Assignment Cover Sheet



* The information on this coversheet will be included in Turnitin’s similarity analysis; however, your lecturers are aware of this and will disregard it.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Student Details | | | | | | | | | | | |
|  | | | | | | | | | | | |
| Student Number | 1 | 0 | 4 | 9 | 8 | 7 | 4 | | 3 |  | |
|  | | | | | | | | | | | |
| Family Name | David Selvam | | | | | | | Given Name | | | Visahl Samson |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| UnitDetails | | | | | | | | | | | |
|  | | | | | | | | | | | |
| UnitCode | CSI2107 | | | UnitTitle | | Software Reverse Engineering | | | | | |
| Name of Lecturer | | | Kang Leng | | | | Due Date | | | 14 October 2019 | |
| Topic of Assignment | | | Review of Reverse Engineering Method | | Group or Tutorial(if applicable) | | |  | | | |
| Course | | Bachelors of Science in Cyber Security (FT) | | | | | | | Campus | | Singapore |
| I certify that the attached assignment is my own work and that any material drawn from other sources has been acknowledged. This work has not previously been submitted for assessment in any other unit or course.  Copyrighting assignments remains my property. I grant permission to the University to make copies of assignments for assessment, review and/or record keeping purposes. I note that the University reserves the right to check my assignment for plagiarism .Should there production of all or part of an assignment be required by the University for any purpose other than those mentioned above, appropriate authorisation will be sought from me on the relevant form. | | | | | | | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Manual Submission | | | | |
| If handing in an assignment in a paper or other physical form,sign here to indicate that you have read this form, filled it in completely and that you certify as above. | | | | |
| Signature |  | Date |  | ***Office Use Only*** |
| ElectronicSubmission | | | |
| **OR,**ifsubmitting thispaper electronically as per instructions for the unit, place an ‘X’ in the box below to indicate that you have read this form and filled it in completely and that you certify as above. Please include this page with your submission. Any responses to this submission will be sent to your ECU email address. | | | |
| Agreement | X☐select checkbox | Date | 21 June 2019 |
| For procedures and penalties on late assignments please refer to the [University Admission, Enrolment and Academic Progress](http://intranet.ecu.edu.au/__data/assets/pdf_file/0005/378320/Admission-Enrolment-and-Academic-Rules.pdf)  [Rules](http://intranet.ecu.edu.au/__data/assets/pdf_file/0005/378320/Admission-Enrolment-and-Academic-Rules.pdf)- rule 24, and the ECU Course and Unit Delivery and Assessment Policy | | | | |

Contents

[Assignment Cover Sheet 1](file:///D:\PSB\SEMESTER%202\SRE\Davidselvam_Visahlsamson_CSI2104_Assignment_1.docx#_Toc21933915)

[Introduction 3](#_Toc21933916)

[Tools selected to compare 3](#_Toc21933917)

[Radare2 3](#_Toc21933918)

[Usage in Software reverse engineering 4](#_Toc21933919)

[Features 4](#_Toc21933920)

[Advantages 5](#_Toc21933921)

[Disadvantages 6](#_Toc21933922)

[Upcoming features 6](#_Toc21933923)

[Ghidra 6](#_Toc21933924)

[Usage in Software reverse engineering 6](#_Toc21933925)

[Features 6](#_Toc21933926)

[Advantages 7](#_Toc21933927)

[Disadvantages 8](#_Toc21933928)

[Upcoming features 9](#_Toc21933929)

[Immunity Debugger 9](#_Toc21933930)

[Role in software reverse engineering 9](#_Toc21933931)

[Features 9](#_Toc21933932)

[Advantages 10](#_Toc21933933)

[Disadvantages 10](#_Toc21933934)

[Reverse engineering methodologies 11](#_Toc21933935)

[Static analysis 11](#_Toc21933936)

[Dynamic Analysis 11](#_Toc21933937)

[Automation 12](#_Toc21933938)

[Code Obfuscation 12](#_Toc21933939)

