International Journal on Perceptive and Cognitive Computing (IJPCC) Vol 9, Issue 2 (2023)
https://doi.org/10.31436/ijpcc.v9i2.397

New demand on assembly language proficiency in performing binary reverse engineering tasks

Khairol Amin bin Mohd Salleh
HeiTech Padu Berhad, Application Security Department, 47600, Subang Jaya, Malaysia

*Corresponding author: khairolms@heitech.com.my

(Received: 31st March 2023; Accepted: 20th May 2023; Published on-line: 28th July 2023)

Abstract— Cybersecurity encompasses a wide field of disciplines and as cyber threat landscape changes, there is a need for tools, techniques and skills to provide safe and secure internet environment. The cyber space industry introduced new roles for reverse engineers, malware analysts, digital forensic experts, exploit engineers, etc which demand the new skill set, and in this context, the proficiency in assembly language programming is highly essential. This paper presents an observation on the training programme for software developers from a software integration company to attain the skills of reverse engineers, application penetration testers as well as application security analysts. In preparing for the new generation of reverse engineers and other new roles that are related to cybersecurity, it would be a good step if assembly language could be taught as a separate programming subject, and it would be highly recommended for higher education institutions to collaborate with the industry to undertake co-teaching in supporting the new roles within the realm of cybersecurity. It has been well observed that being endowed with a working knowledge of developing application from scratch using assembly language would offer a foundation to be a good binary reverse engineer.

Keywords— Reverse Engineering, Assembly Language, Debugger, Static Analysis, Dynamic Analysis.

I. INTRODUCTION

In the modern connected cyber world, cybersecurity is one of the important fields in providing safe and secure internet environment as malware and malicious code are still on a widespread. The cyber space industry introduced new roles for reverse engineers, malware analysts, digital forensic experts, exploit engineers, etc which demand the new skill set, and in this context, an insight on assembly language proficiency in performing binary reverse engineering tasks would be unravelled.

II. LITERATURE REVIEW

There are cybersecurity training houses such as SANS, Kaspersky, Mandiant, Koenig, to name a few, that offer certification programmes in reverse engineering especially in analysing malware. These intensive professional programmes are usually for experienced participants and they include teaching the theoretical aspects of the assembly language, without the inclusion of practical programming sessions.

A typical malware analyst would require at least four to five years of working experience in programming x86 assembly language. However, developers with four years of experience in assembly language programming, especially in x86/x64 architecture, may not be easy to come by, as most of the modern-day applications are developed using high level languages, such as Java, PhP, .NET, Python and the like. To a reverse engineer who needs the skill to dissect a binary file for vulnerability detection, the exposure and experience in developing an application using assembly language is highly required.

In another scenario, assembly language programming is not offered as a single or separate subject under the constantly evolving nature of curriculums in computer science departments worldwide [9], and it often means that less time is spent on this low-level programming language [5]. Assembly or machine language programming is treated as a topic within the computer architecture and organization subject, which entails 25 percent of the said subject[2].

The myths that assembly language is dinosaur language is not true. TIOBE [12] index reported that assembly language was ranked number 9 out of 100 languages, in the TIOBE top 10 popular language index as depicted in Table 1. There is an increase in the usage of assembly language for the year 2023. Assembly language is still the best language to perform binary reverse engineering on malware analysis [6].

Kaspersky mentioned that reverse engineering is the most highly demanded skill among the information security specialists [8]. Assembly language skill is the vital skill to perform binary reverse engineering executable file [6].

TABLE 1
TIOBE TOP 10 PROGRAMMING LANGUAGES

Programming Language	2023	2018	2013	2008	2003	1998
Python	1	4	8	7	13	25
C	2	2	1	2	2	1
Java	3	1	2	1	1	17
C++	4	3	4	4	3	2
C#	5	5	5	8	9	-
Visual Basic	6	15	-	-	-	-
JavaScript	7	7	11	9	8	21
SQL	8	251	-	-	7	-
Assembly language	9	13	-	-	-	-
PHP	10	8	6	5	6	-
Objective-C	18	18	3	45	52	-

III. METHOD

Envisaging the increasing demand for binary reverse engineers and that assembly language is vital for this profession, the Assembly Language for Reverse Engineers programme has been introduced with a comprehensive structure of programming and reversing lessons for a duration of four days, as depicted in Table 2. This has been crafted for five batches of Senior and Junior Software Developers, from the Product Development Department in HeiTech Padu Berhad, and in this respect, they are the participants of this programme.

