ANALYSIS AND CONTEMPLATION OF REVERSE ENGINEERING TECHNIQUES

ROMIL GANDHI AND MADHURI GEDAM
Shree L. R. Tiwari College of Engineering, Department of Computer Engineering, Mumbai, India
romil.gandhi@gmail.com; madhuri.gedam@gmail.com

SACHIN BOJEWAR
Vidyalankar Institute of Technology, Department of Information Technology, Mumbai, India
sachin.bojewar@vit.edu.in

ABSTRACT: This paper discusses different uses of reverse engineering (RE), as well as the change in its original use for beneficial purposes to negative ones. A comparative study of different techniques used for RE is performed, and proposals to improve security are made. These include the obfuscation technique used for Java, and protection from RE using the proposed hardware system.

KEYWORDS: Reverse engineering, obfuscation, security, authentication.

INTRODUCTION

Reverse engineering (RE) is the process of obtaining development knowledge from the final form of an application or software. RE can also define as [4]: the art of problem dissecting and a critical part of problem solving process. RE requires command on the relevant programming language, and it was developed for various useful reasons, for example, if the relevant application or software no longer belongs to a given company, RE can be used to extract the code from the final product (executable) [2]. Another example is the case where the developer who initially created a particular application is no longer associated with the application, and consequently, the application cannot be revised, improved, or "debugged." In such a case, RE is used to recover the application logic and its functionality [5].

To recreate or redevelop a product from its final model is called reversing, e.g., to develop a car by analyzing the original model. This concept was slightly altered by "crackers," such that RE now refers to redeveloping a slightly modified product from its final model.

Currently, crackers use RE to redevelop licensed applications or software at significant cost to the original creator. As they gain more experience, crackers can crack applications/software more quickly. Reversers normally sell such RE software or make it available for free on the Internet. The greater the security implemented to thwart illegal RE, the greater the effort invested by crackers to evade these security measures.

Section II contains a discussion of various tools typically used for RE. In this study, software security and its consequences are discussed; certain software available on the Internet is for study purposes only. The methods to reverse engineer this software include using such tools and procedures as providing a patch for the executable (.exe) file, finding the serial key that allows full use of the software, and providing necessary files, such as dynamic link libraries (DLLs). Section

III describes some Java obfuscation types and determines the one that can be easily implemented in any software application. Section IV describes the piracy rates in major countries, and Section V proposes a method to enhance security against illegal RE.

INTRODUCTION TO REVERSE ENGINEERING TOOLS

The tools required for RE depend on the application in question. Some basic tools include Ollydbg, PEID, Reflector, and Deobfuscator. Ollydbg is a debugger widely available on the Internet. PEID is a small software program that analyzes the programming language used by a specific application, as well as the compression or packing used, if any. Reflector is used to reverse .Net code written in Visual Studio, and Deobfuscator is used to reverse obfuscation implemented in Java. However, Deobfuscator does not always work properly, it is quite difficult for a reverser to understand, and requires considerable experience.

