ANALYSIS OF SAMPLE EXECUTABLE FILES USING OLLYDBG BY ADDITION OF BREAKPOINTS

Dr. Sawtantar Singh Khurmi[1], Kulvir Singh[2], Harkirat Singh Brar[3]

[1]*Department of Computer Science & Engg., BMSCE, Sri Muktsar Sahib/ PTU, Jalandhar,(India)*

[2]*Department of Computer Science & Engineering, MIMIT, Malout/ PTU, Jalandhar,(India)*

[3]*Department of Computer Science & Engineering, MIMIT, Malout/ PTU, Jalandhar,(India)*

ABSTRACT

Most of the malwares come under the format of executable files. Reverse engineering approach can be applied for the detail study of malware like its behaviour; code etc. & results of this approach are fed to Antivirus (AV) in order to generate signatures. So it is important to how to run the executable files under safe conditions. We have used OllyDbg debugger due to its debugging capabilities, which allow user to execute the most interesting parts of the malicious program slowly and under highly controlled conditions, so user can better understand the purpose of code. This paper proposes the steps for the partial execution of any executable file by adding breakpoints on the specific locations for the safe malware analysis.

Keywords- AV, Malware, Reverse Engineering, OllyDbg, Breakpoint, Executables

I.INTRODUCTION

In computing, an executable file or executable program, or sometimes simply an executable, causes a computer "to perform indicated tasks according to encoded instructions,"[1] as opposed to a data file that must be parsed by a program to be meaningful. These instructions are traditionally machine code instructions for a physical CPU. However, in a more general sense, a file containing instructions (such as bytecode) for a software interpreter may also be considered executable; even a scripting language source file may therefore be considered executable in this sense. The exact interpretation depends upon the use; while the term often refers only to machine code files, in the context of protection against computer viruses all files which cause potentially hazardous instruction execution, including scripts, are lumped together for convenience. Malware, short for malicious software, is any software used to disrupt computer operation, gather sensitive information, or gain access to private computer systems.[1] Malware is defined by its malicious intent, acting against the requirements of the computer user, and does not include software that causes unintentional harm due to some deficiency. Reverse Engineering is the art of analyzing a system, software or an object to its minutest detail to understand its functionality/operation principles. A Malware is reverse engineered to understand the working of

a malware and the functionality and capability of the malware. The following are the main reasons to conduct reverse engineering of a malware: -- Assess damage -- Analyze malware functionality -- Identify vulnerability -- Catch the intruder -- Prepare signatures. The main aim of reverse engineering malware apart from aforementioned is to understand the attacker's methodology and skill set. This helps in deeply analyzing the attacker and thus mitigating future attacks on the network along with planning the removal of malware from the network.OllyDbg[7] 1.10 is a 32-bit assembler-level analyzing debugger for Microsoft(R) Windows(R) with intuitive interface. Emphasis on binary code analysis makes it particularly useful in cases where source is unavailable. It predicts contents of registers, recognizes procedures, API calls, switches, tables, constants and strings, locates routines from object files and libraries, allows custom labels and comments in disassembled code, writes patches back to executable file and more. OllyDbg can parse compiled Windows executables and, acting as disassembler, display their code as Intel x86 assembly instructions. OllyDbg also have debugging capabilities, which allow user to execute the most interesting parts of the malicious program slowly and under higher controlled conditions, so user can better understand the purpose of code.

II NEED OF ANALYSIS

Executable files are most frequently used in the Computer Systems. Most of the malwares come under the format of executable files. So it is very important for the user or the system to must be aware before running any executable file. Thus analysis is required for any executable file. Broadly malware analysis is of two categories: Static [5] & Dynamic [5]. Static analysis is study of the code step by step without completing its execution i.e. malicious code will be examined without executing or partially executing it. Static analysis is the safe analysis. On the other hand, Dynamic Analysis is the examining of the malicious code by executing it. It may cause damage to the system in which analysis is going on. Generally dynamic analysis is performed in the environment which can be sacrificed or in the virtual environment.

