UNIVERSITY OF GOTHENBURG

The practical defending of malicious reverse engineering

Bachelor of Science Thesis in the Programme Software Engineering&Management

Ce Wang
Siyang Suo

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, August 2013

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

The practical defending of malicious reverse engineering

Ce Wang
Siyang Suo

© Ce Wang, August 2013.
© Siyang Suo, August 2013.

Examiner: Rogardt Heldal

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: Cover picture taken from: http://www.librait.net/index.php?id=52

Department of Computer Science and Engineering
Göteborg, Sweden August 2013.

Abstract

Reverse engineering of software binary codes have reached an advanced state that can be effectively used by developers and attackers alike. In recent years, reverse engineering for harmful purpose appears to be commonplace and it has significant cost influence on industry sales and profitability. To accomplish this, the issues of useful software protection are becoming more and more popular.

This paper covers the working process of reverse engineering and the list of existing software protection technologies, we also implemented interview study to collect the result of software protection using situation in real companies. The aim of this study is to provide useful information about practical software protection technologies to against malicious reverse engineering. Furthermore, we suggest several software protection technologies to against malicious reverse engineering.

Keywords: reverse engineering, software protection.

I. Introduction

With the rapid development of computer and network technology, many new and advanced development technologies and design have been cited in the software development of thousands of shareware and commercial software. However, attackers are frequently trying all kinds of means to dig out the secret of these core technologies. If attackers steal these technologies, obtaining that knowledge can also reveal business strategies, thus harming the company in the competitive market. Therefore, in order to prevent their work achievements from easily being "drawing", various protection or encryption mechanisms are adopted.

From the paper (Chikofsky and Cross, 1990) prompting by Chikofsky and Cross, they defined “Reverse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction.". In fact, there are several approaches to reverse engineering (Drumm, 2009), two main approaches are representative. In the first situation source code is available, but other artifacts of the program such as documentation are poor or even no longer supported, the process of discovering these artifacts through source code can be seen as reverse engineering. In the second situation everything is available but the source code, which is opposite to the first case and the process of accessing the code level is regarded as reverse engineering.

There is a lot of research about software crack defending, and reverse engineering technology was one of sufficient defending solutions (Treude et al, 2011) due to its working mechanism. On the other hand, reverse engineering could also be a weapon for attackers (Coakley, Freeman, and Dick, 2005). During our research we confirmed that reverse engineering has already been involved in some business dispute due to the authority issue.

The focus of this research is Reverse Engineering by people with malicious purpose. Our problem is to study the practical defending of reverse engineering. Therefore we design an interview study in two Chinese software companies and the most famous Chinese software protection technical exchange on-line forum.

Research questions:

- How does the attacker use reverse engineering on software products and corresponding protection technologies?
- What software protection technologies can be used in practice to protect developers from malicious reverse engineering?

The paper is organized as follows: Section 2 the theoretical background about RE working mechanism and software protection methods. Section 3 describes the methodology that was used to collect and present the finding of the study. Section 4 presents the results of the interview study. Section 5 we discuss our finding based on the literature review and interview result. Finally, section 6 presents the conclusion of our study.

II. Background

Reverse engineering

The aim of reverse engineering is to extract information about software that is not revealed by the developer through any public interface. This information can be of various uses to the third-parties, and is often gathered for malicious activity.

Before the release of software, the source code program is always transformed into an object code program by compilation process. The object code is composed of code section and data section of the program, the code sections contain the information of machine code instructions of the program, and data sections contain the initialized and uninitialized data for the program (Hassan, Jiang and Holt, 2005). However, the code and data are represented in the same way in the object code (i.e., as a series of bytes), this makes it hard to distinguish them (Cifuentes, 2000).

Reverse engineering is able to invert the engineering process, this gives an ability to reverse engineer attempts to evaluate its functionality and determine how its internals are structured in order to obtain some information about the products' design. As the reverse engineering tools became more powerful, however, attackers began using them to automatically reverse engineer software developed by others (Colin, 2005; Chris et al., 2005; Coakley, Freeman, and Dick, 2005).

The reverse engineering can be implemented in documentation level and object code level (Drumm, 2009). In documentation level, the inverse process is depending on the program's documentation (Software requirement specification, software design specification，UML, etc). Nonetheless, the malicious reverse engineering are mainly focus on the object code level, there are three object code reverse engineering technologies were found during the research: obtaining the source code, emulation, and binary translation, each detailed in the following subsections.

Obtaining the source code

The software reverse engineering process can be seen as two phases to obtain the source code: disassembly using disassemblers and decompilation using decompilers (Kim et al., 2010； Kruegel et al., 2004； Cifuentes, 2000; Drumm, 2009). The source codes of software application are often transformed to object codes by to prevent access of proprietary algorithms or to make tampering with licensing verification procedures more difficult (Kruegel et al., 2004). The task of the disassembly phase is the extraction of the symbolic representation of the instructions (assembly code) from the program's binary image (Hsieh et al., 2001). This process transfers the unreadable object code to the readable assembly code, which will be easily to understand and as a stepping-stone for the transition of software's source code. The disassembly techniques can be categorized into two main classes: static and dynamic process (Kruegel et al., 2004; Colin, 2005). Static Disassembly aims to recover assembly language instructions from a software program without the invocation of the executable file, and dynamic disassembly is the process that the program is executed on some input and each executed instruction is monitored and decoded into its assembly equivalent (Colin, 2005).