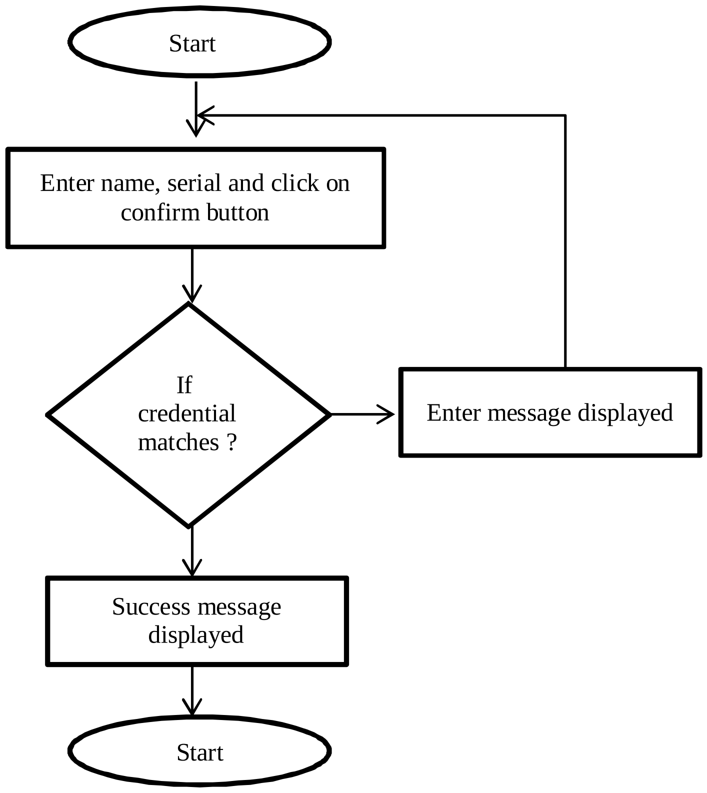

Start
Enter name, serial and click on confirm button
If credential matches ?
Enter message displayed
Success message displayed
Start

Figure 1. Flowchart explaining the functionality of an experimented application

Three applications with different levels of security are discussed in this paper. The first example is quite simple and requires finding the serial key by disassembling the executable file. Fig. 1 shows the Flowchart discussed for RE. For the application, Ollydbg is used as the debugger because it helps to open packed/unpacked executable files in the assembly language, thus helping to reverse engineer the application and determine the logic required to identify the serial key. Fig. 2 shows the code executed when the “Confirm” button on the application is clicked in the disassembler. This is an iterative process that helps to debug the application when using the debugger. When the Confirm button is clicked, the application is managed through the debugger, which executes the application’s code line by line (similar to the interpreter), and in this way, the reverser can determine the application’s logic.

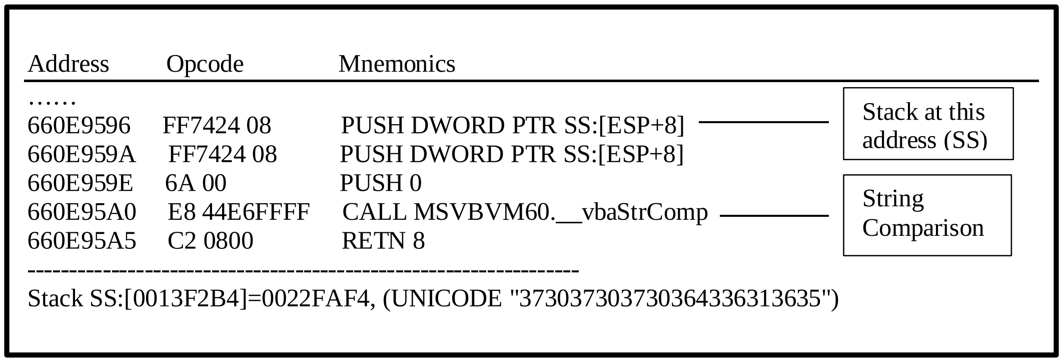
Note that Fig. 2 shows Unicode, which means that Ollydbg converts the code to hexadecimal, and then it determines whether the user has entered the correct key. Through repeated analysis, it is determined that, if a username is longer than six characters, all characters after the sixth character are discarded.

For example, consider the username “apple.” The program proceeds as follows:
First, the first and last characters in the username (“a” and “e”) are concatenated.
Step 1: ae
Next, the same process is repeated for the remaining characters (“ppl”), and the new first and last characters are prefixed to the previous string.
Step 2: plae
The process is repeated until there are no characters left in the original string, and result is obtained.
Step 3: pplae

However, if the result in Step 3 has an odd number of characters, the first character in the final result (“p”) is prefixed to the string in order to ensure the string has an even number of characters.

