Volume 2, Issue 4, April 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Exploring the two faces of Software Reverse Engineering

[1]Dr. Oladipo Onaolapo Francisca,[2]Dr. Odoh McChester Onyemaechi,
[3]Dr Onyesolu Moses Okechukwu
Department of Computer Science
Nnamdi Azikiwe University, Awka, Nigeria
Nigeria
of.oladipo@unizik.edu.ng

Abstract— ***The original concerns of Software Reverse Engineering was with the problem of understanding the architecture of a software application for the purpose of maintenance and re-engineering; it was conceived as a process of examination to unearth the technological principles of a software through the analysis of its structures, functions and operations in order to recreate and not necessarily copying from the original. However, attackers have leveraged on the openness of the concept to explore the vulnerabilities of a software system thereby making the technology an open-ended research area, This paper examined the concept and limitations of software reverse engineering as it related to applications security. The authors presented the good (deployment for maintenance, ensuring code consistency during migration, etc) and evils of software reverse engineering, that is how the process can be adopted for software tampering and how an attacker can explore the vulnerabilities of a software system through the analysis of the dynamic behaviour of the software system, and so on. The paper also presented some the use of structural error-based techniques such as watermarking, obfuscation and mutation analysis to increase the chances of detecting code related security breaches. Based on the facts presented in this work, we recommended engineering software systems with credible technical defenses against code-level breaches, while still adhering to the virtue of openness in the research community.***

Keywords: Analysis, Reverse engineering software, Software tampering, obfuscation.

I. INTRODUCTION

Software is a set of instructions that determines what a general-purpose computer will do. Thus, in some sense, a software program is an instantiation of a particular machine (made up of the computer and its instructions). Machines like this obviously have explicit rules and well-defined behavior. Although we can watch this behavior unfold as we run a program on a machine, looking at the code and coming to an understanding of the inner workings of a program sometimes takes more effort. In some cases the source code for a program is available for us to examine; other times, it is not. Therefore, attack techniques must not always rely on having source code. In fact, some attack techniques are valuable regardless of the availability of source code. Other techniques can actually reconstruct the source code from the machine instructions [1]. Research results had revealed that 92% of exploitable vulnerabilities are in software [2].

Business software is more accessible than ever. Even legacy and in-house applications are now available from the web, in the cloud, and on mobile devices. As a consequence, today's applications can extend far beyond the reach of the best perimeter defenses, leaving them and the sensitive information at the core of your enterprise wholly unprotected. Hackers, organized crime cartels, and rogue governments are highly skilled at exploiting vulnerabilities in software to:

- Steal data, customer identities, intellectual property, and cash
- Disrupt business operations
- Inflict brand damage
- Place employees, customers, and the public at risk [2]

A March 2011 study by the Panemon Institute California, USA revealed an average organizational cost per security breach of $7.2M in the US [3] A similar study conducted earlier in 2009 and released in 2010 presented the consolidated analysis of five national cost of data breach studies: United States, United Kingdom, Germany, France and Australia (all converted into US dollars); showed alarming figures for these countries [4]. The Symantec Internet Security Threat Report, April 2011 revealed a 93% increase in web attacks from 2009 to 2010 [5] and the Juniper Networks study, May 2011presented a 250% increase in mobile malware from 2009 to 2010 [6][7] believed that any software can be cracked and defined reverse engineering (RE) software as the process of analysis of its structure, function and operation This paper examined the original research intention of software reverse engineering and the malicious deployments of the concept. We presented the concept as an open-ended research area and describe a number of structural techniques to prevent security breaches while still preserving the noble intentions of the researchers in this area. This paper is divided into six main sections. Section I provided the introduction to the research work and an empirical support for vulnerabilities in software systems. Section II described the technology of reverse engineering of software system and an exploration of the research intentions of the field was described in section III. The malicious deployments of the techniques of software reverse engineering were analysed in section IV and the structural threat mitigating techniques were described in section V. The conclusion and recommendations were presented in section VI.