On the other hand, the Decompilation is the process that takes as input a program in the form of an executable file, and produces a high level language representation of the program (Emmerik and Waddington 2004), The aim of this process is to produce a high-level language program that performs the same function as the executable program. The structure of a decompiler requires three modules (Yu, 2001):

- Front-end module: a machine- dependent module that reads in the program loads it into virtual memory and parses it.
- The Universal Decompiling Machine: a machine and language- independent module that analyses the program in memory.
- Back-end module: a language-dependent module that writes formatted output for the target language.

The output of this process is the source code of the target program, but the quality of the generated code is normally not the same as the original source code (Cifuentes, 2000). The meaningful variable names and comments are normally missing, however, it also provide wide information of understanding the original software methods.

Both disassembly and decompilation belong to the static analysis, however, in order to fully understand the software, both static and dynamic analysis needs to be extracted (Drumm, 2009; Sun et al., 2012; Stroulia and Systä, 2002). Static information describes the structure of the software in the way it is written in the source code, while dynamic information describes its run-time behavior. The dynamic analysis can produce sequential information, information about concurrency, code coverage, memory management and leaks (Stroulia and Systä, 2002). And for reverse engineering technology itself, if the attackers have a clear map of the structure, then it would accelerate the analysis procedure and

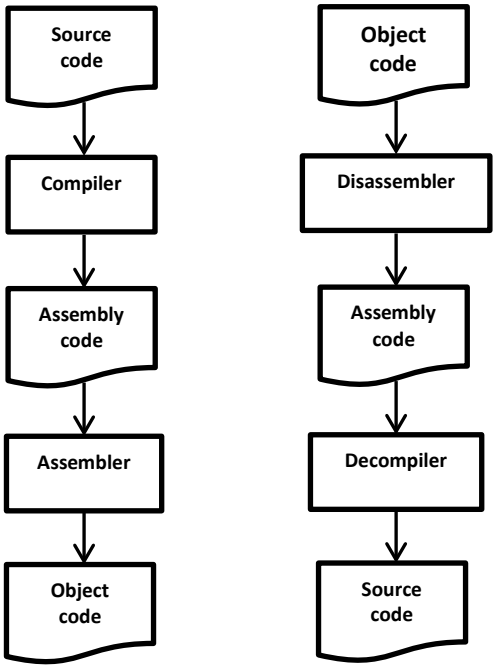

Figure 1: The compilation Process and The Decompilation Process

provide fast response for helping attackers remove the copy protection (Treude et al, 2011).

The most common used dynamic analysis is debugging (Drumm, 2009). Debugging is not a solitary and independent system, it is composed of multiple, layered debugging subsystems that collectively facilitate the debugging process, a canonical debugging system include a hardware layer beneath a software layer, and a not-to-be-underestimated human layer (Lesk, Stytz, and Trope, 2002).

Debuggers are able to set the breakpoints in the code by trace and record the code execution, in order to evaluate and manipulate the objects at those breakpoints. Recording these objects and understanding how the code traces through a certain path is the key to getting a good reversing (Drumm, 2009).

One important characteristic of software system is that there are a lot of performances –related properties, which are extremely important for assessing a software overall quality (Stroulia and Systä, 2002). However, these properties are usually not visible by static analysis, but they become apparent when it is analyzed by dynamic behavior. Such properties are like: memory management, code usage, and efficiency. Both static and dynamic views contain the information about software artifacts and their relations.

Emulation

Another process of reversing the object code is Emulation. Emulation is the process of running a program for a machine (the source machine) on another machine (the target machine) by decoding source machine code instructions and simulating the functionality of such instructions on the target machine (Cifuentes, 2000), the working flow is shown in figure 2. One of the biggest advantages of emulation is that it would convert a binary program for a real instruction set architecture (ISA) (Sharif et al., 2009).

At the begging, the emulation was mainly used for the programs which were written in assembly code and could not be migrated to the new machine without considerable time and expense in rewriting of the code. In recent years, the emulation on malicious purpose is focused on game consoles. Developers of popular game machines are suing or threatening to sue companies that make emulators for

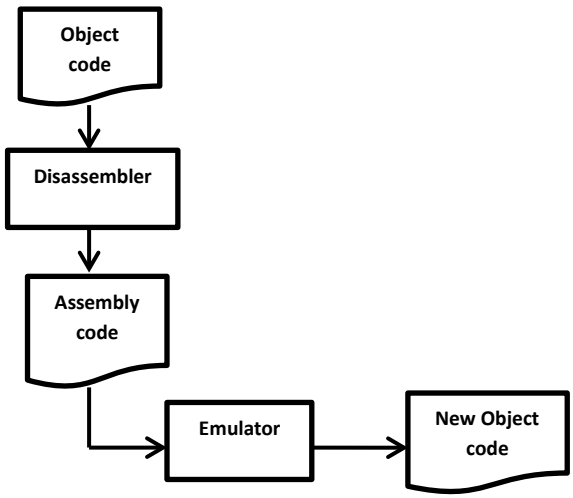

Figure 2: The Emulation Process

those machines (Cifuentes, 2000). For example, in 1999, Sony sued Connectix Corp for development of an emulator of the PlayStation machines which runs on a Macintosh. And the District Court issued an injunction against Connectix to prevent it to use the PlayStation BIOS in their emulator (Cifuentes, 2000).

Binary Translation

The binary translation is the process that automatically translating executable code for a source machine to a target machine by emitting native machine instructions

for the target machine, instead of emulating the source instructions (Cifuentes, 2000). The translator is able to generate functionally equivalent assembly code for the target machine, this feature can be used by the competitor to steeling the core function of the software. This technology provides ability to solve the software-inheritance problem and ISA-compatibility between different computers architecture (Shan, Guo, and Pang, 2012). The process is shown in Figure 3.