Introduction to x86 assembly language was the kick-start for the participants on the first day and thereupon they journeyed into their practical programming session on Win32 assembly language on the second day. Static analysis was introduced on the third day, and on the last day, the participants focussed on dynamic analysis for which they worked on real binary files that have been downloaded from crackmes.one website.

The learning experience for the participants of the Assembly Language for Reverse Engineers Programme was in a lab-based environment with software tools and training kit that entailed the programme content for every participant. Participatory observation in the capacity of a programme facilitator took place throughout the training programme.

A. Tools and Software

The easiest approach to teaching assembly language was by using less complicated operating system such as Microsoft DOS (Disk Operating System). We used vDOS (Virtual DOS) that has been created by JHM Schaars [10] to execute Debug.com (written by Paul Vojta [13]) and Netwide Assembler [11].

TABLE 2
THE PROGRAMME STRUCTURE

Day	Topic
Day 1	Introduction to x86 Assembly Language x86 Architecture DOS internal and services (INT 21) using DEBUG command in virtual DOS (vDOS) x86 command instructions data transfer, arithmetic & logic instructions flags, program control & subroutines using Netwide Assembler (NASM 16bit) in vDOS
Day 2	Programming in Win32 Assembly Language IA-32 architecture & assembly language Programming using NASM in Windows Windows System Call - Win32 API Porting 16bit to 32bit Assembly program
Day 3	Static and Dynamic Analysis Dissecting Portable Executable (PE) CFF Explorer Static Analysis using IDA-Pro (Freeware) Dynamic Analysis -Dissecting Portable Executable (PE) file using WinDbg Preview
Day 4	Reversing, by-passing and patching binary files. C and C++ structure – Heap, Stack, Calling convention Reversing, by-passing and patching using x64dbg

Participants have been taken through to understand the intel x86 16-bit instruction set to develop a simple application program, and by using the debug.com program they performed native debugging of binary .com files. Participants were then exposed to the basic DOS operating system call, to understand how an application could make use of the operating system services to perform a specific task, such as displaying data using standard output.

After familiarizing with x86 16-bit assembly language programming in vDOS, participants were exposed to the use of the 32/64bit Netwide Assembler (NASM) in Windows environment. Basic Windows system call has been taught unto the participants to provide the same functionalities that are used in DOS system call, on standard input output function.

During static analysis, IDA Free 7.7 [4] program was used to perform the binary code analysis. CFF Explorer [7] was used to dissect and patch Windows Portable Executable files. WinDBG Preview [14] program was used to perform dynamic analysis of windows binary files. The actual patching and by-passing of binary files was done using xdbg64 software.

B. Programme Content

In addressing this new demand on assembly language proficiency in performing binary reverse engineering tasks, the programme content has been designed for the workplace learners to:

• understand the x86 and IA-32/Win32 Architecture and assembly language.

• understand the structure of executable file in Windows (PE – Portable Executable).

• learn about the tools that are used in dissecting binary file.

• understand the techniques of debugging, patching and by-passing machine code.

• be able to understand the application logic from the binary file analysis.

Participants of this four-day programme are required to have the pre-requisite knowledge on any programming language, and in this situation, understanding variables, data types, loops, if-else conditions, simple mathematical operations and string manipulations would enable them to close the learning curve in respect of programming competency in assembly language.

Essentially, this programme provides a focus on teaching 16-bit and 32-bit programming to reduce the complexity of having to deal with 64-bit numbers. Understanding 16-bit number is much easier to learn and visualize. It is quite easy to understand 32-bit programming after learning from the 16-bit experience, and the existing 16-bit instruction can be used in 32-bit programming. Saving the registers into the stack using the POP instruction is quite simple in 16-bit and 32-bit, as compared to the 64-bit programming.