II. SOFTWARE REVERSE ENGINEERING

Software Reverse Engineering (RE) was defined as the process of analyzing a subject system to create

representations of the system at a higher level of abstraction in order to unravel the complexities of target software or to grok the assumptions made by the people who created the system and then undermine those assumptions. It may involve going backwards through the development cycle. It is a process of examination to unearth the technological principles of a software through the analysis of its structures, functions and operations in order to recreate and not necessarily copy from the original. The methods and technologies play an important role in many software engineering tasks, such as program comprehension, system migrations, and software evolution [8].

A report by Hoglund and McGraw (2004) stated that reverse engineering allows one to learn about a program's structure and its logic thereby leading to some critical insights regarding how a program functions. This kind of insight is extremely useful when the aim is to exploit software. The researchers believed that there are obvious advantages to be had from reverse engineering. For example, one can learn the kind of system functions a target program is using and learn the files the target program accesses. One can also learn the protocols the target software uses and how it communicates with other parts of the target network [1].

The most powerful advantage to reversing is that it can be enable one to change a program's structure and thus directly affect its logical flow. Technically this activity is called *patching*, because it involves placing new code patches (in a seamless manner) over the original code, much like a patch stitched on a blanket. Patching allows the engineer to add commands or change the way particular function calls work. This enables the addition of secret features, removal or disabling functions, and fixing of security bugs without source code. A common use of patching in the computer underground involves removing copy protection mechanisms. Like any skill, reverse engineering can be used for good and for bad ends [1].

Reverse engineering of software can be accomplished by various methods. The three main groups of software reverse engineering are

1) Analysis through observation of information exchange
2) Disassembly using a disassembler to read and understand the raw machine language of the program in its own terms.
3) Decompilation using a decompiler, a process that tries, with varying results, to recreate the source code in some high-level language for a program only available in machine code or bytecode [9].

III. PURPOSES OF SOFTWARE REVERSE ENGINEERING

Software RE is necessary because there will always be "old" otherwise called legacy applications [10]. Developers today inherit a huge legacy of existing software. These systems are inherently difficult to understand and maintain because of their size and complexity as well as their evolution history [11]. To address the problem of program understanding, software engineers are spending an ever-growing amount of effort on reverse engineering technologies [8].

Originally, legacy code was used to refer to programs written in COBOL, typically for large mainframe systems. However, today's software developers predominantly use Object Oriented languages like C++ and Java. This means that tomorrow's legacy code is being written today. Klosch 1996 opined that most of the aims of reverse engineering are closely related but not limited to software maintenance. The author went further to define some of the purposes of the technology of reverse engineering software as:

i. Reverse Engineering software provides a means to recover lost information by providing proper system documentation. Recovering lost information means both the development of never existing design documents as well as recovering information that has been lost during software development or even during years of maintenance operations. In most cases in the history of a software system, the original designers are no longer part of the development team, and 'strangers' may have to carry out the maintenance of the system [10]. In this situation, recovery of various kinds of information about the software system becomes very important and reverse engineering techniques provide the means for recovering lost information and developing alternative representations of a system, such as generation of structure charts, dataflow diagrams, entity-relationship diagrams, etc [11].
ii. Reverse Engineering supports the migration to another hardware/software platform or integration into a CASE environment. Since the development of applications is usually not completely independent of the underlying hardware/software environment, changes of those platforms also require adaptations of the particular application [10].
iii. The techniques of reverse engineering facilitated software reuse through providing support for the definition, development and identification of reusable components within existing systems [10].
iv. Reverse engineering assists with corrective and adaptive maintenance by several techniques, such as providing additional documentation and restructuring [10].
v. Reverse Engineering of Software or hardware systems can be done to support undocumented file formats or undocumented hardware peripherals thereby providing support for interoperability and enabling the software to run across several hardware platforms.
vi. Software Reverse Engineering can be deployed for security audit, removal of copy protection, circumvention of access restrictions often present in consumer electronics, customization of embedded systems, in-house repairs or retrofits, enabling of additional features on low-cost hardware or can be done for mere satisfaction of curiosity [9]. This purpose may sound like an infringement on copyright; but for researchers, it is may actually be a check for patent infringement [12].
vii. Software Reverse Engineering provides the techniques for ethical hacking; a term for the practice, by those with sufficient skills and with the advance permission of the system owners, of breaking into computer systems to demonstrate security weaknesses. The term, ethical hackers, having a positive connotation, is associated with those using their skills for legitimate purposes, e.g. computer security experts doing system research or vulnerability testing to better defend against attacks. This is in contrast to unethical hacker, having a negative connotation, denotes unauthorized individuals who break in to computer systems for illegitimate purposes – thus being synonymous with crackers [13].