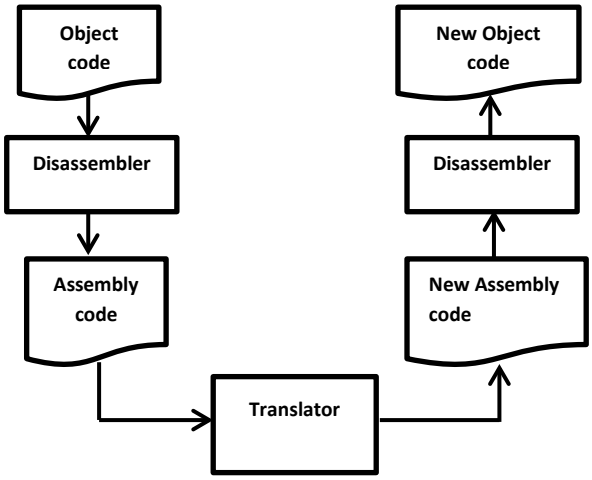

Figure 3: The Binary Translation Process

There are two ways to perform a binary translation process: statically and dynamically (Cifuentes, 2000; Shan, Guo, and Pang, 2012). Static approach involves the generation of a target object code program that can be run on the target machine, this normally need the help of a run-time interpreter for untranslated pieces of code. The dynamic approach does not involve the creation of a target binary code program, but only the dynamic execution of the generated code while the translation is taken places. Dynamic translation is in a sense similar to emulation, but it generates native code and optimizations of frequently executed pieces of code, resulting in more efficient techniques than emulation (Cifuentes, 2000).

Software protection

The group of mechanisms applied to software programs to aid them against malicious reverse engineering is called software protections (Kim et al., 2010; Colin, 2005). The proprietary software is distributed in a low-level representation from of the original program, all potential intellectual property within a software program is distributed in encoded form for execution if the environment is assumed as untrusted (Colin, 2005). Various technologies are developed in order to make the extraction of intellectual properties from binaries as difficult as possible, most of those techniques used to protect computer viruses in the late 1980s and early 1990s against detection and removal have been applied to the protection of modern software systems against reverse engineering (Colin, 2005). Even though these techniques can not eliminate the chance of reverse engineering, they can make the process more costly (Kim et al., 2010).

Economics can change substantially when a competitor cheaply and rapidly reverse engineers a pioneering design of others (McLoughlin, 2008). Their development costs are largely replaced by reverse engineering costs, if that costs is small, would enable the new product to easily undercut the pioneer device in price. This action will both shorten the market lead and sales of the pioneer company. On the other hand, the piracy by crack action will directly affect the profits of the companies. From engineering perspective, one promising solution to the problem of design theft and product piracy is to incorporate reverse engineering protection into the design of new products (McLoughlin, 2008; Oorschot, 2003). Overall, during the literature review, we found that the current software protection technologies which can against reverse engineering are: obfuscation, anti-debugging, tool detection, packing, encryption, self-checking, Steganograph, binary modification, and anti-virtualization.

Software Obfuscation is the technique to obscure the control flow as well as data structures that contain sensitive information and is used to mitigate the threat of reverse engineering (Schrittwieser and Katzenbeisser, 2011). Basically the purpose of the obfuscation is to makes the code more complex and confusing (Linn and Debray, 2003), it transforms a program into an equivalent but more complex structure that is more difficult to understand (Kim et al., 2010). It is the most widely used and heavily researched mechanism of software protection (Colin, 2005), and it is a powerful tools to against disassembly (Lin and Debray, 2003). Regarding to the target goals for the obfuscation, the methods can be categorized as follows: layout, data, and control obfuscations, and preventive transformation (Kruegel et al., 2004; Colin, 2005).

Anti-debugging is a technique to disable or confuse the attacking debugger (Kim et al., 2010). Anti-debugging techniques can determine if it is being debugged by identifying artifacts whether from the hardware, software, or human layers (Lesk, Stytz and Trope, 2007). If the developers can identify which debuggers the reverse engineering is likely to use, then they can write the program to determine if and when the execution environment is under control of these debuggers. The anti-debugging techniques can be classified as follows (Kim et al., 2010):

- API-based: checking for the existence of a debugger by analyzing the system information.
- Exception-based: to check if there is any exceptions are interfering.
- Process and thread blocks: check if the process and thread blocks have been manipulated.
- Modified code: check to see if there are any breakpoints which handled by debugger by analysis code modifications.
- Hardware- and register- based: check for hardware breakpoints and CPU register.
- Timing and Latency: check the time taken for the execution of instructions.

Packing is the process of known cryptographic and non-cryptographic transforms to a software program such that it is distributed in non-traditional from and decoded prior to execution (Colin, 2005). The tools used in this process are executable packer or executable compression, which can pack the source code into binaries or machine- code. Originally, packing was used to minimize the size of execution code for saving storage resources, in today, because compress code cannot be directly disassembled, so packing is widely used as an anti-reverse engineering technique (Kim. 2010).

Self-checking is the technique that verify no modification of instructions or data has occurred by the insertion of integrity verification statements within given source code segments (Colin, 2005).

Steganography is science of writing hidden messages that guarantee the messages are only available for intended recipient, it also can be a means to protect software (Kim et al., 2010). It utilizes redundancy in instruction representation and ordering to encode messages within a software program for version tracking purpose (Colin, 2005). It is always used to identify the original owner of a stolen software package (Colin, 2005).

Binary modification is one of the most basic protection techniques, it stripped or added a executable file to software structure, in order to confuse the reverse engineering process (Kim et al., 2010).