[Anti-dissembler 12](#_Toc21933940)

[Anti-debugging 13](#_Toc21933941)

[Anti-VM 14](#_Toc21933942)

[Conclusion 14](#_Toc21933943)

[References 15](#_Toc21933944)

# Introduction

In the field of software reverse engineering, the tools and software are plays a vivacious role. The main purpose of these tools and software is to make the work easy and efficient. The tools and software can be sorted into several categories such as debuggers, disassemblers, Hex Editor, PE analyzer and so on. Some of the tools can be used in several phases of software reverse engineering while some of them can fit into a specific phase. This report contains a comparison of 3 tools/software used in software reverse engineering in terms of features, advantages, disadvantages and. limitation. Also, this report covers methodologies used in software reverse engineering such as static/dynamic analysis, automated, code obfuscation.

# Tools selected to compare

The tools selected to compare are,

* Radare2
* Ghidra
* Immune debugger

These tools are selected because each tool covers different phases of software reverse engineering.

# Radare2

Radare2 is a well-known framework used to perform software reverse engineering ("radare", n.d.). Radare2 can be also used in malware analysis, firmware analysis and, binary file analysis. Also, apart from software reverse engineering radare2 can be utilized in the field of computer forensics and data carving. Likewise, radare2 can be scripted to run automatically using major scripting languages like Python, JavaScript, Ruby, and Perl. At an advanced level, radare2 can be utilized to perform software exploitation ("radare", n.d.).

Radare is the acronym of Raw Data Recovery ("radare", 2019). It supports performing analysis on file systems to recover useful information. Also, Radare2 can be used to learn how malware/software work, troubleshoot software. The usage of Radare2 is coming under LGPLv3 license.

## Usage in Software reverse engineering

In software reverse engineering Radare2 can be used as Disassembler and used to perform static analysis on the target/specified file system.

The reason why I have selected Radare2 for this comparison is because of its multipurpose usage.

## Features

Radare2 has several features that the majority of other disassemblers lack. According to Radare2 ("Radare2 Comparison", 2018) The highlighted features of Radare2 are ,

|  |  |
| --- | --- |
| **Architecture** | Arn, arm64, avr, dalvik, java, mips, mips64, ppc, x86, x86\_64 |
| **Scripting support** | Python, JavaScript, Ruby, Perl, Go, Lua, Vala, Guils, Sh |
| **Debugger support** | Gdb, native, windbg, lldb, bfdbg, rap, mach, bochs, qnx, r2k |
| **Functionality** | Assembler, Bindings/API, C++ Classes, Classes JAVA/DEX, Comments/Renaming, Cross References, Custom structures, definition, De Bruijn pattern, Diffing, Dwarf, Emulation (via intermediate language), Encrypt/Decrypt support,  Free (as in freedom), Function prototype / Arguments, Graphs, Hash Support, Heap Analyser, Intermediate Language support,  Mitigations detection,  Open bugtracker, Open database format, Patching, PDB, ROP-gadget finder, Saving and exporting project,  Shellcode Compiler,  Signature recognition, Swift, Trace, Types |
| **Filesystems support** | Ext2, vfat, ntfs, etc..  Partition type – gpt, msdos,etc… |
| **Platform** | GNU/Linux, Android, \*BSD, OSX, iPhoneOS, Windows (32,64) and Solaris |

**Dependencies and Packet Managers**

* Radare2 doesn’t require any type of specific dependencies to be built ("Reverse Engineering Framework: radare2", 2018). Also, we can easily obtain a working toolchain such as gcc, clang from Radare2. The framework Radare2 equipped with its own packet manager which is known as – r2pm. Pack manager for Radare2 is available in Github for ease of use.

## Advantages

Major advantages of Radare2 includes the following ("Radare2", 2019),

**Features and ratings**

* It’s one of the primary advantages of Radare2. Radare2’s features enable it to be not only for software reverse engineering but also enable it to use in forensics and data carving domains. Due to its features and functionality, Radare2 holds rank 895 with 10784 star status in Github ("radare/radare2 - Gitstar Ranking", 2019).
* Also, Radare2’s unique features make to work easily in a simplified form.