Based on those aims various benefits can be achieved, such as: maintenance cost savings, quality improvements, competitive advantages, or software reuse facilitation. Savings of cost during the maintenance phase are obvious when considering the important role of the maintenance phase within the whole life-cycle [14]. Reverse engineering supports the improvement of software quality not only on the source-code level, but also on higher levels of abstractions, such as the design or requirements level, by providing alternative views and varying representations of the system. Those activities are also effective in the elimination of redundancies in the system which were often the reason for quality deterioration during maintenance operations. Decompilers, one of the important tools of software RE have many legitimate uses which include code unification, ensuring code consistency and vendor product maintenance.

Wikipedia [9], gave the following as reasons for reverse engineering among others:

i. Software Modernization: reverse engineering is generally needed in order to understand the 'as is' state of existing or legacy software in order to properly estimate the effort required to migrate system knowledge into a 'to be' state. Much of this may be driven by changing functional, compliance or security requirements.
ii. Product analysis. To examine how a product works, what components it consists of, estimate costs, and identify potential patent infringement.
iii. Digital update/correction. To update the digital version (e.g. CAD model) of an object to match an "as-built" condition.
iv. Acquiring sensitive data by disassembling and analysing the design of a system component.
v. Military or commercial espionage. Learning about an enemy's or competitor's latest research by stealing or capturing a prototype and dismantling it.
vi. Creation of unlicensed/unapproved duplicates.
vii. Academic/learning purposes.
viii. Curiosity.
ix. Competitive technical intelligence (understand what your competitor is actually doing, versus what they say they are doing).
x. Learning: learn from others' mistakes. Do not make the same mistakes that others have already made and subsequently corrected [9].

IV. MALICIOUS DEPLOYMENT OF SOFTWARE RE

According to Hoglund and McGraw (2004); "Because reverse engineering can be used to reconstruct source code, it walks a fine line in intellectual property law". The researchers also stated that "the single most important skill of a potential attacker is the ability to unravel the complexities of target software". They referred to this skill as *reverse engineering* or sometimes just *reversing*. The authors further believe that software attackers are great tool users, but exploiting software is not magic and there are no magic software exploitation tools. To break a nontrivial target program, an attacker must manipulate the target software in unusual ways. So although an attack almost always involves tools (disassemblers, scripting engines, input generators), these tools tend to be fairly basic. The real smarts remain the attacker's prerogative [1].

Grimm, (2004) believed that the term, reverse engineering: implies unethical behavior, lacking meaning and conjuring up images of the past. He contended that the most widespread use of reverse engineering is in software development and decompiling computer code, thereby providing a direct access attack with the application source code as the target. Analysis and tampering typically involve direct access to the target code, involving a skilled attacker, with sufficient resources and time to manipulate the code in a controlled environment [15].

