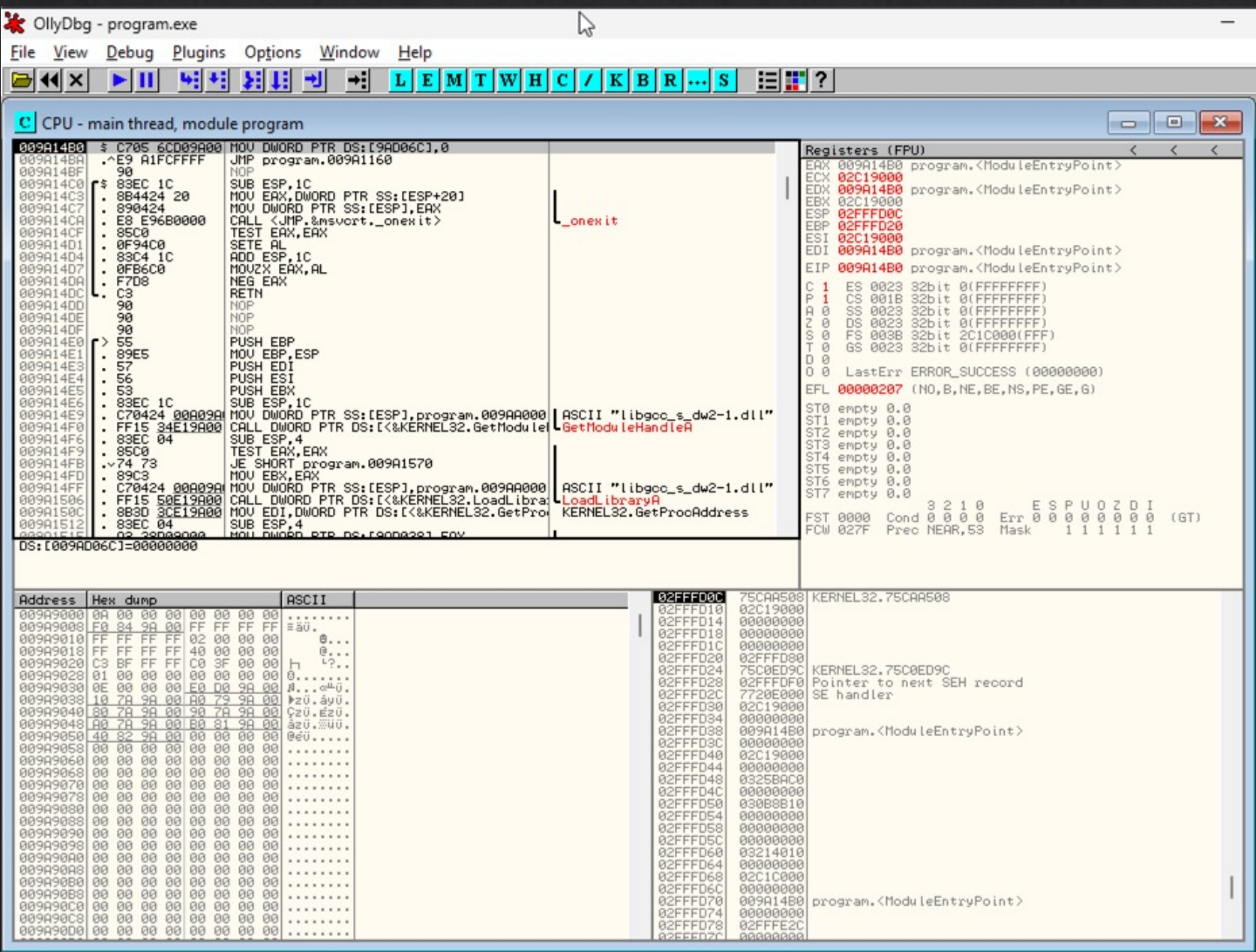
◈demo

- You need a Windows VM and install OllyDbg
  - tested on Windows 11 Preview VM
  - Run OllyDbg as administrator
  - If any prompts about DLL, don't delete any.

