9. Reverse Engineering

Jin Hong

jin.hong@uwa.edu.au

Preparation for the RE demo later

- If you want to follow along, setup your Windows VM if you haven't done already
- On your Windows VM:
  - Download the test program – program.exe
    - https://github.com/uwacyber/cits3006/raw/2022s2/cits3006-labs/files/program.exe
  - Download the OllyDBG debugger
    - https://www.ollydbg.de/download.htm
- You will be given a time later to try, so please pay attention to the lecture for now

What is Reverse Engineering?

- Reverse engineering (RE) is the process of extracting the knowledge or design blueprints from anything man made.
- The difference between RE and scientific research is that with RE the artifact being investigated is man made.
- Goal of RE is to obtain missing knowledge, ideas, and design philosophy when such information is unavailable.

Software Reverse Engineering: Reversing

- Reversing is about dissecting a program and examining its internals.
- In most industries RE is used for developing competing products, but this is not the case for the software industry.
  - RE of software is thought to be too complex to make sense financially.
- Common applications of RE in the software industry:
  - Security
  - Software development

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
- Reversing cryptographic algorithms
- Digital Right Management (DMR)
- Auditing program binaries

Security-related Reversing:
Malicious Software (malware)

- 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.

Security-related Reversing:
Reversing Crypto 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 Rights Management

- 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.

Security-related Reversing:
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
- Developing competing software
- Evaluating software quality and robustness

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.

Software Development-Reversing:
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.

Software Development-Reversing:
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

```
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
```

We know it's a factorial function

Check if value is zero or not

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.

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
```

RE exercise

RE exercise

RE exercise

Is RE legal?

- What social and economic impact does RE have on society?
  - Depends on what RE is used for …
- Whether a RE scenario is legal or not depends on many factors.

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.

Decompilers

- 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.

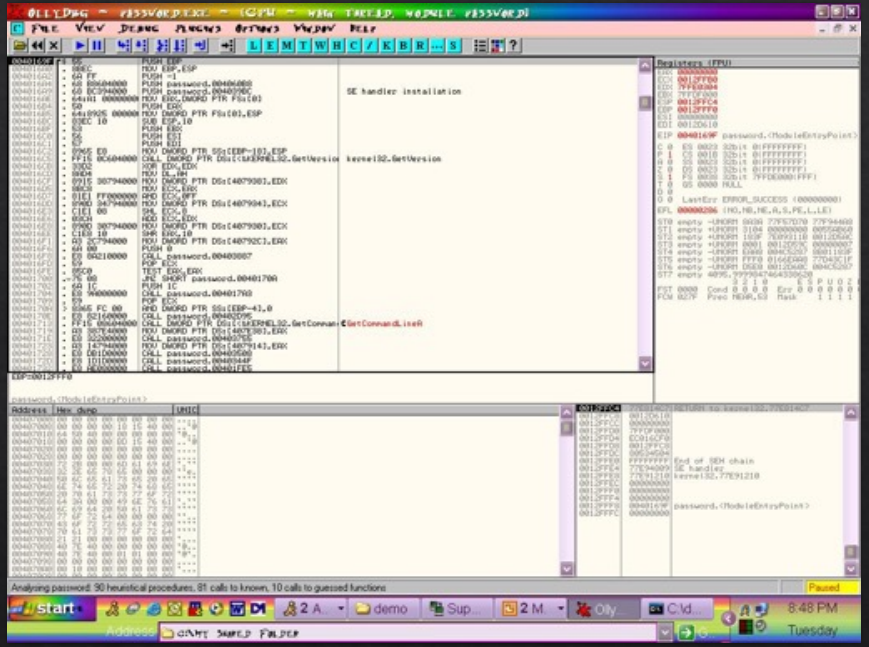
Disassemblers/Debuggers Programs & Features chart

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

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



Demo

Reverse Engineering Tool Categories

- Hex Editors
- Decompilers
- Disassemblers/Debuggers

Reverse Engineering Prevention Tools

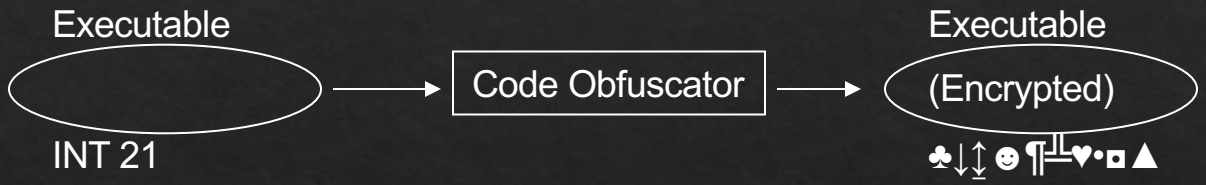
- **Code Obfuscators**

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	

Diagram of Code Obfuscators



Resources

- Decompiler tools
  - https://en.kali.tools/all/?category=decompiler

References

- Materials adopted from
- Spiros Mancoridis, Drexel University