Step 4: ppplae

After Step 4, the result is converted to hexadecimal notation (HEX).
Step 5: 7070706C6165
This HEX code is the “key,” and the username is compared with the key.



```
Address       Opcode          Mnemonics
……
660E9596     FF7424 08       PUSH DWORD PTR SS:[ESP+8]
660E959A     FF7424 08       PUSH DWORD PTR SS:[ESP+8]
660E959E     6A 00           PUSH 0
660E95A0     E8 44E6FFFF     CALL MSVBVM60.__vbaStrComp
660E95A5     C2 0800         RETN 8
-----------------------------------------------------------------
Stack SS:[0013F2B4]=0022FAF4, (UNICODE "373037303730364336313635")
```

Stack at this address (SS)
String Comparison

Figure 2. PlanetHaX application, partial assembly code

Thus, it was fairly easy to debug the first application, and the key required by the program was found. Note that only Ollydbg was used. However, there is another method to reverse engineer applications, though it does not qualify as actual reversing. According to this method, bad code can be jumped by changing the operation code so that the application always moves to the good code and never receives an error message. In other words, security is bypassed through JMP instruction (i.e., unconditional jump) to move to the desired message, and receive no error messages.

For the second application examined in this paper, the case where a user wants the full version of the software when only the demo version is available is considered. When the demo version is downloaded, it provides a 30-day trial. If the "Enter Key" option is selected, the demo version asks for "Full Name" and "Registration Key."

Once the program opens in the debugger and is analyzed, the key for the professional version of the software (Fig. 3) and the keys for several other products are discovered.

There is software on the Internet not adequately secure, and hence, the companies that develop such software suffer large losses because of the illegal proliferation of the software based on RE techniques. The key factors are time and money, which do not guarantee security following implementation.

The third application presented for examination is an instance of the reversing of a .jar file (Fig. 4). Jd-gui generates the original code as originally written. The example used here is intended to show the functioning of jd-gui, which is the Java debugger.

The login application asks for the username and password, which are unknown to the cracker. The cracker simply needs to open the application in jd-gui, which provides the code. As can be seen in Fig. 4, the entire class file is generated and the username and password become available. In this manner, a normal Java program with low security is easy to reverse. Using similar techniques, .Net applications can be reversed engineered using the .Net Reflector, which provides the .net code that helps to find the application's logic, similar to jd-gui.

```
006AE35E   MOV EDX,SuperEZW.006AF974                    ASCII "GWP-TYBKU-6UY80-UIFK7-
ER56J"

006AE363   CALL SuperEZW.00404DF0

006AE368   LEA EAX,DWORD PTR DS:[EBX+8]
006AE36B   MOV EDX,SuperEZW.006AF974                    ASCII "GWP-TYBKU-6UY80-UIFK7-
ER56J"

006AE370   CALL SuperEZW.00404DF0

006AE375   LEA EAX,DWORD PTR DS:[EBX+C]
006AE378   MOV EDX,SuperEZW.006AF974                    ASCII "GWP-TYBKU-6UY80-UIFK7-
ER56J"

006AE37D   CALL SuperEZW.00404DF0

006AE382   LEA EAX,DWORD PTR DS:[EBX+10]

006AE385   MOV EDX,SuperEZW.006AF998                    ASCII "SOFTWARE\\WaveEditorPro\\"

006AE38A   CALL SuperEZW.00404DF0

006AE38F   LEA EAX,DWORD PTR DS:[EBX+14]

006AE392   MOV EDX,SuperEZW.006AF8D0                    ASCII "http://www.wav-editor.com/"
```

```
006AE397   CALL SuperEZW.00404DF0

006AE39C   LEA EAX,DWORD PTR DS:[EBX+18]
006AE39F   MOV EDX,SuperEZW.006AF8F4                              ASCII "http://www.wav-
editor.com/support.html"

006AE3A4   CALL SuperEZW.00404DF0

006AE3A9   LEA EAX,DWORD PTR DS:[EBX+1C]

006AE3AC   MOV EDX,SuperEZW.006AF8D0                              ASCII "http://www.wav-editor.com/"

006AE3B1   CALL SuperEZW.00404DF0

006AE3B6   LEA EAX,DWORD PTR DS:[EBX+20]

006AE3B9   MOV EDX,SuperEZW.006AF8D0                              ASCII "http://www.wav-editor.com/"

006AE3BE   CALL SuperEZW.00404DF0

006AE3C3   LEA EAX,DWORD PTR DS:[EBX+24]
006AE3C6   MOV EDX,SuperEZW.006AF924                              ASCII "http://www.wav-
editor.com/order.html"

006AE3CB   CALL SuperEZW.00404DF0

006AE3D0   LEA EAX,DWORD PTR DS:[EBX+28]
006AE3D3   MOV EDX,SuperEZW.006AF924                              ASCII "http://www.wav-
editor.com/order.html"

006AE3D8   CALL SuperEZW.00404DF0

……..

006AE406   JNZ SuperEZW.006AE51D

006AE40C   XOR EDX,EDX                                            SuperEZW.<ModuleEntryPoint>

006AE40E   MOV EAX,DWORD PTR DS:[ESI+4EC]

006AE414   MOV ECX,DWORD PTR DS:[EAX]

006AE416   CALL DWORD PTR DS:[ECX+44]
```