III METHODOLOGY

OllyDBg tool is used to perform analysis of the notepad.exe file. Here partial execution will be done by adding the breakpoints. Now the question is where to add breakpoint? As Dynamic Link Libraries[3] are responsible for system calls so breakpoints will be added on kernel32.dll file. It is must for the analyst to know the behaviour of executable file like no. of files created, no. of process created, no. of ports opened, and registry changes etc. As every time when notepad.exe is executed it creates a new file. So, in order to stop this event breakpoint should be applied over CreateFile function. There are two create file functions-CreateFileA & CreateFileW. CreateFileA is just a wrapper to convert ASCII to Unicode. CreateFileW is for the unicode and is always called for the creation of file. So finally breakpoint will be added at CreateFileW function in order to stop the execution of notepad.exe. Step by step snapshots for whole settings is shown below:

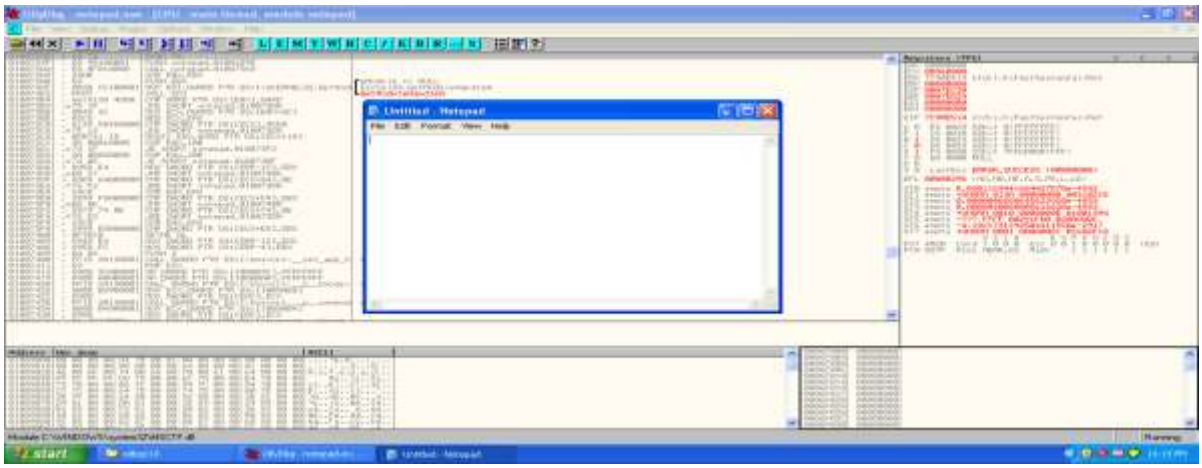

Fig. 1: Normal execution of Notepad.Exe without any breakpoint

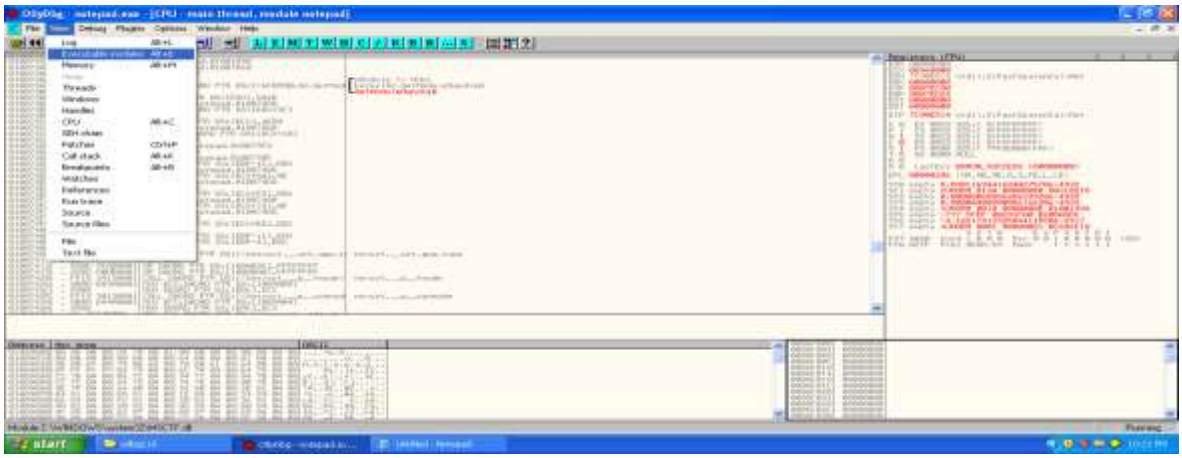

Fig. 2: Addition of breakpoint through executable module

