Introduction to Reverse Engineering concepts for solving UniWA-CTF

Michael Kokkos, Alexander Soloviev

Athens, Greece

Abstract

Software reverse engineering is the field of study that allows to reverse the development and production processes of software and gain a thorough look on the functioning of the program's code. The deconstruction and reverse engineering of software provides insight into the source code of an program. This paper introduces and helps the participants of UniWA-CTF in different concepts of reverse engineering which are necessary to solve the challenges. The purpose of the current paper is to demonstrate basic implementations of software reverse engineering and their usage in different disciplines of computer science.

Keywords: Reverse Engineering, Software Reverse Engineering, Malware Analysis, Binary Analysis

1. Introduction

Reverse Engineering is the method of extracting designs, data and source code from computer software programs or binary files. The concept of reverse engineering existed years before the computers and was performed in various scientific fields.

Software reverse engineering is the method of studying the software inner functioning. It can be achieved by analysing the software program, recognising the design information of a program and finding out the implementation's information of the software.[1] This comes in contrast with the process of software development which is starting with the basic concept of program and the gathering of the requirements. The next step is the process of content analysis, the design process and finally the implementation of the program. This process is called forward engineering.

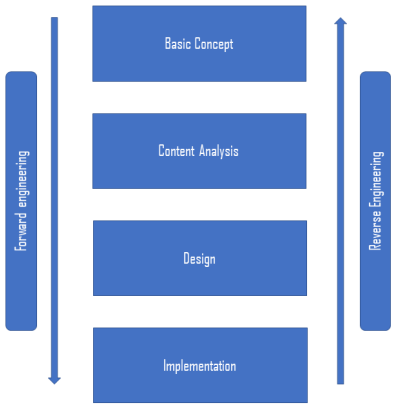

Figure 1: Forward and Reverse Engineering

2. Binary Analysis

Binary analysis is the process of analysing the source code and gather data from binary programs. The main objective of binary analysis is to process and, in some cases, modify the inner information of binary programs. A point of view is to think that binary analysis is synonymous with reverse engineering and disassembly. Disassembly is an important procedure in different types of binary analysis, and reverse engineering may be a similar implementation of binary analysis.[2] However, the field of binary analysis encompasses a lot of different techniques, that can be divided into two categories.

2.1. Static analysis

Static analysis, is the technique of analysing a binary file without having to run it. This approach has many advantages, that is because the engineer can analyse the total binary file at once, and does not need a specific computer processor unit to run this file. For example, an ARM processor binary program can be analysed on a computer with different type of architecture. The drawback is that static analysis does not provide information about the runtime state, which might do the analysis more difficult.[3]

2.2. Dynamic analysis

Dynamic analysis, is the technique of analysing and executing the binary file at the same time. Commonly is less complicated than static analysis because the engineer can get full information of the whole runtime state, as well as the values of variables and the outcomes of conditional branches.[3]

Unfortunately, when the engineer executes the code, dynamic analysis could miss important elements of the program. Both the techniques of static and dynamic analysis have their benefits and their drawbacks. In addition to binary analysis, a lot of binary techniques exist that are used to analyse and modify binary files. For example, binary instrumentation is based on analysis techniques such as disassembly, and it can be used to help binary analysis.[3]

3. Basic Software Reverse Engineering Concepts

Software reverse engineering needs a combination of skills, including a basic understanding of computer and software engineering. Software reverse engineering is used on the recovery of program source code or for re-documentation purposes, this process is needed when the source code is locked or not available.[4] An example of that is the closed-source proprietary software.

3.1. Implementations of Reverse Engineering

Reverse engineering plays an important role in malware analysis, that is because a lot of malicious software is created by reversing programs or operating system key-files. An other example is its use in the field of cryptanalysis, one implementation, between others, is detection of vulnerabilities in symmetric-key cryptography and ciphers.[5]

It is also used as technical obsolescence. Often integrated circuits are designed and built on production lines and in a few years become antiquated. That type of systems using those parts can no longer be maintained, that is because the parts are no longer supported, the way of development a new embedded system is to reverse the existing chip and then to reconstruct it using newer tools.[6]

3.2. Malware Analysis

Malware analysis is the technique of dissecting malware, that allows the analyst to know how it works and also helps in finding out ways to defeat or eliminate it. Several malicious programs are encountered each day, malware analysis is a very helpful skill for anyone who is interested in computer security. Two approaches of malware analysis exist, static and dynamic.[7]

3.2.1. Static Analysis

Static analysis is a technique that allows an engineer to analyse a file without him or her having to execute it. The basic methodology includes, scanning the suspicious file with anti-virus software to detect common malware, by utilizing its malware-signature database. Once the scan has been completed, the next step is to open the file with a hex editor to find out what type of software it is. Then, the analyst has to use the *strings* utility, to inspect for ASCII and Unicode characters. That utility searches into the suspicious file for protocols, ports, IP addresses and other important information about the functionality of the malware. Final step is disassemble the file. The files can be decompiled by using debuggers or disassemblers.[8]

3.2.2. Dynamic Analysis

Dynamic analysis must complement static analysis, so that the analyst can have proper results. This type of analysis uses a debugger to examine the internal execution state of a malicious file. For example, debuggers can show the values of memory addresses as they change during the execution of a program and allow to show the value of every memory location, register, and argument to every function.[7] Dynamic analysis needs to be performed in a safe environment, preferably that of a virtual machine and needs to be ensured that the virtual machine networking is not connected to any networks.[8]

4. Conclusions

Reverse engineering is a very important skill in computer security because many of the challenges encountered in a real work environment can be approached by utilizing reverse engineering-related skills. The UniWA-CTF challenges have been designed with the purpose to introduce basic concepts and assist them to build a foundational knowledge on the topic.

References

[1] Z. Z. Qingjun Yuan, SiqiLu, X. Chen, Researching on aes algorithm based on software reverse engineering, Advances in Intelligent Systems and Computing (2020).

[2] A. Djoudi, S. Bardin, Binsec: Binary code analysis with low-level regions, International Conference on Tools and Algorithms for the Construction and Analysis of Systems (April 2015) 212–217.

[3] D. Andriesse, Practical Binary Analysis, 1st ed., William Pollock, 2019.

[4] E. Eilam, Reversing: Secrets of Reverse Engineering, 1st ed., Wiley Publishing, Inc., 2005.

[5] B. Dang, A. Gazet, Practical Reverse Engineering: x86, x64, ARM, Windows® Kernel, Reversing Tools, and Obfuscation, 1st ed., John Wiley and Sons, Inc., 2014.

[6] B. Bartels, U. Ermel, P. Sandborn, M. Pecht, Strategies to the Prediction, Mitigation and Management of Product Obsolescence, Wiley Series in Systems Engineering and Management, Wiley, 2012.

[7] M. Sikorski, A. Honig, Practical Malware Analysis, 1st ed., William Pollock, 2012.

[8] A. Ray, D. A. Nath, Introduction to malware and malware analysis: A brief overview, International Journal of Advance Research in Computer Science and Management Studies, Volume 4 (October 2016) 22–30.