C. Observational Analysis

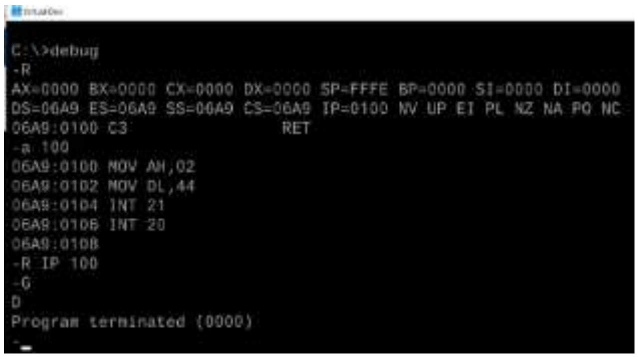
The programme has been conducted for five times within a duration of two years. At the end of every programme, questionnaires were issued as a feedback mechanism from the participants. In addition, an evaluation process has been undertaken on the last two days of each programme, for which individual assignment has been given to gauge their level of understanding in respect of the lessons that they have journeyed through in this programme. Observation on the Assembly Language for Reverse Engineers Programme could well be elucidated in the following manner:

1. Day 1 – Introduction to x86 assembly language

At the onset, participants have been introduced to the basic x86 architecture, with the inclusion of the 16-bit general purpose registers such as ax, bx, cx and dx, special purpose registers, like ip, bp, sp, si, di and segment registers, like ds, ss, cs and es. Before using the debug.com, participants were given a recapitulation briefing on hexadecimal, binary number, and ASCII (American Standard Coding version II).

Participants were guided on the use of the basic system call that has been provided by DOS operating system via interrupt 21 (INT 21h). The basic call included the display of the OS version, print string and print a single character.

Participants undertook to write a simple program using debug.com to display single character and string and save file to .COM extension without using any compiler, as illustrated in Fig. 1.



```
C:\>debug
-R
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06A9 ES=06A9 SS=06A9 CS=06A9 IP=0100 NV UP EI PL NZ NA PO NC
06A9:0100 C3                RET
-a 100
06A9:0100 MOV AH,02
06A9:0102 MOV DL,44
06A9:0104 INT 21
06A9:0106 INT 20
06A9:0108
-R IP 100
-G
D
Program terminated (0000)
-
```

Fig. 1 Running debug.com on vDOS

The x86 instructions have been divided into two parts. The first part encompassed data transfer instructions especially the MOV (move) command and the type of memory addressing, followed by simple arithmetic instructions such as addition, subtraction, multiplication and division. Bit wise logic instructions like OR, AND, NOT and XOR (Exclusive OR) have been addressed in this part.

The second part of the x86 included the usage of x86 flag registers and how each flag register changes; the different types of jump instructions, as well as the calling and writing subroutines.

At the end of Day 1, participants have gained an appreciation of the basic structure in the assembly language program. The basic NASM (Netwide Assembler) compiler which includes labelling, assembler directives, as well as declaration of memory section with its data, code and stack segment have been addressed as part and parcel of the lesson. Finally, participants were given the experience in writing their first assembly program using selected Windows editor with the use of Notepad or Notepad++ and NASM compiler.

In respect of the Day 1 encapsulation, the exposure to x86 16-bit architecture which includes basic x86 instructions, enabled the participants to create a program named disphex.asm to display hexadecimal number as an ASCII character. The disphex.asm program uses string and bit manipulation with the aid of DOS system call to display the printable hexadecimal number.

2. Day 2 - Programming in Win32 Assembly Language

Win32 architecture which includes the 32-bit register and the technique of using Windows system service via stdcall (standard call) have been presented as the highlight on Day 2. A sample program to display "Hello world" string was used to introduce Win32 program structure and method of compiling 'console' program in Windows. In ensuring that participants have been endowed with an understanding of the 32-bit programming knowledge, the 16-bit disphex.asm that was written for vDOS environment was ported to the Windows environment.