Encryption, developers are always stored the important source code or data in an encrypted form, in order to protect the privacy of them (Kim et al., 2010). The complexity of the encryption algorithm makes it greatly enhance the protection of critical information, if the crackers want to restore the encrypted information in a relatively short period of time is rather difficult.

Tool detection is the process that inserting specific code segments into a software program where the code can check the operating system characteristics for the run-time identification of reverse engineering tools (Kim et al., 2010). It is mainly used in anti-debugging to detect debuggers (Colin, 2005).

Anti-virtualization is the technique to prevent the attackers use virtual machines to study activities and techniques (Chen et al., 2008). Attackers can use virtualization to manipulating memory objects during run-time to exploit multiple execution paths of software and build a full view of the software behavior. And other advantage of using virtual machines is that attackers often attempt to distinguish systems running on virtual machines from those running on plain machines, so they can avoid potential monitoring by adversaries (Chen et al., 2008).

III. Methodology

This section provides the details of the research design for the study. We choose to conduct both literature review and interview study to achieve the research goals. The result of literature review is shown in Background section and the result of interview is shown in Findings setiction.

Literature review

Literature review is aiming to help the researchers to organize the essential understanding about reverse engineering process and existing software protection

technologies. In the literature review, we mainly focused on obtaining information on the proposed literature.

Source of information

The research was processed automatically by search engines of specific proceedings and journal papers. Each literature was reviewed by two researchers. The sources of the literature are well-known digital libraries, the reason of choosing these libraries is because they can provide a large amount of articles with reasonable good quality regarding the research conducted. The following digital libraries were used:

- IEEEXplore (http://ieeexplore.ieee.org/Xplore/guesthome.jsp)
- Springerlink (http://link.springer.com/)
- ACM Digital Library (http://dl.acm.org/dl.cfm)
- Scisence direct (http://www.sciencedirect.com/)
- Wiley InterSciene: (http://eu.wiley.com/WileyCDA/Section/index.html)

At the end, 20 articles from those sources were used during the literature review.

Search criteria

The selection criteria were decided in order to reduce the likelihood of bias and validity of resource. The search is based on three steps:

- Identify the keywords for searching: <reverse engineering> <software protection>.
- Search for synonyms of the key words, such as <against reverse engineering>, <reverse engineering resistant>, <software encryption> <disassembling>.
- Combine the different search terms together as one search term, such as <reverse engineering software protection>, <reverse engineering software encryption>, <software protection against reverse engineering>, <disassembling software protection> and so on.

By the search strategy presented above, there was a massive amount of the articles found in the database. Therefore, the include criteria were used to determine which literature will be used during the research, and exclusion criteria were used to determine which literature will be excluded.

Include criteria:

- The studies that describe the software protection in the context of reverse engineering;
- The studies that describe the process of reverse engineering from coding or binary aspect;
- The studies that describe the defending technologies for reverse engineering in coding or binary level;
- Only work published in English between 2000 and 2013 will be accepted.
- Literature must be peer reviewed.

Excluded Criteria:

- The studies describing software protection technologies but unrelated with reverse engineering.
- The studies that describe the process of reverse engineering in documentation aspect (Software requirement specification, software design specification, UML diagram).
- The studies that describe the defense of reverse engineering in documentation aspect (Software requirement specification, software design specification, UML diagram).
- The studies that are part or sections of a book.

Data extraction

The data collection focuses on solving the first research question. By reviewing the articles, our focus is on the issues of the process of reverse engineering and the correspondent software protections applied on reverse engineering.

Interview

The aim of the interview is to investigate the software protection technologies and reverse engineering more depth.

Research design

The data sources of the interview are two Chinese software development companies which are located in Beijing, and the most famous on-line software protection technical exchange forum. The first company

founded in 2000 and has around 180 employees now, their product is on-line downloader and provide the service of online storage. Two interviewees from this company are responsible for the product and web security, both of them have more than 5 years working experience in this area. The Second Company established in 2006 and owns around 80 employees, their product is PC game software. The two interviewees from this company are working on the software security department. The other two interviewees are the moderators of the online forum, they put the software reversing as a hobby and they reversed some small program at their off hours.

The four interviewees from companies were interviewed face-to-face, and the other two interviews were conducted through online chatting software.

The interview question is attached in appendix.

Data analysis

We conducted the thematic analysis to analyze the collected data which guided by Braun and Clarke (2006). Thematic analysis as the most common form of qualitative research and it focus on examining and recording patterns of the data. The process of data analysis is followings:

1. We went through the data and locate the raw data.
2. We went through the data again and generate the initial codes.
3. We categorize the codes into different themes.
4. Refine and review the themes.
5. We defining the names of each theme and started the writing.
6. Producing the report.

Threats to validity

The potential threat of the interview is that the two software companies were selected through convenience sampling instead of random sample. However, in this case, the software protection is a common and widely used technology, it does not depends on specific company culture. In order to guarantee the validity of result, we include two reverse engineering enthusiasts in the study.

IV. Findings

This section presents the results obtained from the thematic analysis on the data collected through the interviews.

Harm of malicious reverse engineering

From the response of our interviewees, the interesting thing is that the reverse engineering of entire competitor' software really isn't as popular in the software industry as one would expect. There are various reasons for this phenomenon, however, the main reason is that "*software is too complex, so the reverse engineering for competitive purposes in many cases is thought to be such a complicated and enormous process, which is uneconomical for the software development budget, It is always easier to independently develop the own software than reverse engineer the entire product from others."*

Although there is one common phenomenon in software industry, software companies developed most of the application independently, but they reversed highly complex or unusual components from other product and implemented into their own product. They can apply a decompilation process and recompile the new binary code which have same functions but with seemingly different code. The harm of this behavior is that "*it will stifle the innovation*", since the creative technologies can be easily attained by competitors, the developers will have little incentive to invest in development.