According to Main and van Oorschot (2003), a threat model identifies the threats a system is designed to counter, taking into account the nature of relevant classes of attackers (including their expected attack approaches and resources – e.g. techniques, tools, powers, geographic access), as well as other environmental assumptions and conditions [13]. Van Oorschot (2003), defined direct access attacks to be those developed on a local machine using a local copy of the target code. However in some cases, the term may also include attacks developed over a network connection; the main point is direct human involvement: this is the bad news of Reverse engineering, the process cannot be fully automated, human interference is possible. Sadly, an attack on a slice of the program is an attack on the entire application because a slice is a subprogram which, behave upon termination like the entire program. Reverse engineering for malicious purpose – e.g. theft of intellectual property (such as a competitor's secret formula or process), software tampering, or the discovery and exploitation of vulnerabilities – is facilitated by a number of advanced program analysis tools which also serve the legitimate software development community, e.g. in debugging, software engineering, and understanding malware [16].

Chandran (2008) believed in the open-ended attribute of software reverse engineering. He described Software reverse engineering as the technique of getting the original source code from the binary. He also stated that competitors might use reverse engineering to figure out how certain important features of an application, crackers might use it to see how they can bypass the license policy and game cheats use reverse engineering, as well to cheat [17].

A. Software Analysis and Vulnerabilities

The major approach to reverse engineering software is analysis. However, this genuine process can be put to malicious intent. One malicious application of analysis is software tampering. Reverse engineering may lead to the discovery of vulnerabilities in the internals of an application. An attacker may therefore explore this vulnerability in the form of software tampering. Another is the malicious deployment of software static or dynamic analysis. Static analysis refers to analysis of software and data when it is not running and dynamic analysis is performed on executing code and involves tracing of data values and control flow. Main and Oorschot (2003) believed that software tampering attacks may be static or dynamic. A static tampering attack modifies code in a non-executing state and the modified code is subsequently run. If a software integrity mechanism is in place, then the integrity-checking mechanism must be defeated for the modified software to execute as desired. A dynamic tampering attack changes values (data or code) in memory during execution. An attack may be developed or tested dynamically, on a separate platform, and then turned into a static attack on a target platform. A typical goal of

tampering attacks is software piracy, or unauthorized duplication of files in violation of a licensing agreement [13]. Other more complex direct access attacks from RE process according to the same authors are: Software differential analysis (SDA), In which case, two or more different versions of an application are compared, to identify which parts have been changed. Crackers who develop copy protection removal tools use SDA to quickly isolate changed protection techniques, updating their tools to allow them to continue working on new releases.

Collusion attacks involve multiple attackers sharing analysis, to leverage not only different skills to reverse engineer a system, but also to pool user-specific data or knowledge of help in defeating security mechanisms. Replay attacks capture program state and later restore it. For example, a user downloading a movie may watch it within three days of pressing play. A backup of the entire machine state is made using a disk imaging tool, once the original digital rights are consumed, the user restores them using the back-up machine state [13].

Major aspects of reverse engineering include disassembly and decompilation, To this end, foundational tools in the cracker's reverse engineering toolkit include: debuggers, disassemblers, decompilers and emulators. Decompilation recovers higher-level program abstractions and semantic structure from binary programs while disassembly reconstructs assembly language instructions from machine code; it may be considered a subset of decompilation, or a step along the way. A disassembler is typically the first tool used in reverse engineering an executable program, whether for legitimate purposes (e.g. automated code optimization) or otherwise. Debuggers [18] trace the program logic and data values during program execution. Breakpoints can be set and code and data modified on the fly, making debuggers valuable tools for uncovering bugs and addressing performance issues, as well as reverse engineering and tampering with applications.

Emulation and spoofing attacks are methods that, rather than tamper directly with an application, exploit an interface or impersonate presumably-trusted system components. Thus emulators and simulators [19] allow crackers to emulate the environment in which an application expects to run. Emulators can be used to store state information, to help replay attacks. They are also used to create virtual drives to bypass copy protection schemes [20].