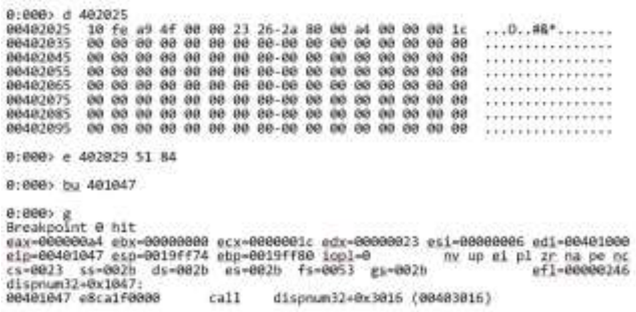
The objective of Day 2 was to provide an understanding of the 32-bit architecture with the expansion of registers from 16-bit to 32-bit length, together with the technique of using Windows System call using stdcall and the required registers, in comparison with using DOS system call via INT 21h and 32-bit memory addressing. Participants have been able to observe the changes involved in porting 16-bit DOS program to 32-bit Windows program.

3. Day 3 - Static Analysis

Participants were taken through static analysis on Day 3, for which they were given an understanding on how Address Space Layout Randomization (ASLR) could be used to protect hackers from hacking the fix loading address of an executable program. In order to perform reverse engineering on an executable file, having a fix loading address is important for the debugging and patching process. Participants have been taught on how the Portable Execute (PE) file was organized. CFF Explorer was used to change the PE file characteristics to load on a fixed address.

Using the executable files that have been written in Day 2, IDA-Free was used to perform static analysis on the files. Participants have been able to observe and identify the fixed static data area and the subroutines.

At the end of Day 3, dynamic analysis was also introduced by using WinDbg Preview software. Some of the commands in WinDbg Preview are similar as in debug.com in vDOS. WinDbg Preview was used to change the static data that have been found in Day 2 PE files to illustrate dynamic patching (as in Fig. 2). The patching involved changing the register value and the static data value.



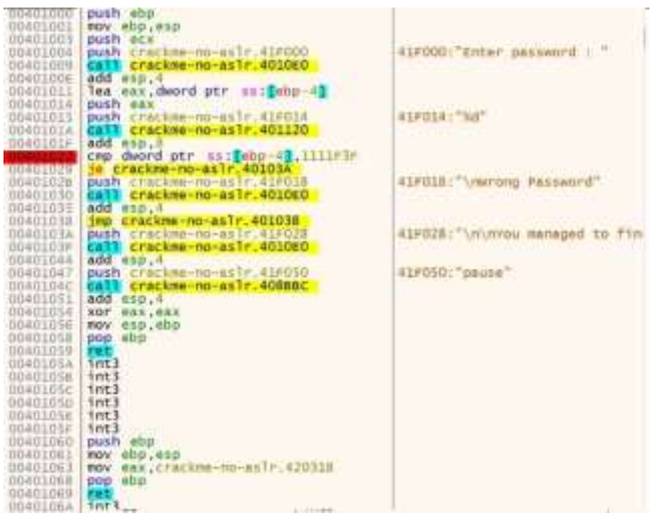
```
0:000> d 402025
00402025  10 fe a9 4f 00 00 23 26-2a 80 00 a4 00 00 00 1c  ...O..#&*.......
00402035  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00402045  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00402055  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00402065  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00402075  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00402085  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00402095  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................

0:000> e 402029 51 84

0:000> bu 401047

0:000> g
Breakpoint 0 hit
eax=000000a4 ebx=00000000 ecx=0000001c edx=00000023 esi=00000006 edi=00401000
eip=00401047 esp=0019ff74 ebp=0019ff80 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
dispnum32+0x1047:
00401047 e8ca1f0000      call    dispnum32+0x3016 (00403016)
```

Fig. 2 Changing Data Content in a Binary File Using WinDbg Preview

In encapsulating the Day 3 programming experience, participants have gained an understanding on the basics of reverse engineering through static and dynamic analysis. The use of Day 2 executable file could ease the patching exercises since the program was written by the participants themselves. The knowledge of using debug.com in Day1, could ease the participants in using WinDbg Preview.