Reverse engineering also can be used as a tool to develop malicious software. Developers always reverse engineer the operating system or software to find the vulnerabilities in order to break the defense layers, "*people can use the malicious program to gain sensitive information and take the control of the system.*"

Most of all, all the interviewees agreed that the piracy is the biggest threat of malicious reverse engineering. The attackers could use reverse engineering to remove the copy protection of the software. Piracy is highly rely on cracking technology, which is the process of breaking the copy protection, is essentially one kind of reverse engineering. "*According to statistics released by authoritative department, the economic cost by piracy is around 50 billion every year."* One interviewee describes the current status of pirated software: "*Under the*

open architecture of today's computer program, it is impossible to develop a copy protection technology which is completely uncrackable. Even though it can be done in software-level, but crackers can still attack your program in hardware-level."

Programming language and reverse engineering (pp.33)

There are two interviewees from the software security technology forum expressed the idea that "*During the reversing process, in order to fully understand the program, you need to start analyze the underlying operation of the program, this is highly rely on the specific programming language during the development*." In this case, the important thing about programming languages is to understand how the language abstracts the underlying machine, for example, C language provides enough low-level perspective on machine level, the code can directly running on the target processor. On the other hand, Java language provides a separation between the code and underlying processer, instead it highly relay on the Virtual Machine. In this section, we will introduce the relationship between the reverse engineering and few currently most popular programming languages.

C

C language provides a directly control of memory pointers. When you run a program, the compiler will create specific platform program binaries, which include the machine code in the own native language of target processors.

"C language is easy to be reversed because it shortens the distance between the programmer and the machine. C code is similar to the machine code and the C compiler can directly translate the code to machine code and only add little Auxiliary code." So if you can understand machine code of the C program, it will be easily reconstructing the source code written by C language from the binaries.

C ++

C++ language is an extension of C language because they shared the basic syntax. C language put emphasis on design an arithmetic process to obtain the output, but then, the primary consideration of C++ language is to construct an object model which can fit the corresponding problem domain. Therefore people can get the output by accessing the object's state information.

"Reverse engineer a C++ program is similar to working with C program, but the programmer should focus on the class hierarchy, constructor calls, and identifying class method calls."

Java

Java is an object-oriented programming language, it compile Java bytecode rather than native assembly language. The Java code is interpreted by Java Virtual Machine.

"*The Java bytecode contain more information than the machine code created by native processor. Decompilation can be used on Java classes, the output is source-code-level representation of the program, which is much easier to understand than assembly language representation."*

C#

C# is another object-oriented programming language, C# code can be compiled to a bytecode format which was called Microsoft Intermediate Language (MSIL). MSIL code contain the detailed information of the data type of the program. The MSIL runs on the CLR (common language runtime), which is another runtime environment like Virtual Machine in Java.

"*Sometimes you need to learn the CLR-MSIL language if you intend to reverse a C# program, but most of time it is unnecessary to read the MSIL code manually, the decompilation process can generate a high-level language representation of the program, which is easily to read."*

Basic requirement for a successful reverse engineering

"Reverse engineering can be divided in two phases: system-level and code-level."

System-level reversing is the process of running a variety of tools, using different operating system services to gain

information, checking the executable file of program and tracking the input and output of the program. Most of the information is from the operating system, system-level reversing helps to determine the program structure and locate the interested area. Code-level reversing is a complex process of extracting the program design and code algorithms from binary code, it not only requires the engineer master the reverse engineering technologies, but also have a considerable understanding of software development, CPU and operating system. This section represents the basic requirement of a successful reverse engineering.

Low-level software

Low-level software is the generic name for the infrastructure of the software. In order to successfully implement a reverse engineering, it demands to have an indepth understanding of low-level software. Because "*the low-level information is all you can use during the reversing, high-level information are always deleted before the software hand over to the customers*." After all, the reverse engineering tools are only responsible for displaying the information. The useful information extracted by the engineer is from the result of reverse engineering tools, so the reverse engineering developers should have enough low-level software knowledge in order to extract such information. Following low- level software plays a significant role during the reverse engineering.

Assembly language

Some developers call assembly language "the reverse engineering language", because for most of the program, assembly language is the only way to obtain the original source code. Each operation of the software is corresponding to its own assembly code, the developer can obtain the information about how the operation is achieved by analyzing the assembly code. However, assembly language is a platform-specific language, each platform has its own assembly language. The reverse engineering developers cannot choose which assembly language they want to analyze.

Date management

All the operation of a software program requires input data, intermediate data room, and returned data. "*In order to understand a program, you must aware of the data management inside the program"*. For reverse engineering, the developers should be familiar with the low- level perspective of data management, which is from the machine's perspective. Because the high-level perspective (from software developers perspective) information were removed from the binary code before submit to customers. To understand the data management from machine's perspective is far more difficult than from human's perspective, because many high-level programming languages hide a large number of information about data management, so the developers have to observe the data flow from low-level constructs such as stacks, heaps, and registers.

Compilers

Compiler is the program that accept source code file and generate the corresponding machine code file. "*The biggest challenge of compilers during reversing is that modern compilers include optimization function, the compilers use various technologies to minimize the size of source code and increase the execution performance. The optimized code is usually violated the design thinking, which makes it difficult to understand*." To solve this problem, the engineer should improve the awareness of compilers processing model and the way of compilers understand the source code.