V. SOFTWARE PROTECTION TECHNIQUES AGAINST RE AND CODE TAMPERING

Several strategies had been proposed and implemented to protect against the use of reverse engineering for malicious purposes. A simple yet very powerful process is envelope protection. According to Chandra (2008) software can be protected within an envelope without making any change to the source code. The software is passed through a special utility, and it comes out encrypted, with the envelope protecting it. Envelope protection, in addition to encryption, also provides anti-debugger strategies to prevent an attacker from attaching a debugger to the program, periodic polling the USB port to see if the right dongle is still present, implementation of several code obfuscation strategies, and provision of different grades of encryption to the binary [17].

Techniques to disrupt the process of static disassembly of programs have recently been explored by [21]. The goal is to make correct disassembly more difficult. Their techniques are complementary and orthogonal to software obfuscation.

However for reverse engineering we advocate techniques that will make the source code more difficult to understand by the attacker i.e tamper resistant software. Suggested are:

A. *Obfuscation*

Software Code Obfuscation is a cracker-centric approach to disrupt cracker's actions by hiding secrets involved in the software systems. Obfuscations transform a program so that it is more complex and difficult to understand, yet is functionally equivalent to the original program. The secrets in a program may include subroutines, algorithms and constant values that are valuable and/or related to system security. There are various types of software obfuscation methods, including control flow obfuscation, inter-module call relation obfuscation, identifier obfuscation, self-modifying code, data obfuscation, etc. [22]. [17] believed that code obfuscation is the simplest (and cheapest) method to deter reverse engineers because it changes function names, alters the sequence of code, and adds noise, without changing the functionality of the code itself.

B. *Software Tokens*

Another common technique is software tokens. This involves shipping a 'license' file along with the software product. This file contains information that the product checks every time it is run; if the file is not present, or the information is wrong, the product exits with a license violation error. The information may include information specific to the installation site [23], such as the hardware network card address.

C. *Tamper Proofing*

Providing tamper resistance may involve making software difficult to modify or tamper using static and dynamic tamper detection approaches such as co-designing and dynamic self-checking; to watch out for integrity violations of any component of a software application or its operating environment.

D. *Code Partitioning*

Code Partitioning is the technique of placing a portion of the software in inaccessible memory. This portion may be just the license-checking part of the application. However, the attacker may find the code within the application (which is in unprotected memory) that invokes the protected license-checking code, and patch around it. To discourage such attempts, it will be necessary to physically protect a more substantial portion of the application [22].

E. *Mutation Analysis*

Mutation Analysis is a method of software testing, which involves modifying program's source code in small ways. This technique has been successfully deployed in analyzing threats in software system [24].

VI. CONCLUSION AND RECOMMENDATIONS

We have presented an unbiased exploration of the technology of software reverse engineering in this paper. It is our position that the technology represents a two-edged-sword - an important tool for maintenance and roundtrip

software engineering and a serious threat to software integrity and authentication and application security through many of its approaches to understanding the inherent structures and functionality of a software system. We therefore recommend that even with a slice of the program, software engineers must be cognizant of the threats from reverse engineering and engineer software systems with credible technical defenses against code-level breaches, while still delivering value to customers.

REFERENCES

[1] Hoglund, G. McGraw, G. (2004). *Exploiting Software How to Break Code*. Addison Wesley. Chapter three, pp 71-145. ISBN: 0-201-78695-8

[2] HP Fortify Software Security Center: *Proactively Eliminate Risk in Software*. Downloaded February 2012 from https://www.fortify.com/downloads2/public/Fortify_360_Datasheet.pdf

[3] Ponemon Institute released findings of the 2010 Annual Study: *U.S. Cost of a Data Breach*. Available at http://www.ponemon.org/data-security

[4] Larry Ponemon (2010). Ponemon Institute LLC *Five Countries: Cost of Data Breach Sponsored by PGP Corporation*. Downloaded April 2012 from http://www.ponemon.org/local/upload/fckjail/generalcontent/18/file/2010%20Global%20CODB.pdf

[5] *Symantec Corp., Internet Security Threat Report, Vol. 16*. Accessed April 1 2012 from https://www4.symantec.com/Vrt/wl?tu_id=ybxO1301703708025310104