Figure 3. SuperEZ Wave Editor code when opened in Ollydbg

```
package com.login;
public class Login
{
  public static boolean authenticate(String username, String password)
  {
    if ((username.equals("VenDeTt@")) && password.equals("$2BoR0toBe#")))
{       return true;
    }
    return false;  }}
```

Figure 4. JDialog demo retrieved code in jd-gui

DIFFERENTIATING OBFUSCATION TYPES IN JAVA

This section discusses Java obfuscation types [3], given that Java is considered the preferred language among programmers for its platform-independent features, functionality, and easy-to-use concept. Java has two types of obfuscation techniques [3]: i) source code obfuscation, and ii) byte code obfuscation. Source code obfuscation is not as secure as byte code obfuscation. The source code technique changes the identifiers, which makes the code more difficult to read and reverse. However, it does not change the functionality of the code or its execution. Nonetheless, experienced reversers can extract the partial original code, following which it becomes easy to guess and recover the entire code for the application in question. A popular tool for this is the DeDe deobfuscator. Because the availability of several online tools to deobfuscate code is the main security concern in Java, developers have to consider security while developing applications on the platform.

Byte code obfuscation replaces the classes, packages, field names, methods, etc., with characters that are difficult to read (special characters, and sometimes, symbols). Byte code obfuscation can also be reversed, but this is much more difficult than reversing source code obfuscation. A byte code obfuscator that is freely available on the Internet is yGUARD.

If, following the application of obfuscation, the relevant Java code is not as secure as required, it could be because any reverser can extract (unencrypted) code from the Java Virtual Machine (JVM), which is the compiler that executes all Java code.

PIRACY

This section shows the extent of piracy in various countries. The data is obtained from Business Software Alliance (BSA) reports [1].

As Fig. 5 shows, piracy was practiced the most in Pakistan in 2010, followed by Thailand, India, Malaysia and others. Results from a similar report by the BSA in 2011 are also shown (Fig. 6). It is apparent that the piracy percentage for almost all countries decreased in 2011, but the maximum reduction was only 3%. The piracy rates remain high, and enhanced security is needed to prevent it.

HARDWARE PROTECTION SCHEME

There are several methods to improve security; however, none of these methods is infallible. For example, developers can use obfuscators, or change the physical address or starting address of an application. A few companies attempt to protect code from debuggers by implementing anti-debuggers; however, as mentioned above, these do not work.

The system proposed in this paper is intended to enhance security. The hardware used is a dongle with read-only memory (ROM) that contains the actual code and logic of the application in question. The system is divided in two parts, hardware and software. The software part contains the code, and it only consists of function invocations. One function calls another function in order to execute code. The proposed system consists of anti-debugging code with the ability to find malicious code, such as keyloggers. This makes the system more difficult to reverse engineer.
The hardware part consists of the actual code and logic of the application. All information is obfuscated, and almost everything is secured.

