ANALYSIS OF THE SOFTWARE BEHAVIOUR USING FORENSIC METHODS FOR COMPUTER SECURITY PURPOSES

Liberios VOKOROKOS, Branislav MADOŠ, Marek ČAJKOVSKÝ, Ján HURTUK, Kristián MORAVČÍK
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, tel. +421 55 602 3023,
e-mail: {liberios.vokorokos, branislav.mados, marek.cajkovsky, jan.hurtuk}@tuke.sk, krsitian.moravcik@student.tuke.sk

ABSTRACT

Abstract: Static analysis of malicious software is a complicated process. This complication stems from the fact that the process of analysis has to be carried out on the malicious binary file which is represented in assembly language and therefore lacks critical semantics such as functions, types and buffers that are only found in the source code of high-level languages. This work presents a method that is built on top of IDA Pro disassembler and can be used for analysis of binary files by describing their structure using finite-state automaton. This kind of approach allows a reverse engineer to perform advanced features such as visualization, comparison, etc on a malicious file.

Keywords: *reverse engineering, malware analysis, static analysis, finite state automaton, visualization*

1. INTRODUCTION

The tremendous growth of the internet over the previous decade led to the emergence of countless new activities which are using internet, such as the various financial services, business solutions, social networking and others. However, the rapid development of the internet has brought several critical issues with itself, mainly in the area of computer systems security which to this day still continues to require further design of optimal solutions.

One of the fundamental problems in the field of computer security is the enormous number of malicious software that intentionally makes harm and causes damage in the operating system or in its files and is generally denoted by the term malware. Malware is currently one of the most serious security threats and is a major cause of most problems on the internet, such as spam, Distributed Denial of Services (DDoS) attacks, etc. [1].

Number of computer programs written for malicious and illegal purposes is rapidly increasing every year. The year 2012 has seen a record number of 27 million new malware samples with an average amount of 74 000 specimens per day, while the predominant attack (76% of the total) were Trojans [2].

That is why new discipline of computer science called forensic computing was established, with the main aim to investigate computer crimes using information technology and enable to identify, analyze, maintain, and provide the evidence gathered [3].

Despite the above however, there are still not enough sophisticated tools for analyzing malicious software that would be able to clearly identify the behaviour of these programs through a thorough analysis of their internal structure.

The aim of this work is to present a solution that could be applied within methods of static and dynamic malware analysis to describe the structure of a binary file using finite-state machines and then demonstrate its use on selected samples of malicious software.

2. MALWARE ANALYSIS

In order to protect authorized users against the dangers of the Internet, various security companies are offering antivirus products in which the primary technique to identify new malware threats lies in the comparison of signatures [4].

On this basis, it is necessary to constantly update the database of known signatures in the event of discovery of new samples of unknown malware.

The problem with existing antivirus solutions is the insufficient protection in case of zero-day attacks because complex malware may use sophisticated defence mechanisms to prevent detection by antivirus tools or because the database does not contain the relevant signature to identify these threats [5][6].

Given the vast number of new samples that appear every day there is a need for new solutions that could be used to effectively automate the process of classifying the collected malware samples and to distinguish them from existing documented threats. The process of malware detection automation can be addressed applying the methods of forensic analysis, using a static or dynamic analysis on binary files.