**Open source**

* The Primary benefit of Radare2 is it’s free and available open source. So, all its features, and functionalities can be used for free without the need of paying. Also, the source code for this tool available online for free.

**Community**

Since Radare2 available for free it has the biggest contributors’ community which helps all the way along using Radare2. Also, dedicated forums are available for any queries regarding Radare2 functionality and operations. Likewise, so far, 5991 issues have been closed in Radare2’s Github repository ("radareorg/radare2", 2019)

## Disadvantages

**Usability issues**

* Radare2’s CLI is quite an unease while doing the work. Also, in many instances, Radare2 doesn’t display feedback or error messages (Giovanni, 2017).

**Loss of control**

* In Radare2 it’s possible that the user might lose control if the program has highly complex logic it’s because of it’s CLI nature.

## Upcoming features

* Currently, radare2 has several project ideas ("RSoC 2019", 2019) such as Console Interface Improvements, Radare2 commands syntax parser, Type Analysis Improvements, CPU/Platform profiles, Handle EXE/DLL as FAT binaries, Proper Windows platform support, Radiff2 improvements, Real-time collaboration platform

# Ghidra

* Ghidra is a well-known software reverse engineering framework created and maintained by the National Security Agency (NSA) Research Directorate. Ghidra was released in March 2019 ("Ghidra", 2019). It is an open-source tool that enables the information security researchers/malware analysts/software reverse engineering researchers to disassemble the code.
* NSA developers hold a GitHub page for all the technical support, questions and enhancement requests. Since its release, Ghidra has increasing the contributor community (The Cylance, 2019).

## Usage in Software reverse engineering

* Ghidra can be used as a disassembler in software reverse engineering apart from it Ghidra is also capable of performing decompilation, inspecting symbols and references. Scripting variable identification and so on.

## Features

Ghidra has several unique features that help it to become one of the must-try tool, the features include (Elias, 2019),

|  |  |
| --- | --- |
| **Architecture** | 6502, 68000, 6805, 8051, 8085, AARCH64, ARM, Atmel, CR16, DATA, JVM, MIPS, PA-RISC, PIC, PowerPC, Sparc, TI\_MSP430, Toy, x86, Z80 |
| **Scripting** | Can support both automated and user-interactive instruction sets. Also, users can develop their own plugins for Ghidra based on their requirments |
| **Functionality** | Assembler, Bindings/API, Classes JAVA/DEX, Comments/Renaming, Cross References, Custom structures definition, Diffing, Dwarf, Open-source, Function prototype / Arguments,  Graphs, Hash Support, Intermediate Language support, Open bugtracker, Patching, Saving and exporting project, ("Radare2 Comparison", 2018) |
| **Plugin support** | Cryptanalysis, interaction with OllyDbg and the Ghidra Debugger. (MALWAREHUNTERS, 2019) |
| **File format support** | Avr, dex, elf, elf64, fatmach0, mach0, mach064, mz, pe, pe64, zip (jar, apk, ipa), ihex |
| **Supported platforms** | Windows (64 bit), Linux, OS X (MALWAREHUNTERS, 2019) |

## Advantages

**Open source**

* Though Ghidra is developed by NSA it made available open source. Users can either visit Ghidra website to download or can clone it from its Github repository.

**Interactive GUI**

Ghidra comes with an excellent GUI feature which makes it looks elegant. Also, the GUI is quite good to work with.

**Workflow**

* Ghidra allows the user to load multiple binaries into a single project without causing any mess (Polynomial, 2019). Also, it’s disassemblers consist of in-built data flow analysis which will show from where the data come whenever the user selects a variable or register.
* Ghidra comes with collaborative options for projects that related to disassembly/decompiler which is unique and many of the disassemblers lack this feature.
* Also, Ghidra does perform undo operation which cannot be found on many other tools.