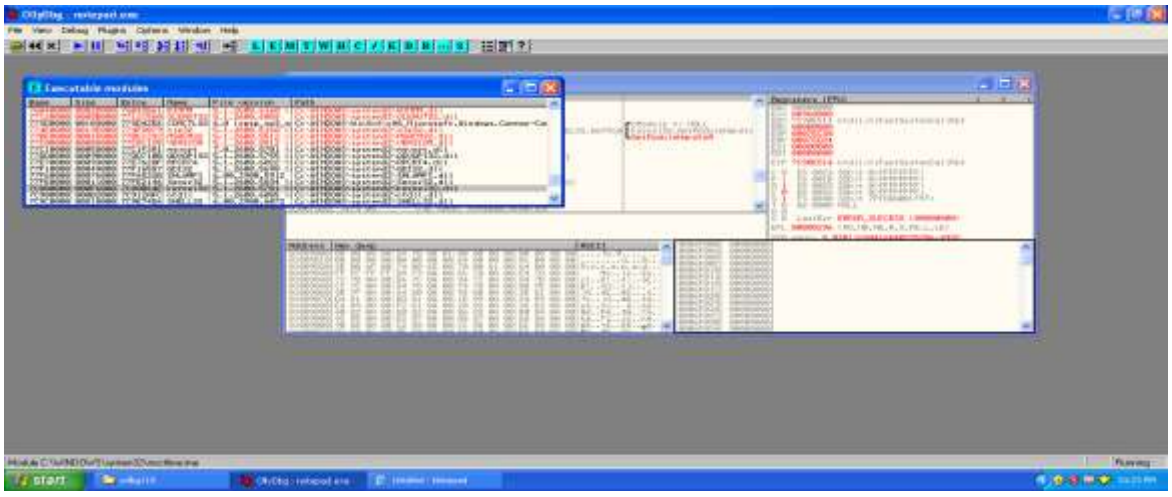

Fig. 3: Chossing Kernel.dll file

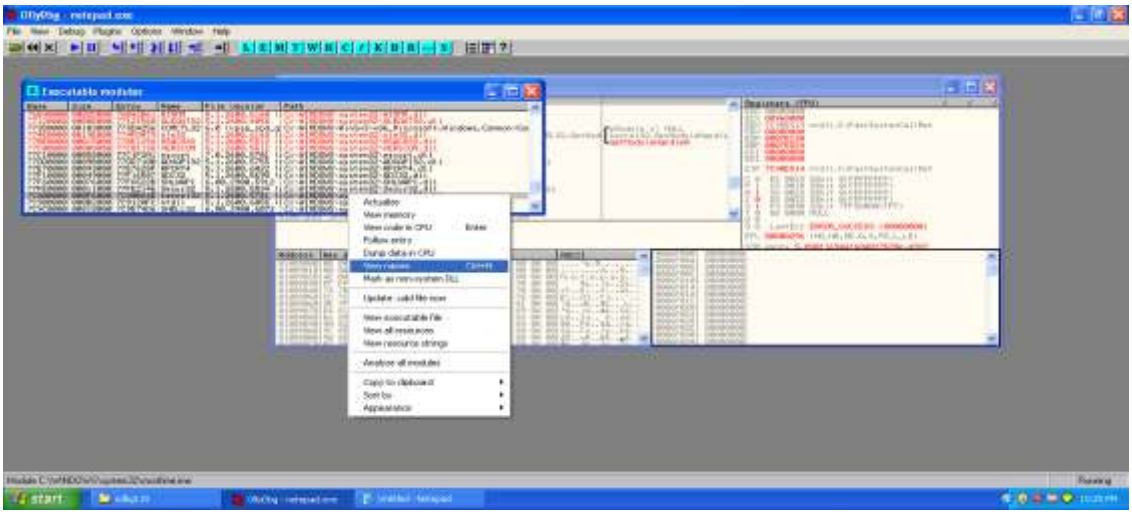

Fig. 4: Viewing names

Fig. 5: Selecting CreteFileW function

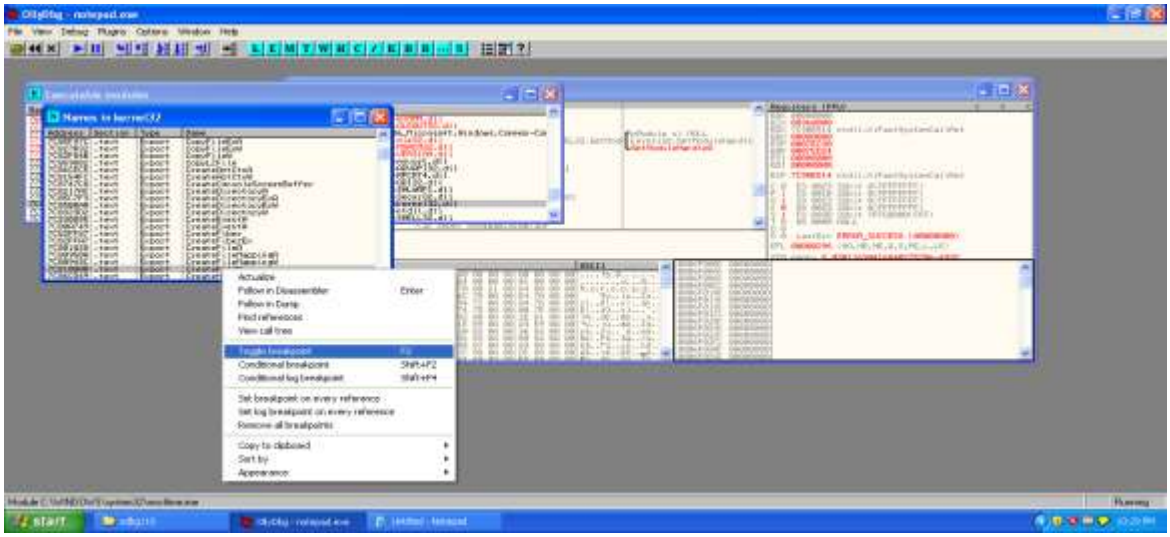

Fig. 6: Addition of Breakpoint

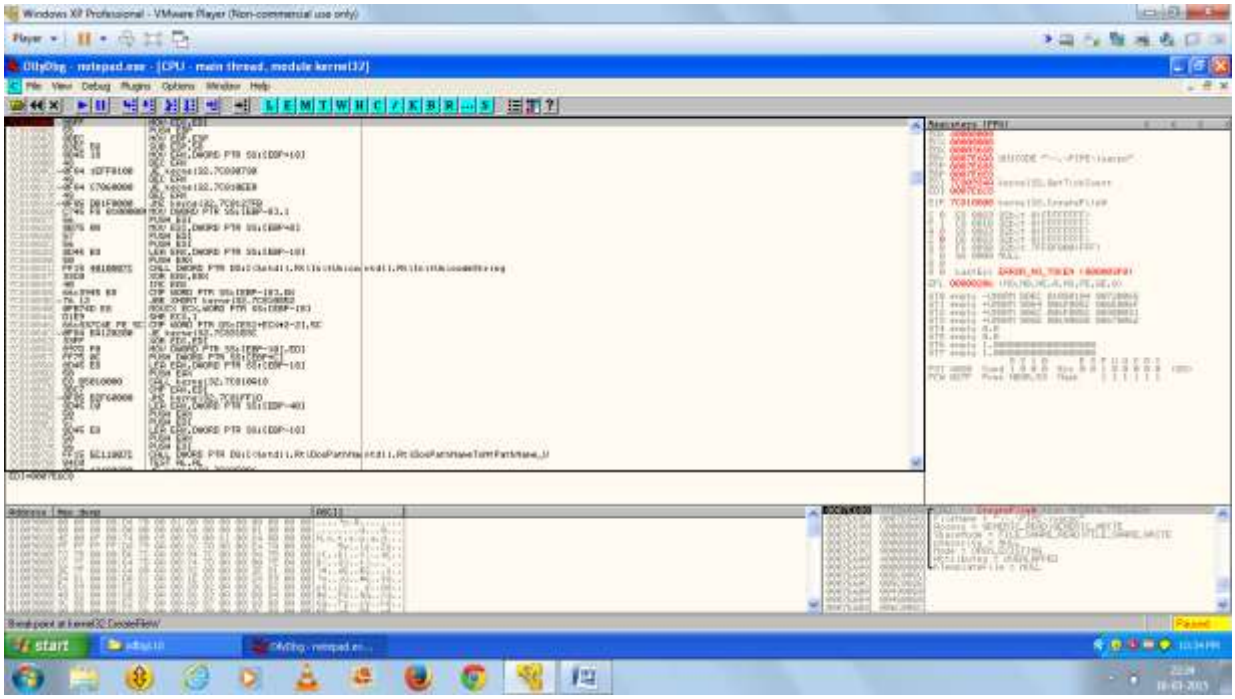

Fig. 7: Partial Execution of Notepad.exe

Above diagrams are the execution traceouts of OllyDbg for the partial execution of notepad.exe. It shows the step by step description how to add breakpoints on the kernell32.dll file by exploring the names of variuos under it.