[6] Juniper Networks study, May 2011. Downloaded April 2012 from www.**juniper**.net/us/en/local/pdf/whitepapers/2000415-en.pdf

[7] lARP64Tech Software. *Protection Versus Hacking*. Downloaded February 2012 from http://www.larp64.com/index.html

[8] Osuagwu, O. E., Oladipo, O. F. and Yinka-Banjo, C. (2008). Deploying Reverse Software Engineering as tool for Converting Legacy Applications in critical-sensitive systems for Nigerian Industries. In Proceedings of the 22nd National conference and AGM of the Nigeria Computer Society Conference (ENCTDEV 2008). 24-27 June

[9] Reverse engineering from Wikipedia the free encyclopedia. http://en.wikipedia.org/wiki/Reverse_engineering

[10] Klosch, R.R. (1996). Reverse Engineering: Why and how to reverse engineer software, Proceedings of the California Software Symposium (CSS '96), Los Angeles, California.

[11] Rugaber, S. (1994). "Program Comprehension for Reverse Engineering," http://www.cc.gatech.edu/reverse/papers.html, College of Computing, Georgia Institute of Technology, March 9, 1994

[12] Oladipo, O.F. (2010a). Software Reverse Engineering of Legacy Applications. Unpublished Ph.D Thesis. Computer Science Department, Nnamdi Azikiwe University, Awka Nigeria. External Assessment, November, 2009.

[13] Main, A. and Van Oorschot, P.C. (2003). Software Protection and Application Security: Understanding the Battleground. State of the Art and Evolution of Computer Security and Industrial Cryptography, June 2003, Heverlee, Belgium, Springer-Verlag LNCS.

[14] Jean-Marie F. A (2007). Flexible Approach to Visualize Large Software Products Downloaded December 2007 from http://citeseer.ist.psu.edu/140717.html.

[15] Grimm, T. A. (2004). Reverse Engineering is criminal. Downloaded February 2009 from http://www.tagrimm.com

[16] Van Oorschot, P. C. (2003). Revisiting Software Protection. Proceedings of the 6th International Conference on Information Security, ISC 2003, Bristol, UK, pp.1–13, Springer-Verlag LNCS 2851 (2003), Colin Boyd, Wenbo Mao (Eds.).

[17] Chandran, R. (2008). Defend against Reverse Engineering. Palizine Information Security intelligence Magazine, Issue 34, July. Downloaded February 2012 from http://palpapers.plynt.com/

[18] Cifuentes, C., Waddington, T., and Van Emmerik, M. (2001). Computer Security Analysis through Decompilation and High-Level Debugging., Workshop on Decompilation Techniques, pp.375-380, 8th IEEE WCRE (Working Conf. Rev. Eng.), Oct.2001

[19] Magnusson, P. S., Christianson, M., Eskilson, J. (2002). Simics: A full system simulation platform, IEEE Computer vol.35 no.2 (Feb.2002), pp.50-58.

[20] Kennell, R. and Jamieson, L.H. (2003). Establishing the Genuity of Remote Computer Systems, Proceedings of the 12th USENIX Security Symposium (August 2003), pp.295-310.

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

[22] Yamauchi, H., Kanzaki, Y., Monden, A., Nakamura, M. and Matsumoto, K. (2006). Software Obfuscation from Cracker's Viewpoint. Proceedings of the IASTED International Conference Advances in Computer Science and Technology January 23-25, 2006, Puerto Vallarta, Mexico

[23] Joshi, B., S. (1987). Computer Software Security System. United States Patent 4,688,169, 1987.

[24] Thomas, L., Xu, W., Xu, D. (2011). Mutation Analysis of Magento for Evaluating Threat Model-Based Security Testing. Downloaded April 2, 2012 from http://www.dsu.edu/research/ia/documents/[11]-Mutation-Analysis-of-Magento-for-Evaluating-Threat-Model-Based-Security-Testing.pdf