**Extensions**

Ghidra is equipped with few extensions such as GhidraDev Eclipse plugin for a pre-existing Eclipse installation, Ghidra extension, IDA Pro plugins/loaders for transferring items with Ghidra. Though these extensions come in-built the user must install it themselves (MALWAREHUNTERS, 2019).

## Disadvantages

**No Debugger**

It’s one of the primary downsides of Ghidra. Though, it has many unique features it has no debugger in-built or debugger support which is quite inconvenient.

**Way of installation**

Installing Ghidra in Windows is quite weird. Because installing Ghidra doesn’t modify any windows registry values so if we delete the Ghidra folder the tool can be uninstalled.

**File format support**

Though Ghidra supports various file formats the number of formats it supports is kind of less than other disassemblers.

## Upcoming features

* According to reports (Vavra, 2019) It’s expected that in separate version 9.1 rollout Ghidra will start an android executable format. So, after this rollout Ghidra can be used to analyze the android malware that uses Dalvik executable format (DEX).
* Also, it’s expected that the Debugger functionality will be added to Ghidra. If it is added, then Ghidra can be used not only for static analysis but also for dynamic analysis purposes as well.

# Immunity Debugger

Immunity debugger is a powerful debugging tool mainly designed to write exploits, malware analysis, and software reverse engineering binary file analysis. It designed with a compact user interface with various functions such as function graphing. Also, it’s the first-ever tool in the industry with a heap analysis feature along with Python API to provide easy extensibility (Immunity, 2019).

Since code obfuscation can be done in the anti-debugging level and immunity debugger has an automated scripting feature it’s wise to use this tool to reduce the time to crack the anti-debugging obfuscation (Immunity, 2019). This is one of the reasons to add immunity debugger in this list of comparisons.

## Role in software reverse engineering

In software reverse engineering domain Immunity debugger can be used for dynamic analysis purposes.

## Features

Immunity debugger’s functions are specifically designed for the information security domain. It’s stated (Immunity, 2019) that using immunity debugger for exploit development can reduce the development time up to 50%. Immunity debugger is lightweight and can perform fast to prevent corruption when analyzing complex logic. It also has connectivity to many exploit development tools and fuzzers which make it a unique debugger in industry.

## Advantages

**Interface**

* Immunity debugger has both GUI and CLI feature. The CLI can always be found at the bottom of the GUI. Also, the GUI is quite simple and straight forward.

**Extendibility**

* In immunity, the debugger has an easy extendibility feature using Python. Also, in the immunity debugger, the python command can directly run from the command box.

**Remote debugging**

* Immunity debugger comes with a plugin that allows connecting to the server and remote debugging. This option can be found in the command line menu.

**Python Scripting**

* Users can load and modify the python scripts even during the runtimes without causing any interference/issues.
* Since the immunity debugger’s python scripting engine is fully integrated the user can easily monitor the variable size and usage which can be utilized for the automated debugging process.

**Automated tasks**

* Because of the Immunity debugger’s feature of Python API, the user can run scripts to automate the analysis processes which makes the work easier.

## Disadvantages

**Architecture**

Immunity debugger is a 32-bit debugger (Alexis, 2018) so it does only support 32-bit based executables/programs which is the main drawback of it.

**Lack of community**

Unlike any other tools’ immunity debugger doesn’t have a significant contributor community so if any issue raises it’s quite complicated to solve.

# Reverse engineering methodologies

When it comes to software reverse engineering there is step by step procedures that every analyst should follow. The procedure includes static and dynamic analysis, doing it either manual or automated and then breaking the code obfuscation. The detailed explanation of these steps is given as follow (Monnappa, 2018),

## Static analysis

* Static analysis is basically analyzing the executable/program without running it. It involves finding the patterns. Static analysis can be categorized into two levels they are Basic static analysis, another one is Advanced static analysis. It basically works in a non-runtime environment.
* The basic static analysis could be very easy but at the same time it might be ineffective against sophisticated executable. Advanced static analysis involves usage of disassemblers like Radare2 to load the executable to reverse engineer the binary and to get assembly language source code.
* The Static analysis consists of few techniques such as determining file type, finding strings encoded into the binary files, looking for code obfuscation, hashing.
* In malware analysis, the initial analysis starts from using the Virus total search engine or any other Anti-virus search engine to find out possible information about it. Then the process starts from generating the hash, extracting the string and so on.