IV RESULTS

Table 1: Mnemonic codes for normal & partial execution

Normal execution of Notepad .exe without breakpoint	**Execution of Notepad .exe with breakpoint**	**Message for Partial Execution**
EAX 00000000 ECX 00960000 EDX 7C90E514 ntdll.KiFastSystemCallRet EBX 00000000 ESP 0007EC30 EBP 0007ED24 ESI 00000000 EDI 00000000 EIP 7C90E514	EAX 40000000 ECX 00000000 EDX 0000051E EBX 0007E6A8 UNICODE "\\.\PIPE\lsarpc" ESP 0007E688 EBP 0007E6E8 ESI 7C80934A kernel32.GetTickCount EDI 0007E6C8	0007E688 77E84B2A /CALL to CreateFileW from RPCRT4.77E84B24

ntdll.KiFastSystemCallRet C 0 ES 0023 32bit 0(FFFFFFFF) P 1 CS 001B 32bit 0(FFFFFFFF) A 1 SS 0023 32bit 0(FFFFFFFF) Z 0 DS 0023 32bit 0(FFFFFFFF) S 1 FS 003B 32bit 7FFDF000(FFF) T 0 GS 0000 NULL D 0 O 0 LastErr ERROR_SUCCESS (00000000) EFL 00000296 (NO,NB,NE,A,S,PE,L,LE) ST0 empty 5.0301169441683534340e-4932 ST1 empty +UNORM 013A 00000000 000102D9 ST2 empty 0.0000064189652390030e-4933 ST3 empty 0.0000040080005616620e-4933 ST4 empty +UNORM 0010 0000000E 01001394 ST5 empty -NAN FFFF 806D9295 B217A3DC ST6 empty - 4.1821731292504411550e-2917 ST7 empty +UNORM 0001 00000081 BC6B5E88 3 2 1 0 E S P U O Z D I FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EQ) FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1	EIP 7C810800 kernel32.CreateFileW C 0 ES 0023 32bit 0(FFFFFFFF) P 1 CS 001B 32bit 0(FFFFFFFF) A 0 SS 0023 32bit 0(FFFFFFFF) Z 0 DS 0023 32bit 0(FFFFFFFF) S 0 FS 003B 32bit 7FFDE000(FFF) T 0 GS 0000 NULL D 0 O 0 LastErr ERROR_NO_TOKEN (000003F0) EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G) ST0 empty -UNORM BDEC 01050104 00730065 ST1 empty +UNORM 0064 006F005C 006E006F ST2 empty +UNORM 006C 006F005C 00300031 ST3 empty +UNORM 006E 0069002E 00670062 ST4 empty 0.0 ST5 empty 0.0 ST6 empty 1.0000000000000000000 ST7 empty 1.0000000000000000000 3 2 1 0 E S P U O Z D I FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EQ) FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1	

Above table shows the register status/contents during normal execution and partial execution with breakpoints. Here breakpoint has stopped the remote procedure call (RPCRT4), which does not allow creating of a new file.

V CONCLUSION & FUTURE SCOPE

Here a simple notepad.exe is used as a sample for executable file. This work provides the detail usage of OllyDbg tool for the partial execution. It concludes that most of the malicious code creates the new files during its execution. So such creation can be stopped by adding breakpoints and restricting the malware to partial execution. In future, same steps can be used for the analysis of any malicious file.

REFERENCES

[1] "A Cyveillance Analysis August 2010: Malware detection rates for leading AV solutions." https://www.cyveillance.com/web/docs/WP_MalwareDetectionRates.pdf. Last accessed Feb.10,2013.

[2]Jacob, G. Debar, H. & Filiol, E., "Behavioral detection of malware: From a survey towards an established taxonomy", Journal in Computer Virology, 4, 251–266, 2008.

[3] Gil Tahan, Lior Rokach and Yuval Shahar, "Automatic Malware Detection Using Common Segment Analysis and Meta-Features", Journal of Machine Learning Research, pp- 949-979, 2012.

[4] Mohammad Nour Saffaf: Malware Analysis Bachelor‟s Thesis., Helsinki Metropolia University of Applied Sciences, May 27, 2009.

[5] Vinod P. V.Laxmi,M.S.Gaur, "Survey on Malware Detection Methods", 3rd Hackers Workshop on Computer and Internet Security, Department of Computer Science and Engineering, Prabhu Goel Research Centre for Computer & Internet Security,IIT, Kanpur, pp-74-79, March,2009.

[6] Eilam, Eldad (2005). "Reversing: secrets of reverse engineering" Wiley. p. 118. ISBN 978-0-7645-7481-8.

[7] Ferguson, Justin; Kaminsky, Dan (2008). "Reverse engineering code with IDA Pro." Syngress. p. 130. ISBN 978-1-59749-237-9.