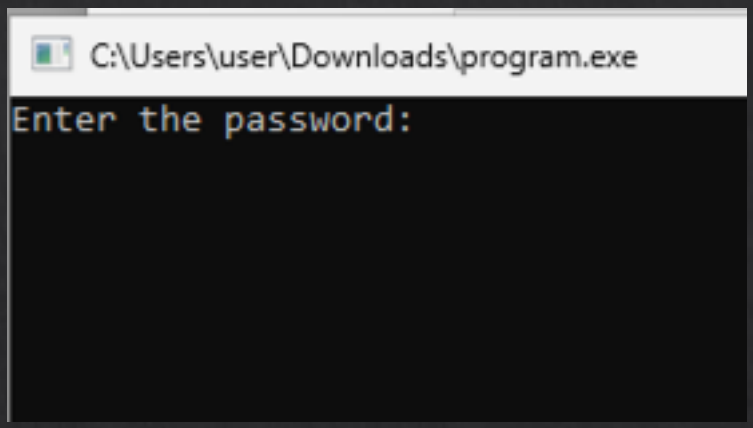
wget https://github.com/uwacyber/cits3006/raw/2023S2/cits3006-labs/files/program.exe -O program.exe

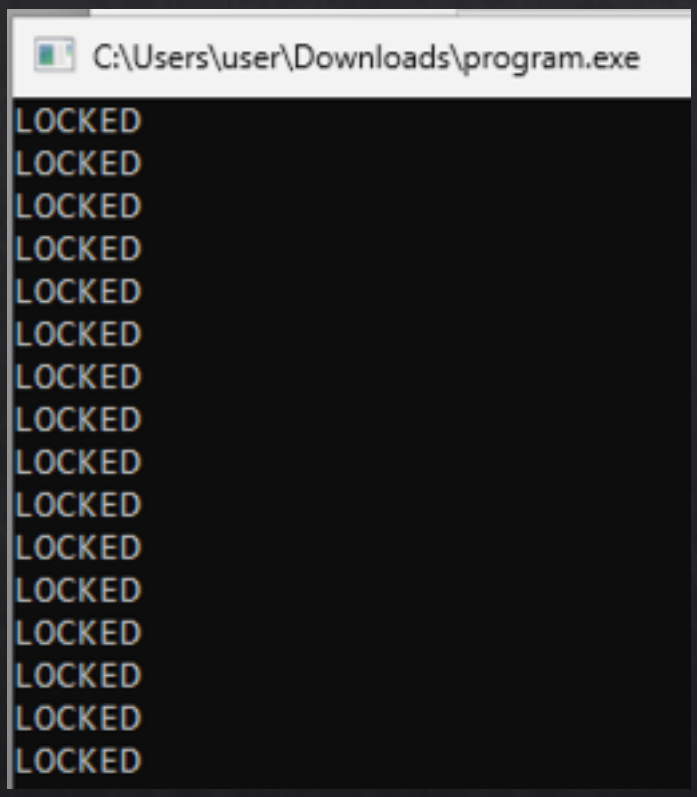
Disassemblers/Debuggers: OllyDbg

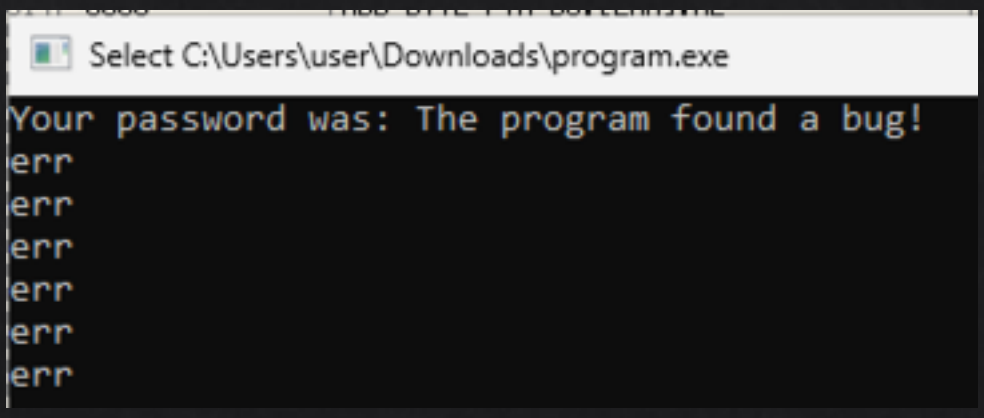
- Executes program in a controlled environment.
- Allows the flow of the program to be controlled.
- Uses a convenient layout showing hexadecimal, assembly, CPU registers and stack.
- Allows the program to be dumped from the memory onto the hard-disk.
- Highlights recently changed values in memory/stack/CPU registers.