Bytecode

In Java language, it compiles bytecode instead of machine code, the Virtual Machines provide the ability to read the bytecode and execute the operation described in bytecode. Compare with machine code, the bytecode contains more information, such as in .Net executable files, it includes highly detailed data type information, which called metadata. From the metadata, we can get the information about classes, function parameters, local variable types and other information about the program. "*This feature enable developers to develop a very efficient decompilers, it can restore an excellent readability high-level language representation from bytecode files.*"

Control flow

"*A good understanding of control flow statements will help you to reconstruct the logic behind those*

statements." Control flow is an abstract representation of all possible execution sequence of events. By study control flow, it increases the ability to read the code and understand how the operations are achieved.

Reverse engineering tools

Reverse engineering cannot be implemented without the right tools, "*Now, there are hundreds of reverse engineering tools, some of them are free and some of them cost thousands dollars, however, there is no one tool includes all the features of reverse engineering. You need to choose a set of tools to achieve a reverse engineering goal."*

System monitoring tools

System monitoring tools are used to display the information of application and environment, which collected by operating system. Such information includes monitor networking activity, file accesses, registry accesses, I/O channels, etc. That information helps to understand the program structure.

Disassemblers

Disassemblers translate the binary machine code into a readable assembly code.

Decompilers

The purpose of decompilers is to produce a high-level language code from program executable binary files. However, "*we cannot get the exactly same source code of the original program, because some important information was deleted during the compiling.*"

Debuggers

Debuggers allow user to keep track of the program's runtime execution. "*The advantage of debugger is that it can execute a piece of code in the program and then pause, so the user can observe and even change the state of the program, and then the user can execute the next section of code. Developers can observe the exact process of the program under a relatively slow speed."*

Anti-reverse engineering technologies

All the interviewees agreed that "*To completely prevent reverse engineering is impossible, all we can do is increases the difficulty of the reversing, so the attacker may give up the idea*". On the other hand, "*Each anti-reversing method comes at a price, this price sometimes take up lots of CPU time, sometimes increase the size of code, sometimes may affect the stability of the program."*

"To be able to effectively prevent reverse engineering, the developers should use a combination of different approaches":

- **Eliminating Symbolic Information.** The aim of this approach is to delete any useful textual information from the program. For example, in the bytecode of Java language, it contains a large number of meaningful information such as class name, class member names, global objects' name, and so on. All those information is extremely important for the attackers. "*We can use an obfuscator to rename the entire symbolic name into meaningless characters*".
- **Obfuscating the program.** The idea of this approach is to modify the layout, logic, data, and organization in order to decrease the readability of the program, all the action is under the context of remain the program feature unchanged.
- **Embedding anti-debugger code.** The purpose of this approach is to perform some operation to damage the debugger or fail the debugger process when the system detects a debugger is tracking the program.

During the interview, all the interviewees agreed that currently most useful technologies to against reverse engineering are: anti-debugging, code obfuscation, code encryption and disassemblers confusing technology.

Anti-debugging technology

Strength: Most of the revering working is done in debugger process, without debugging, it is very difficult to understand the program's structure and code execution state.

Weakness: anti-debugging is a platform-specific technology and heavily rely on the specific operating system. Another weakness is that sometimes the anti-

debugger may make false judgment and cause malfunction in the program even though no debugger is present.

Process: Embedding specific debugger checking code into the program. The operating system provide some API of detecting debuggers for the program, for example, *IsDebuggerPresent* is a Windows API can be used to detect if the program is traced by debuggers. The developers can call this API in the program, if the return is TURE, and then the system can stop the debuggers. *"The more effective way is to achieve this function internally, inside the program code, in this case, your function cannot be easily found by attackers."* There are also some native APIs can be used to detective debuggers.

Another method is to use trap flag. Since the all kind of debuggers are using trap flag to step through the execution of software program, so" *the developer can add the trap flag code in the program, and then check the exception. If there is no exception that means your program is likely traced by debuggers."*

Code obfuscation

Strength: "*Obfuscated code is difficult to decompile, even though the decompilation is successful, the attackers are difficult to understand the real program's semantics."* The obfuscated code still follows the original file format and instruction set, the executed results are same as it before obfuscation. The variables, functions, and class names are transformed into English code after the obfuscation, so the code is very difficult to read. On the other hand, the obfuscation is not reversible. The information which does not affect the system operation will be lost permanently, the loss of that information make the program more difficult to understand.

Code Obfuscation is not rely on the specific platform, it achieved by modifying the code to hide the intend information. From 6 interviewees, 5 of them agreed that code obfuscation is currently most powerful technologies to against reverse engineering. "*This technology is greatly extended the time and difficulty of the reversing"*.

Weakness: Increase the size of code, slower execution speed, and increase memory runtime consumption.

Process: The software companies always use automatic obfuscator to make the transformations. Comparing with manually obfuscation, it can obfuscate the whole program rather than one part of the program. Automatic obfuscation is implemented after the compile process.

Code encryption

Strength: complicate the analysis process, prevent static analysis, a mature technology and easy to implement.

Weakness: In most situations, the decryption logic and decryption key are inside the executable file, once the attackers find that information, the encryption is useless. Some unpacker program can decrypt the program.

Process: The process of this technology is to encrypt the program before delivering to customers and embedding the decryption code in to the executable file. The program will decrypt the code in runtime before it is executed.

In order to against the unpacker program, *"A good strategy is to calculate the decryption key during the runtime of program."* To design the algorithm, the developers can maintain multiple global variables, which can be accessed and modified by different part of the program. Those variables can be a part of the complex mathematical formula during the decryption. In this method, the program will generate many keys and it is difficult for attackers to gain all the keys, the attackers need all the decryption keys to start the static analysis of the reverse engineering.