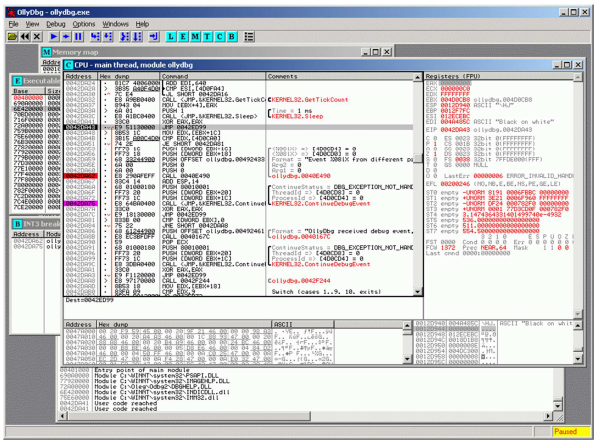

Fig. 1 OllyDBG software tool

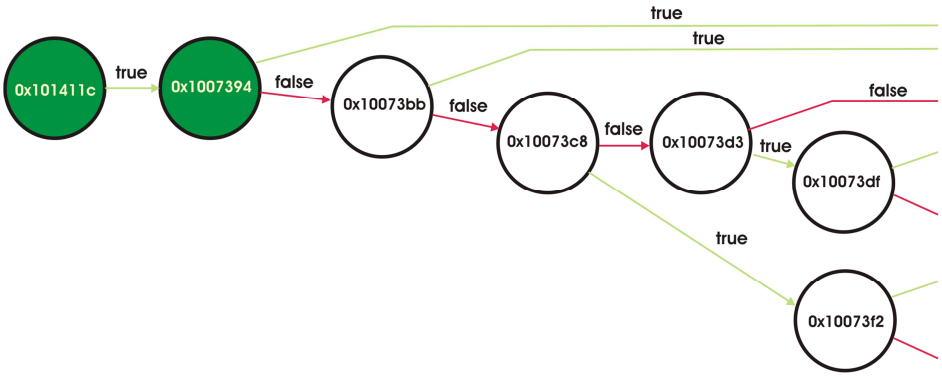

Fig. 2 Identifying the presence of malicious code in the program

If we focus our attention on analysis tools which as their output produces a solution in the form of disassembled code we can find that such generated code represents only static data that is data structures over which it is not possible to perform some other additional specific operations.

In most cases, the generated content is represented using text mode or some kind of graph view to be able to make the process of analysis easier in case the binary file that contains thousands of assembly instructions. To be able to disassemble malicious binary files, tools such as IDA Pro, OllyDbg (Figure 2), etc can be used [7].

In case of IDA Pro, the output of the analysis, that is the disassembled code can be represented using text view or more helpful graph view. In text mode it is possible to store the disassembled sequence of instructions in different output formats such as .asm, .html, etc. With graph view, the reverse engineer can use the internal viewing capabilities of IDA or generate an output graph to be viewed in third party tools. The problem of graph view lies in the fact that IDA can only apply it on the functions of the analyzed binary file. In the case of malware that may use some kind of obfuscation technique, IDA can fail to identify functions present in the program and therefore there is no graph available at all for the reverse-engineer to analyze [8].

In addition to static analysis IDA also offers program debugging capabilities, in which the binary file is loaded into computer memory and the process of stepping can be applied where the program is executed one line at a time. During the process of debugging, IDA offers an option to log every executed instruction that was interpreted by the CPU in a text file. OllyDbg's primary use is for debugging Windows applications. Within this dynamic analysis process, OllyDbg offers only a standard text view of disassembled code.

Similar to IDA, interpreted instructions can be recorded to a text file for the purposes of a later analysis. OllyDbg can be extended using third party plugins that can add additional features to the program such as graph view. By comparing the above mentioned tools, it is clear that they produce the same results.

The difference between them lies only in the analysis process. The outcome of static analysis is the complete structure of the binary file in case the analysis is being done in laboratory conditions e.g. the code does not contain signs of obfuscation. Dynamic analysis has the advantage that before executing the actual program data, instructions are loaded into the computer memory which activates the process of deobfuscation.

The problem arises when there is a need to perform additional operations with the data on the output that is produced as the result of analysis. In terms of solving this particular problem, it was necessary to design an approach that would be able to represent these data as a new data structure and thus the proposed approach would be able to process any kind of sequence of instructions without the need for specific type of analysis.

After the consideration of existing data structures it was decided that the proposed approach would represent evaluated data based on finite-state automaton. Using finite-state machines as a data structure enables to perform complex operations on collected data such as visualization, comparison, state reduction, etc.

3. SOLLUTION AND RESULTS

One of the main issues that had to be resolved concerned with the question of representation of the data in the resulting form. Finite-state machine is a mathematical model of computation and is conceived as an abstract machine that can be in one of finite number of states. It can change from one state to another when initiated by a triggering event condition. Individual states are defined by a state-transition function that for each specific combination of state and input alphabet returns a new state.

The initial approach of the problem solution was based on the implementation of the concept, in which each individual instruction was represented by a specific state. After thorough testing of this idea, it was rejected because of the enormous number of generated states which the proposed program produced within the finite-state machine as the output.

During the analysis of this approach we also experimented with the omission of procedures within the output so that the resulting finite-state machine would only focus to cover the main branch of the program. In this case, however, the condition to analyze the whole structure of the program would not be met. Said principle, however, could operate in dynamic analysis, which needs to be concerned only with one actual executed branch of the analyzed program.

After performing several tests it was decided that the best approach would be to represent each state of the automaton as if they were basic blocks of the program, as is the case with IDA Pro tool which uses basic blocks when representing the program in graph view. Basic block is defined as a sequence of instructions within this block and every block has a single entry and exit point.

Each defined block of the code is responsible for transferring control flow to other responsible instruction that follows the sequence of blocks. For each block is a characteristic in which all the instructions in a given block must be carried out after initial activation of the first instruction.