4. Day 4 – Dynamic Analysis - Reversing, By-passing, and Patching Binary Files.

The approach for Day 4 was by way of sharing tips on performing reverse engineering. Participants have been given some of the useful techniques on where to look for password and expiry date checking, static data, stack framework, observing Windows API, dynamic allocation of virtual memory and the like. These tips and techniques have been useful to assist the participants to spark the momentum for reverse engineering process.



```
00401000 | push ebp
00401001 | mov ebp,esp
00401003 | push ecx
00401004 | push crackme-no-aslr.41F000                41F000:"Enter password : "
00401009 | call crackme-no-aslr.4010E0
0040100E | add esp,4
00401011 | lea eax,dword ptr ss:[ebp-4]
00401014 | push eax
00401015 | push crackme-no-aslr.41F014                41F014:"%d"
0040101A | call crackme-no-aslr.401120
0040101F | add esp,8
[illegible] | cmp dword ptr ss:[ebp-4],1111F3F
00401029 | je crackme-no-aslr.40103A
0040102B | push crackme-no-aslr.41F018                41F018:"\nWrong Password"
00401030 | call crackme-no-aslr.4010E0
00401035 | add esp,4
00401038 | jmp crackme-no-aslr.401038
0040103A | push crackme-no-aslr.41F028                41F028:"\n\nYou managed to fin
0040103F | call crackme-no-aslr.4010E0
00401044 | add esp,4
00401047 | push crackme-no-aslr.41F050                41F050:"pause"
0040104C | call crackme-no-aslr.4088BC
00401051 | add esp,4
00401054 | xor eax,eax
00401056 | mov esp,ebp
00401058 | pop ebp
00401059 | ret
0040105A | int3
0040105B | int3
0040105C | int3
0040105D | int3
0040105E | int3
0040105F | int3
00401060 | push ebp
00401061 | mov ebp,esp
00401063 | mov eax,crackme-no-aslr.420318
00401068 | pop ebp
00401069 | ret
0040106A | int3
```

Fig. 3 Changing Data Content in a Binary File Using WinDbg Preview

The x64dbg [15] software has been used to perform patching binary file and by-passing certain checking, i.e., password (as in Fig. 3). The participants performed reverse engineering on selected binary files from crackmes.one.

Crackmes.one is a site that contains crackmes, which is a small program designed to test a programmer's reverse engineering skills [3]. This was created by Stanislas [1]. Participants were exposed on the technique of patching registers and instruction sets (as in Fig. 4). They were able to perform the by-passing of the password checking routine, as well as to identify simple encryption algorithm and discover hardcoded password.

At the point of performing the reversing process, the participants have only been given some clues for them to experience themselves. Most course providers supply step-by-step guide for the lab exercises except in the “catch the flag (CTF)” exercise. The real hands-on experience in this programme was a trigger for them to apply all the knowledge gained from day one to day four. At the end of Day 4, it is interesting to note that 85% of the participants completed the assignments without much assistance.

In encapsulating the Day 4 session, participants have been able to perform dynamic analysis using x64dbg program and to apply the tips and techniques that have been learned from the practical sessions. Participants have been given the live patching and by-passing experience on dissecting the selected crackmes. Day 4 marked the end of the binary reverse engineering programme with the tenacity of providing the exposure and experience to software developers as workplace learners in encountering the real-world situation.