Disassembler confusing technology

Strength: this technology can confuse the disassemblers, so that the output of the disassemblers will contain many errors, such as directive dislocation and the output is not aligned with the object code.

Weakness: "*Disassembler confusing is not as powerful as code obfuscation, code encryption, and anti-debugging, experienced attackers can solve this technology easily."* Some advanced disassemblers (IDA Pro) allow the users add disassembly hits, so the disassembler will prompt the user when the error is present.

Process: This technology is able to confuse the disassembler that takes the invalid data as the beginning of an instruction, this makes the disassembler lose the synchronization. Once the disassembler loss the synchronization, each instruction cannot find the corresponding start address and end address of machine code. To accomplish this, the programmer can embed the junk byte (useless byte) or false branches, those data will confuse the disassemblers and force them generate the error output.

Other software protection technologies

"There are many other software protection technologies, some of them are not mature enough to integrate into the software development, some of them perform the almost the same function as those four technologies, but they do not have the equally effective results." For example, self-checking, tool detection, and anti-virtualization are also the dynamic analysis as anti-debugging, they share the same purpose to determine whether the program is traced or not. Binary modification and control flow transformations are both aiming to confuse the reverse engineering process, *"But they don't have strong effect as code obfuscation"*. Steganography is not widely used in software development, only one interviewee heard of this technology but not familiar with it.

A Theoretical Unbreakable model

The theoretical unbreakable model requires two conditions: first, "*The encryption key is long enough and the decryption algorithm is security enough, the decryption algorithm should not be easily found by attackers*" .Nowadays, this is a solution for this condition, the decryption algorithm is stored in the external devices such as USB flash drive. The user needs to plug in the USB flash drive to decrypt the sensitive information. However, this solution is not convenience for the customer and cannot prevent the second problem. "*How to avoid expose the decrypted data from the attackers after the decryption*." This problem is the major challenge for software protection, in recent year, there is some improvement in this area, such as trusted computing (the data of the program can only use on the user's system), but this technology is not well accepted by many software protection experts.

V. Discussion

In this section, we prevent the points we found during the research.

Weak link in software protection

One reason that restricts the development of software protection technology is software developers do not have clear understanding of software protection technology. The lack of experience and knowledge of developing fully functional software protection models directly affect the security of the software.

Despite developers' personal reasons, some software factors also led the restriction. First, nowadays the system's complexity, build-in scalability and universal connectivity of software are growing up quickly, these three growing trends make the software security issue more pressing than before. Secondly, with the increase of functional module, the combination of components and development tools also increase the software vulnerability. Thirdly, the customer interactions message might expose the software processing mechanism, which leaves potential problems for the software security.

Recommendations on software protection

1. *Strengthen the link between hardware and software*

For some software include highly sensitive information, increasing the cooperation between hardware and software protection technology is a good way to prevent the reverse engineering. Embedding the decryption code in the USB flash driver is not a convenient way for user to use the program, user should carry the driver all the time and there will be lots of trouble once the driver is lost. We think mount decryption hardware into the user's computer maybe a better way, the decryption algorithm is stored inside the hardware and that information should in great security level. In this case, attackers need user's computer to implement the decryption process and reverse engineering. This method requires the good cooperation between the software companies and computer manufacturers.

2. *Design own method of packing, compression, and obfuscation*

The reverse engineering tools are updated frequently, the common method of packing, compression and obfuscation are quickly involved inside those tools. Using the own method may confuse the reversing tools to generate the false output. If the software company does not have time and energy to develop such method, the developers should try to avoid the popular tools for those technologies.

3. *Add randomization checking into the software protection.*

The program can run the security checking at a random time instead of at the software start time. This also can be used to against piracy, in addition to check the registration code at boot time, the system can also check the registration code at some point of runtime in a random time.

4. *Decentralize the check registration and security code*

The developer should avoid calling same function or determine same global flag for registration checking or any other software protection technology. Otherwise, the attacker only needs to change one part of your code to crack the program.

5. *Online verification*

If the software requires internet service, the developer can add some code in to program to access information from the online database, that information is necessary for the program execution, and cannot be forged.

6. *File and function re-naming*

The developer should avoid use direct name on functions or files, such as *checkingLocialTime( )*、*key.dat*. All software protection related character string cannot directly store in clear text in the executable file, it is better to dynamically generate these strings.

7. *Distinguish the trial version and full version*

For many software programs, the only difference between the trial version and full version is the validation of license. Once the attacker cracked the trial version, they get the full version of the software. The better way is to distinguish the two versions and add more software protection technologies in full version.

8. *Service-oriented*

Try to make the software as a service, not only a technology in itself. If the software is only a technology, once the software is cracked the user does not need the software company anymore. On the other hand, if it is a service, even though the user has the cracked version of the software, they still need the service from the company and ask for technical support. Through this strategy will decrease the losses of cracking.

Feature of software protection

We believe the software protection development has a long way to go. Software system is the open architecture program, all the process of hardware and software can be observed, this makes hiding software protection technologies extremely difficult. By appropriate means and technologies, those protection methods can be removed by reverse engineering. In the short feature, we think the best way to protect the unique technology is increase the relevant law protection. Stronger legal sanctions can constrain the attackers' behavior.

VI. Conclusion

Software engineers originally developed the first reverse engineering tools to help automating the debugging process their software. Nowadays, some purposes of reverse engineering include security auditing, removal of copy protection, and circumvention of access restrictions. Many software developers worry about their applications being reverse engineered, so the issue of useful software protection is becoming more and more popular. The powerful software protection will keep the attackers away.

By the literature review, we present the common object code reverse engineering: obtaining the source code, emulation, and binary translation. We summarize the current software protection technologies: obfuscation, anti-debugging, anti-virtualization, packing, tool detection, encryption, self-checking, Steganograph, and binary modification.