To show the practicality of our proposed approach we have prepared case studies to demonstrate the use of our implemented solution ida-fsm on the real malware samples.

3.1. Application of the analysis: Visualization

To demonstrate the visualization capabilities of ida-fsm we have acquired a sample of a trojan (md5: 1429fff7a09b103e43613273c24b7781). Subsequently the sample was loaded into IDA Pro for initial analysis. Because of the limitations of IDA graphing capabilities no functions were identified during the analysis and therefore no graph view was available to process. In cases like this it is obvious that the sample contains obfuscation technique. To detect the type of present obfuscation, we used ProtectionID tool. It was revealed that the sample is packed using UPX. The advantage of our solution is in the capability to produce output even when there are traces of obfuscation present. Ida-fsm can be used for visualizing the structure of the file in the form of finite -state automaton.

This kind of visualization enables us to generate signatures for different samples. Using these signatures we can easily identify used protections in the analyzed samples. To verify our assumption we used UPX to compress a benign application.

Afterwards we ran out tool ida-fsm on the packed binary. The generated state machine contained the same structure as the output for iExplore1.exe trojan.

3.2. Application of the analysis: Comparison

For the purpose of demonstrating the comparison capabilities of our tool, we chose notepad.exe (md5: 388b8fbc36a8558587afc90fb23a3b99) as an example of the benign program which is included with every Windows XP installation.

The file was loaded into IDA Pro for analysis and we ran ida-fsm on the file to generate the automaton. Later on we applied msfvenom tool on this binary to infect it with a backdoor payload and to create a trojan sample. To check the integrity of the created malware we uploaded the sample to VirusTotal online service where 33 antivirus solutions from 51 identified the file as a malware. Subsequently we loaded the created trojan to IDA Pro and applied our tool to compare the newly created state machine with the previous state machine created for benign notepad.exe. Based on the results, our tool can highlight new states that were introduced into the state machine as the result of infecting the benign file.

According to highlighted state we can easily identify the basic blocks where the payload is located and analyze the virtual memory address where the malware links to the program.

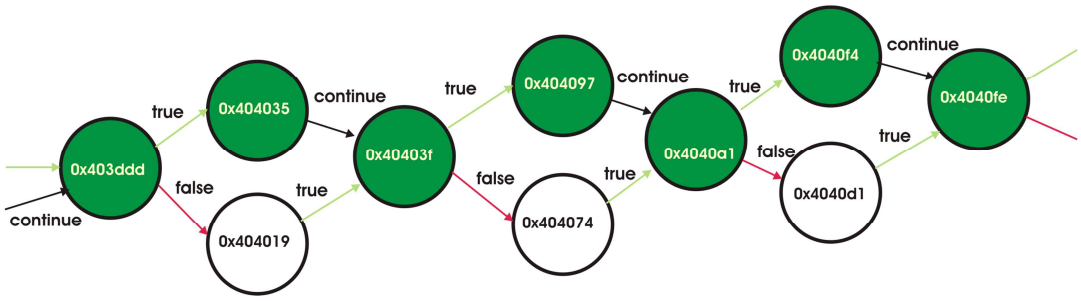

Fig. 3 Execution trace of malicious file

To further demonstrate the comparison capabilities of ida-fsm we obtained a sample of a computer worm (md5: 24279b569c7f301460e0c092c80f0919). We loaded the malicious binary into malwasm tool which uses binary instrumentation to capture every executed instruction of the program. Using this approach enables us to visualize the execution trace of the malicious file using ida-fsm by comparing the virtual addresses of the executed instructions with the virtual addresses of the state machines basic blocks. Figure 3 shows a part of the generated execution trace for the analyzed malware based on instructions captured using malwasm's binary instrumentation technique.

4. CONCLUSIONS

The goal of this work was to implement a simple tool to help in reverse engineering malicious binary files using static analysis methods by describing their internal structure in the form of finite-state automaton to be able to perform advanced operations such as visualization, comparison, and others.

Based on presented case studies our tool ida-fsm provides several advantages. During static analysis of obfuscated files it can be used to generate a signature which is based on the visualization. Analyzing the signature of each file enables us to clearly identify the

type of obfuscation that is present in it. Another use for ida-fsm resides in the comparison of generated finite-state machines which can help identify potential presence of malicious code in the binary files and can help to generate trace execution of analyzed file when ida-fsm is used in combination with dynamic analysis tools.

There are also some disadvantages to consider. Since our tool is based on Python language, the processing of large files can cause some performance issues. Another issue concerns the algorithm that is used to process the instructions of the disassembled binary file. There are still some minor discrepancies during the generated output that needs to be solved. These disadvantages might be improved by the next research.