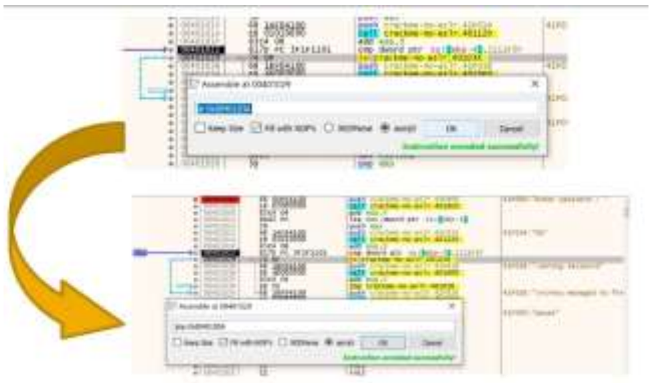


Fig. 4 Patching instruction using x64dbg

IV. PROGRAMME FEEDBACK

Most of the participants who have not had previous exposure to assembly language have modestly rated themselves as having poor skills before attending this programme. They account for 92%, as illustrated in Fig. 5. After the completion of the programme, it is worthy to note that 33% rated themselves as Very Good; 42% rated themselves as Good, whilst 25% rated themselves as Medium.

The overall evaluation was well received as the programme has improved their assembly language skills at the end of the four-day practical sessions.

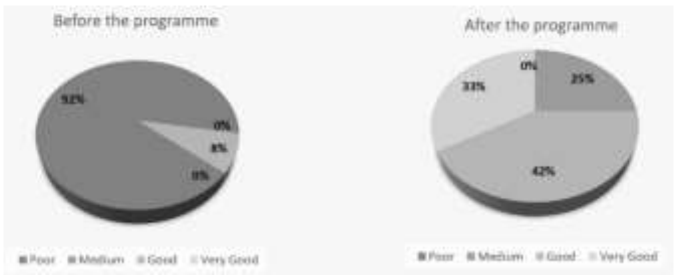


Fig 5 Skills before and after attending the programme

The participants indicated that they were pleased with the programme structure. In Fig. 6 there is indication that 50% of the participants who are the workplace learners rated it as good whilst the remaining 50% rated it as very good. This denotes that all the participants have been satisfied with the programme structure which entailed a blend of theory and practical aspects of binary reverse engineering using assembly language.

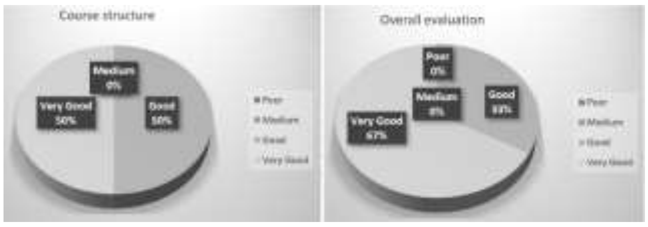


Fig. 6 Programme structure and overall evaluation

Rating for the overall program was overwhelming as 67% rated the course as very good and 33% rated as good. An interesting observation to note is the indication by the participants who have highlighted on the value of the plentiful hands-on exercises relating to patching and by-passing actual binary files, and they have also recommended for this programme to be conducted as a five-day programme with more lab exercises.

V. FINDINGS

1. Teaching assembly language can be effective by lessening the complexity of having to use a complicated operating system such as Windows or Linux. With the aid of virtual DOS, programming in assembly language and learning the operating system internals become simple. Visualizing 8-bit and 16-bit data is much easier for the participants in understanding the data movement and memory address especially in respect of the little-endian byte ordering. 16-bit processor is an easy path to learn

assembly language before engaging into 32-bit and 64-bit processor with the complex Windows operating system.

2. The art of reverse engineering is basically connected with the technique of debugging binary files. Introduction to debugging technique at the early stage could contribute to a better understanding on the working of assembly language. Doing programming using Debug.com in 16-bit vDOS should be able to speed-up the reversing skill and ease the ability to perform reverse engineering in Windows using WinDbg Preview software which represents the new Debug.com that runs on Windows.

3. Prior to dealing with complex malware analysis, basic knowledge of by-passing and patching would equip the participants with real reverse engineering on selected binary files. It would be pertinent for Software Engineers to be well exposed to the different techniques of by-passing and patching instruction on simple executable PE files in order to grasp a better understanding of PE files that are written in C and C++.

4. Professional courses on reverse engineering (SANS, Mandrake) lay emphasis on assembly language, theoretically within a brief duration. It will only be an advantage to an experienced and seasoned assembly language programmer to get the benefits of the binary reverse engineering course. In this regard, we strongly believe that some exposure in writing assembly language from scratch would be invaluable before embarking on a reverse engineering professional course, especially in performing reverse engineering on malware.