From the interview, we describes the ham of malicious reverse engineering, the relationship between the

programming language and reverse engineering, the basic requirements of a successful reversing, the approach and currently most powerful software protection to against the reverse engineering.

The result of this study can provide useful information to companies which are trying to improve or research complementary software protection technologies. It will help the companies to learn more about how the reverse engineering is attacking the software and what their choices are to prevent the attacks. And the individual author of shareware can also use the information from the study to protect their software.

A major limitation of this study is that the reverse engineering is a complicated and extensive process, we only list the main techniques to study. Another limitation is due to the limited time and scope of this research, only 6 interviewees are involved in the survey study.

In future work, we would like to research this field in depth, for example, design a survey study to observe what software protection technologies are used in different companies. Implement a reverse engineering in practice, so we will have a better clue of how the reverse engineering works. We also hope to study the relevant law about against the malicious reverse engineering.

Acknowledgement

The authors wish to thank all the teachers in SE&M department for three years of education and help. Many thanks to Francisco G. de Oliveira Neto for his time and assistance in the supervision.

Reference:

Business software alliance (BSA). (2010). 2010 piracy study. *Eighth annual BSA Global Software.*

Braun, V and Clarke, V., (2006) Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77-101.

Cifuentes, C., (2000). The Impact of Copyright on the Development of Cutting Edge Binary Reverse Engineering Technology. *Software Engineering Australia*.

Colin W. Van Dyke., (2005)., Advance in Low-Level Software Protection, Ph. D. Thesis, Oregon State University.

Coakley, C., Freeman, J., and Dick, R., (2005), Next-Generation Protection against Reverse Engineering, Anacapa Science, Inc.

Chen, X., Mao, Z. M., Bailey, M., Nazario, J., (2008). Toward an Understanding of Anti-virtualization and Anti-debugging Behavior in Modern Malware. *Proceedings of DSN-DCCS*. pp177-186.

Chikofsky, E. J. & Cross, J. H., II (1990). "Reverse Engineering and Design Recovery: A Taxonomy". *IEEE Software* **7** (1): 13–17.

Drumm, P. M, (2009). Reverse Engineering Tools and Practices in Software Maintenace. Enterim LLC.

Emmerik, M. V., Waddington, Trent., (2004). Using a Decompiler for Real-World Recovery. *IEEE computer society*. pp. 27-36.

Hassan, A. E., Jiang, Z. M., Holt, R. C., (2005). Source versus object code extraction for recovering software architecture. *Reverse engineering, 12th Working Conference on.* Pp.7-11.

Hsieh, W. C., Engler, D., and Back, G., (2001). Reverse-Engineering Instruction Encodings. *In Proceedings of 11th USENIX Security Symposium.*

Kruegel, C., Robertson, W., Valeur, F., Vigna, G., (2004)., Static Disassembly of Obfuscated Binaries, *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA.

Kim, M. J., Lee J. Y., Chang, H. Y., Park, Y., Park, M., Wilsey, P. A., (2010). Design and Performance Evaluation of Binary Code Packing for Protecting Embedded Software against Reverse Engineering. *IEEE Computer Society*, pp. 80-86.

Lesk, M., Stytz, M. R., Trope, R. L. (2002). Software Protection through Anti-Debugging. *IEEE computer society*, pp.82-84.

Linn, C., Debray, S., (2003). Obfuscation of Executable Code to Improve Resistance to Static Disassembly. *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pp.290-299.

McLoughlin, I., (2008). Secure embedded systems: the threat of reverse engineering. *14th IEEE International Conference*, pp.729- 736.

Oorschot, P. C. V., (2003). Revisiting Software Protection. *Proceedings of the 6th International Conference, ISC 2003.* Pp.1- 13.

Schrittwieser, S., Katzenbeisser, S., (2011). Code Obfuscation against Static and Dynamic Reverse Engineering. *Information Hiding, computer science,* Volume 6958, pp.270-284

Shan, Z., Guo, H. R., and Pang, J. (2012) BTMD: A Framework of Binary Translation based Malcode Detector. *IEEE computer society*, pp. 39-43

Sharif, M., Lanzi, A. Giffin, J., Lee, W. (2009). Automatic Reverse Engineering of Malware Emulators. *IEEE computer society*. pp.94-109.

Sun, X. X., Chen, H., Wen, Y., Huang, M. H., (2012). Reversing Engineering Data Structures in Binary Programs: Overview and Case Study. *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing.* pp.400-404.

Stroulia, E., Systä, T., (2002). Dynamic analysis for reverse engineering and program understanding. *ACM SIGAPP Applied Computing Review*, Volume 10 Issue 1, pp.8-17

Treude, C., Fernando, F. F., Storey, M. A., and Salois, M., (2011). An Exploratory Study of Software Reverse Engineering in a Security Context. *IEEE Computer society.* pp. 184-188.

Yu, X., (2001). Decompilation of Binary Programs & Structuring Decompiled Graphs. *The University of Arizona*.

APPENDIX

Those questions were translated from the original Chinese version.

Interview Questions:

Can you introduce yourself?

Do you have any working experience about reverse engineering or anti-reverse engineering?

Can you tell us the threat or the harm of the malicious reverse engineering?

From business perspective, which malicious reverse engineering is more harmful for the organization, from business competitor or hacker?

Do you familiar with the process of reverse engineering?

What are the key factors of a successful reverse engineering?

Which software protection technologies do you thinks is most powerful for against reverse engineering?

Do those technologies have any weakness?

What do you think the future of software protection technologies?