## Dynamic Analysis

* Dynamic analysis involves analyzing the executable/program by executing/running it. It basically involves observing the behavior of the executable/program. Dynamic analysis can be performed in the runtime environment. The Dynamic analysis helps could be very useful to find out the behavior that helps us to understand how the executable/program works and what are all the changes can be done in the source code to change the outcome.
* Dynamic analysis techniques involve the activities that focus on the file system, registry, network, process and system calls.
* In malware analysis dynamic analysis plays a vital role without it it’s not possible to do the analysis. Also, dynamic analysis in malware analysis will take place in a fully isolated lab environment that uses a sandbox like Cuckoo sandbox or a dedicated Virtual machine like flare VM in order to prevent any issues. Likewise, it helps to eliminate the malware infection and to produce effective signatures.

## Automation

* Software reverse engineering can be done in two ways wither by Manual or by automated. In recent days automated is growing gradually and many products have been developed by vendors in order to support the automated process. The automated process can be done either by using separate tools or by using specific frameworks. The most common way of using automation in software reverse engineering is by executing scripts.
* The Automation process can be used in both static and dynamic analysis of the executable/program. Especially while unpacking the binary the automation tool can be utilized better.
* Tools like Ghidra, Binary ninja comes with this automation feature which makes the workflow better. Apart from its tools like TitanMist (Monnappa, 2018) which consists of different packer signatures and unpacking scripts are available to use.

## Code Obfuscation

* Code obfuscation is generally involving the practices of hiding the code in order to prevent the application/program from being reverse-engineered.
* In malware analysis, the malware authors will use a different kinds of obfuscation techniques to fur the information and helps to modify the malicious content to make the analysis process to take time and difficult for the analyst. The malware authors usually use encryption/encoding practices to hide the information from security tools/products such as anti-virus products/services and evade them.
* In general, code obfuscation can be done in 3 ways they are, Anti-dissembler, Anti-debugger, Anti-VM (Infosec, 2018).

### Anti-dissembler

In this method, the author can include a code that causes the disassembler tools to produce wrong program listing which will cause a delay in the reverse engineering process. Some of the techniques used in Anti-dissembler method are (Infosec, 2018),

**API Obfuscation**

Using this method will change the name of identifies such as method, class and field names to some random names so the analyst doesn’t know the functionality of the code and will take some to solve this issue.

**Assembly/ Opcode code obfuscation**

This method will change the code instructions to hard to read format that will cause difficulty in disassembly of the program.

**Junk code**

As the same describes the author will include a random chunk of junk code which does nothing. The purpose of this method is to confuse the analyst with unwanted junk code and make it tough to analysis.

**Control flow graph (CFG) flattening**

This step involves flattens the control flow of the function. What it does basically is to break all the nest, if-statements and hide them into a hefty switch statement. This method will make the analyst to ignore/misunderstand the original code.

### Anti-debugging

It’s an anti-analysis method that helps the program to check whether it’s being debugged. The techniques used in this type are aimed to slow down the reverse engineering analyst. Common anti-debugging methods are (Infosec, 2018),

**Windows API**

This is one of the well-known anti-debugging methods in use. Windows API basically produces numerous functions that will help the program/malware to find out whether they’re being debugged or not. The common function that produced by Windows API which helps the malware/program is IsDebuggerPresent. In this function, it checks for a definite flag that will return as zero when not debugged and return as 1 if debugged.

**Timing defense**

RDTSC is also known as Ream Time Stamp Counter is a well-known example in this case. Since the debugger takes time to process every instruction it will cause the system slow. So the malware will read the time stamp of the system and will cause an error if it finds any irregularities in processes.