5. It is recommended for a participant to have knowledge of at least one high level programming language before embarking on assembly language programming, in order to understand a typical program flow.

6. In preparing for the new generation of reverse engineers and other new roles like malware analysts, digital forensic experts and others that are related to cybersecurity, it would be a good step if assembly language could be taught as a separate programming subject.

7. It would be highly recommended for higher education institutions to collaborate with the industry to undertake co-teaching in supporting the new roles within the realm of cybersecurity.

VI. PERSPECTIVE AND CONCLUSION

Adjudged from the Assembly Language for Reverse Engineers programme that has been crafted for five batches of software developers from the Product Development Department in HeiTech Padu Berhad, it has been well observed that being endowed with a working knowledge of developing application from scratch using assembly language would offer a foundation to be a good binary reverse engineer. This paper has unleashed some pointers in respect of x86 assembly language, practical programming session on Win32 assembly language, Static Analysis and Dynamic Analysis to ease the process of debugging, fixing, patching and by-passing executable files as a plausible programme structure to address the new demand for assembly language proficiency in performing binary reverse engineering tasks.

VI. ACKNOWLEDGEMENT

The authors hereby acknowledge the review support offered by the IJPCC reviewers who took their time to study the manuscript and find it acceptable for publishing

VII. CONFLICT OF INTEREST

The authors declare that there is no conflict of Interest.

REFERENCES

[1] Arnoud, Stanislas. “Creakmes.” Twitter, Twitter, https://twitter.com/sar5430/.
[2] Computing Curricula 2020. https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf
[3] “Crackme.” Wikipedia, Wikimedia Foundation, 6 Sept. 2022, https://en.wikipedia.org/wiki/Crackme.
[4] IDA Free, https://hex-rays.com/ida-free/.
[5] Jalal Kawash, Andrew Kuipers, Leonard Manzara, and Robert Collier. 2016. Undergraduate Assembly Language Instruction Sweetened with the Raspberry Pi. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). Association for Computing Machinery, New York, NY, USA, 498–503. https://doi.org/10.1145/2839509.2844552
[6] Mohd Shaid, S Z. “Introduction to Malware Reverse Engineering”. Issues in Computer Security & Networking, Part 1. pp. 105-127. ISBN 978-983-52-0905-5. Penerbit UTM Press
[7] Pistelli, Daniel. “CFF Explorer.” GitHub, 18 Apr. 2016, https://github.com/cybertechniques/site/blob/master/analysis_tools /cff-explorer/index.md.
[8] Roberge, Alexandre. “Neutralize Malware with Reverse Engineering.” Thot Cursus, Thot Cursus, 1 Feb. 2023, https://cursus.edu/en/26547/neutralize-malware-with-reverse-engineering.
[9] Sanati-Mehrizy, Reza, and Afsaneh Minaie. “A New Role of Assembly Language in Computer Engineering/Science Curriculum.” 2003 Annual Conference Proceedings, https://doi.org/10.18260/1-2--11839.
[10] Schaars, JHM. “VDOS .” VDos, https://www.vdos.info/index.html.
[11] Tatham, Simon, and Julian Hall. “NASM - Netwide Assembler.” NASM, https://www.nasm.us/.
[12] Tiobe Index.” TIOBE, 23 May 2023, https://www.tiobe.com/tiobe-index/.
[13] Vojta, Paul. “Debug 1.29: for Freedos.” Index of /Pub/Micro/PC-Stuff/Freedos/Files/DOS/Debug/1.29, http://www.ibiblio.org/pub/micro/pc-stuff/freedos/files/dos/debug/1.29/.
[14] “WinDbg Preview.” Microsoft Apps, Microsoft Corporation, https://apps.microsoft.com/store/detail/windbg-preview/9PGJGD53TN86?hl=en-my&amp;gl=my&amp;rtc=1.
[15] “X64DBG.” x64dbg, https://x64dbg.com/.