ACKNOWLEDGMENTS

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and project KEGA 008TUKE-4/2013: Microlearning environment for education of information security specialists.

REFERENCES

[1] KOLBITSCH, C. – COMPARETTI, P. M. – KRUEGEL, C. – KIRDA, E. – ZHOU, X. – WANG, X. F.: Effective and Efficient Malware Detection at the End Host. *USENIX Security Symposium*. 2009, pp. 351-366, ISBN 978-1-931971-69-0.

[2] Panda Security S.L.: PandaLabs Annual Report. 2012 Summary. PandaL- abs, 2012. Retrieved October 19, 2013 from http://press.pandasecurity. com/wp – content/uploads/2013/02/PandaLabs-Annual-Report-2012.pdf.

[3] MCKEMMISH, R.: What is forensic computing?, In: Trends and Issues in Crime and Criminal Justice, Australian Institute of Criminology, June 1999, ISSN 0817-8542, ISBN 0 642 24102 3.

[4] BAYER, U. – KRUEGEL, CH. – KIRDA, E.: TTAnalyze: A tool for analyzing malware. 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference.

[5] CARDENAS, A. A. – AMIN, S. – LIN, Z. – HUANG, Y. – HUANG, CH. – SASTRY, S.: Attacks against process control systems: Risk assessment, detection, and response. Proceedings of the 6th ACM symposium on information, computer and communications security (ASIACCS 2011), ACM, 2011, pp. 355-366, ISBN: 978-1-4503-0564-8.

[6] BAILEY, M. – OBERHEIDE, J. – ANDERSEN, J. – MAO, Z. M. – JAHANIAN, F. – NAZARIO, J.: Automated classification and analysis of internet malware. Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, 2007, 5th September 2007, XP019098093, ISBN: 978-3-540-74319-4, pp. 178- 197.

[7] QUIST, D. A. – LIEBROCK, L. M.: Visualizing Compiled Executables for Malware Analysis. Visualization for Cyber Security, 2009. VizSec 2009. 6th International Workshop on Visualization for Cyber Security 2009, Atlantic City, New Jersey, USA October 11, 2009, pp. 27-32, ISBN: 978-1-4244-5413-6.

[8] Eagle Chris: The IDA Pro Book: The Unoficial Guide to the World's Most Popular Disassembler. No Starch Press, 2011. 672 p. ISBN 978-1593272890.

Received May 22, 2014, accepted June 20, 2014

BIOGRAPHIES

Liberios Vokorokos (prof., Ing., PhD.) was born on 17. November 1966 in Greece. In 1991 he graduated (MSc.) with honours at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He defended his PhD. in the field of programming device and systems in 2000; his thesis title was "Diagnosis of compound systems using the Data Flow applications". He was appointed professor for Computers Science and Informatics in 2005. Since 1995 he is working as an educationist at the Department of Computers and Informatics. His scientific research is focusing on parallel computers of the Data Flow type. In addition to this, he also investigates the questions related to the diagnostics of complex systems. Currently he is dean of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. His other professional interests include the membership on the Advisory Committee for Informatization at the faculty and Advisory Board for the Development and Informatization at Technical University of Košice.

Branislav Madoš (Ing., PhD.) was born on 20th May 1976 in Trebišov, Slovakia. In 2006 he graduated (MSc.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. He defended his PhD. in the field of Computers and computer systems in 2009; his thesis title was "Specialized architecture of data flow computer". Since 2010 he is working as a professor assistant at the Department of Computers and Informatics. His scientific research is focused on the parallel computer architectures and architectures of computers with data driven computational model.

Marek Čajkovský (Ing.) was born on 17th December 1986 in Veľký Krtíš, Slovakia. In 2011 he graduated (MSc.) at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice and received the engineering degree. Since 2011 he is PhD. student at Faculty of Electrical Engineering and Informatics at Technical University of Košice. His research is focused on computer security, the title of his doctoral thesis is: Identifying Security Threats by System Services Calling. His professional interests include programming, computer networking, computer security and UNIX based operating systems.

Ján Hurtuk (Ing.) was born on 4th October 1988 in Kežmarok. In 2013 he graduated (MSc.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. Since 2014 he is studying as a PhD. student at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice. His scientific research is mainly focused on the computer security.

Kristián Moravčík (Ing.) was born on 27th March 1990 in Kráľovský Chlmec, Slovakia. He received the engineering degree in Informatics in 2014 from the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University of Košice. His research is focused on computer security.