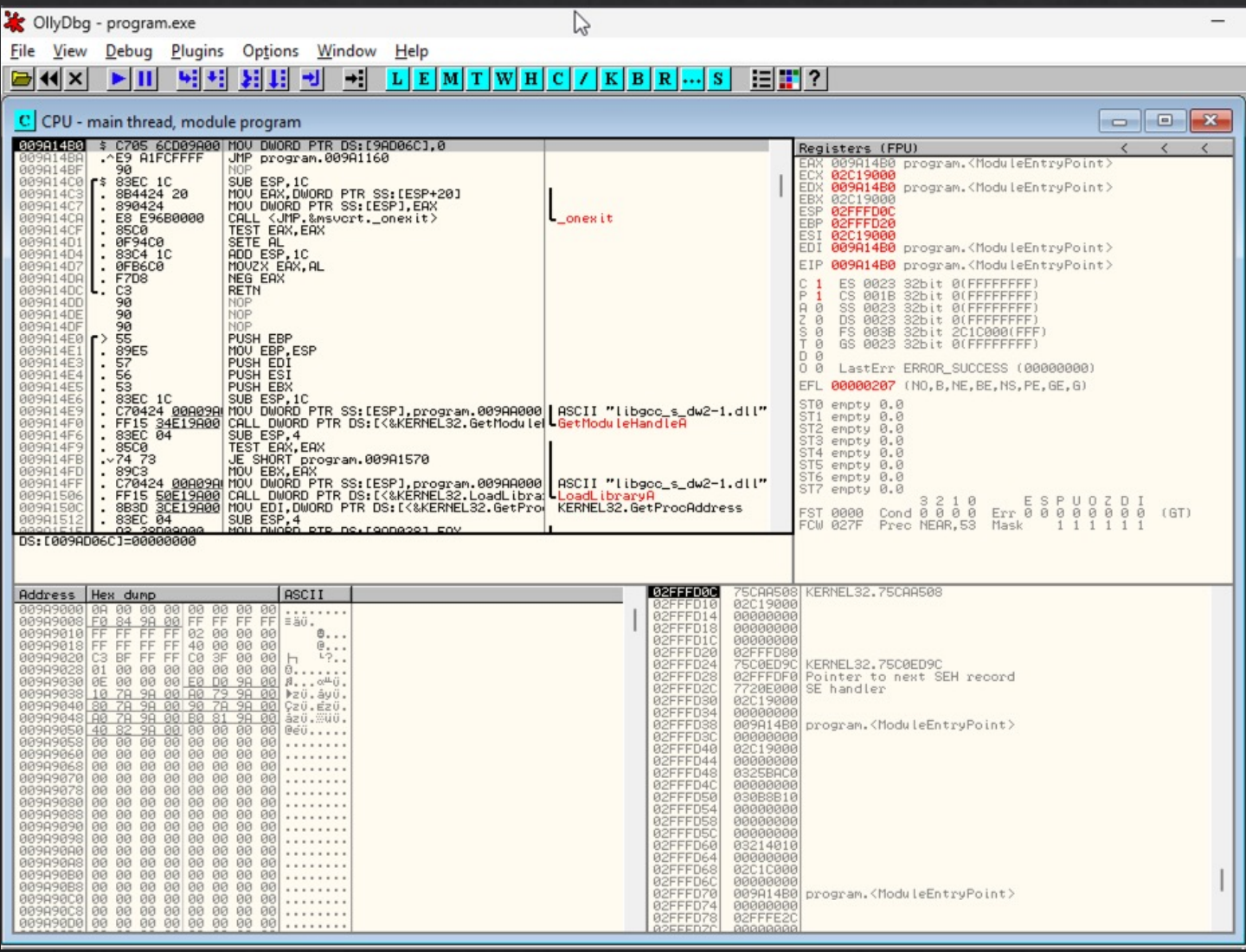
Disassemblers/Debuggers: OllyDbg







Disassemblers/Debuggers: OllyDbg



Reverse Engineering Tool Categories

- Hex Editors
- Decompilers
- Disassemblers/Debuggers

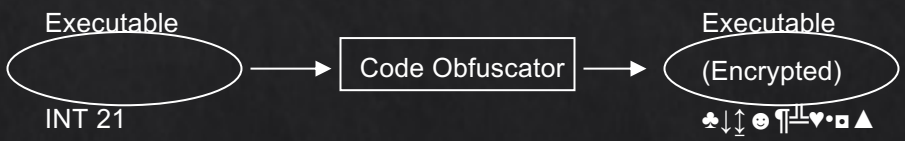
Reverse Engineering Prevention Tools

- **Code Obfuscators**

Code Obfuscators

- Encrypts the code of a program so you cannot view it in assembly.
- Disallows the program to be run if it detects a known disassembler or debugger running.

Obfuscators	Obfuscation	Anti-debugging techniques	GUI
Y0da's Cryptor	x	x	x
NFO	x	x	



Resources

- Reverse Engineering
  - https://0xinfection.github.io/reversing/
- Decompiler tools
  - https://en.kali.tools/all/?category=decompiler

References

- Materials adopted from
  - Spiros Mancoridis, Drexel University