**TLS callback**

In general, all the debuggers are beginning the process with the program’s entry point. In Windows, this TLS callback can be practiced to execute the code before the execution of the main application. This allows the programmer/malware author to execute the malicious code before the debugger program to check with a traditional entry point.

TLS callback allows the program/malware to disable the debugger before it starts to analyze the program.

### Anti-VM

In the Anti-VM method (Wong, 2018), the programmer/malware author includes a code that exits the program if it is running in a virtualized environment. Basically, the code written by the authors will look for the Virtualized environment artifacts that can be found under the registry or when running a service. So, once it finds out the processes are running the program will automatically exit.

Though there are different types of code obfuscation methods are in use all these methods can be evaded with the right choice of tools and techniques.

# Conclusion

Software reverse engineering is a vast environment where we need to understand several things. Activities in software reverse engineering aim to understand the system and try to get the information from the system. Also, software reverse engineering methodologies are highly required in the subset of Malware analysis. Because in this digital era the intruders come with new and hard to crack coded malware/program to perform offensive operations so, analytics knowledge /methods about reverse engineering are required to perform countermeasures on them.

# References

radare. Retrieved from, <https://rada.re/r/>

radare. (2019). Retrieved from, <https://www.radare.org/r/history.html>

Radare2 Comparison. (2018). Retrieved from, <https://rada.re/r/cmp.htm>

Reverse Engineering Framework: radare2. (2018). Retrieved from, <https://n0where.net/reverse-engineering-framework-radare2>

Radare2. (2019). Retrieved from, <https://linuxsecurity.expert/tools/radare2/>

radare/radare2 - Gitstar Ranking. (2019). Retrieved from, <https://gitstar-ranking.com/radare/radare2>

radareorg/radare2. (2019). Retrieved from, <https://github.com/radareorg/radare2/issues>

Giovanni, L. (2017). Introduction to Binary Reversing using Radare 2. Italy.

RSoC 2019. (2019). Retrieved from, <https://www.radare.org/rsoc/2019/ideas.html>

Ghidra. (2019). Retrieved from, <https://www.nsa.gov/resources/everyone/ghidra/>

The Cylance, T. (2019). An Introduction To Code Analysis With Ghidra. Retrieved from, <https://threatvector.cylance.com/en_us/home/an-introduction-to-code-analysis-with-ghidra.html>

Elias. (2019). Ghidra: A quick overview for the curious. Retrieved from, <http://0xeb.net/2019/03/ghidra-a-quick-overview/>

MALWAREHUNTERS. (2019). Ghidra: All you Need to Know about the NSA's Reverse Engineering Tool - MalwareHunters.co.uk. Retrieved from, <https://malwarehunters.co.uk/ghidra-all-you-need-to-know/>

Polynomial. (2019). What is the difference between Ghidra and Ida?. Retrieved from, <https://security.stackexchange.com/questions/204876/what-is-the-difference-between-ghidra-and-ida>

Vavra, S. (2019). NSA's reverse-engineering malware tool, Ghidra, to get new features to save time, boost accuracy - CyberScoop. Retrieved from, <https://www.cyberscoop.com/ghidra-nsa-new-version-black-hat-2019/>

Immunity. (2019). Immunity Debugger. Retrieved from, <https://www.immunityinc.com/products/debugger/>

Alexis. (2018). Immunity Debugger Basics - HackerSploit. Retrieved from, <https://hsploit.com/immunity-debugger-basics/>

Monnappa, K. (2018). Learning Malware Analysis (1st ed.). Birmingham: Packt Publishing Ltd.

Infosec. (2018). Anti-disassembly, Anti-debugging and Anti-VM. Retrieved from, <https://resources.infosecinstitute.com/category/certifications-training/malware-analysis-reverse-engineering/anti-disassembly-anti-debugging-anti-virtual-machine/#gref>

Wong, R. (2018). Mastering Reverse Engineering (1st ed.). [S.l.]: Packt Publishing.