The software initiates if the dongle is connected to the system. The invoking function calls the functions written in the dongle and executes the code. The procedure seems simple, but it is

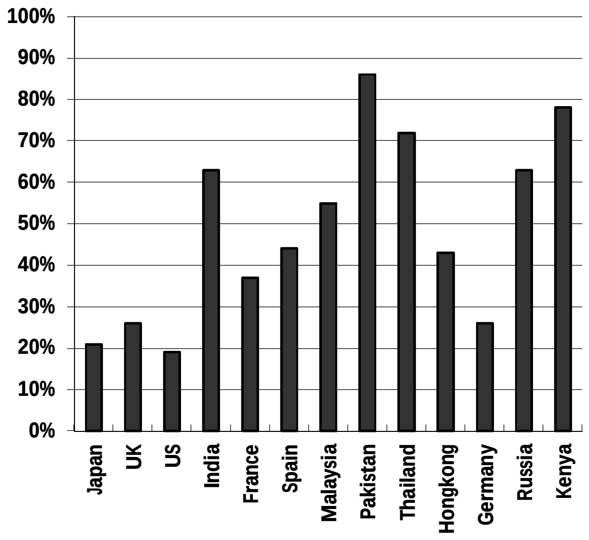


Figure 5. BSA report (2010)

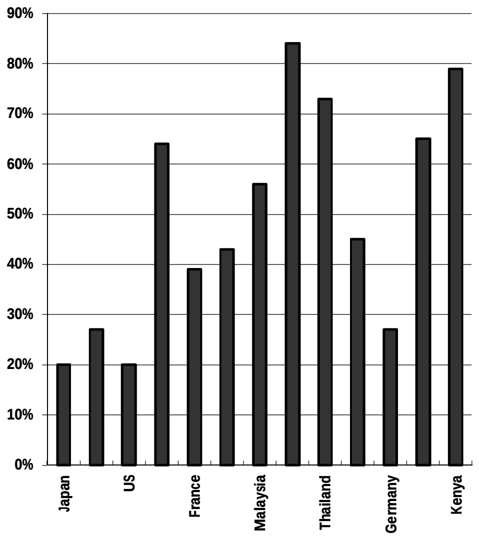


Figure 6. BSA report (2011)

difficult for reversers to overcome: without hardware, they cannot reverse the logic. Thus, at least initially, users who want to use the application need to buy the software. If a reverser attempts to intrude into the code in the dongle, the code is obfuscated, and hence, the reverser requires more effort.

Thus, security is improved by adding more logic and functions. The dongle can be provided with a machine ID so that no new machine can use the same hardware. These are only possible security enhancements that companies can implement to avoid losses caused by illegal RE.

The demo version of the downloaded software should provide no core operations, it should be developed separately from the full version, and it should not contain the logic of the application. In this manner, even if the software is reversed, none of the functions can be activated and used.

CONCLUSION

Most reversers need easy methods to reverse engineer systems and avoid laborious techniques. The proposed hardware and obfuscation will render applications more robust against RE. Instances of reversing carried out using the applications: PlanetHaX KeyGen, SuperEZ Wave Editor and JDialog Demo, prove that most applications available online are easy to crack and hence require better security. We hope that our proposed system will improve security and authentication of the application will have new way of implementation.

REFERENCES

Al-Hakimy, A.M., Rajadurai, K.P., and Ravi, M.I., "Formulating a defensive technique to prevent the threat of prohibited reverse engineering," *Current Trends in Information Technology (CTIT), 2011 International Conference and Workshop*, Dubai, pp. 82–85.

Von Mayrhauser, A. and Vans, A.M., "From code understanding needs to reverse engineering tool capabilities," *Computer-Aided Software Engineering, 1993. CASE '93. Proceeding of the Sixth International Workshop*, pp. 230–239.

S. Shah, P. Nixon, R.I. Ferguson, S.R. ul Hassnain, N. Arbab and Khan. L. "Securing Java-Based Mobile Agents through Byte Code Obfuscation Techniques", *Multitopic Conference, 2006. INMIC '06. IEEE*, pp. 305–308.

I. Klimek, M. Keltika, F. Jakab, "Reverse engineering as an education tool in computer science", *Emerging eLearning Technologies and Applications (ICETA), 2011 9th International Conference*, pp. 123–126.

K. Gallagher, C. Kaner, J. Deignan, "The Law and Reverse Engineering", *19th Working Conference on Reverse Engineering (WCRE), IEEE, 2012*. pp. 3–4.