IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

Malware Catcher using Crypto Identifier

Thomsy William

Department of Computer Science and Engineering, Viswajyothi College of Engineering and Technology,

Vazhakulam, Kerala, India

Abstract: Today the world is heading towards internet related activities at every zone of life. With the same pace the threat of malwares are also proportionally increasing with the usage. Although most of the threats detecting strategies are highly active, malware builders are also trying to strengthen their shield to overcome malware detection. Cryptography's dark side is being utilized for this purpose. Using cryptography the appearance of the malicious code is scrambled, thereby helping to bypass the anti-virus employed for detection purpose. Hence to identify the underlying cryptography is the main goal to be achieved to stop such malicious activities. To identify the presence of cryptography the execution of such programs were monitored using a DBA tool named Valgrind. The results shows the memory locations of famous cryptographic routines. Further with the help of signature based matching, the malicious presence was confirmed.

Index Terms: AES; Cryptography; Malware; Malware Signature; RSA; SHA-1; Valgrind.

I. INTRODUCTION

Cryptography is a way in which one scramble the data so that no one understands its actual meaning. Hackers have utilized the dark side of cryptography. Traditionally cryptography was used for defending malicious activities. But now, cryptography is also used as a means of hiding the attacker's content from recognition. Using cryptography attackers try to break the similarity between what an antivirus analyst views about a virus and what the virus writer views. The major problem solved by this thesis work is to overthrow the cryptography employed by malwares. Most of the malwares employ many of the software which could simply encrypt the given data as per the requirement. The encrypted data is then send to the destination wherein the malicious activities are performed. Still manual analysis is being carried out by researchers on malware binaries to reverse engineer the decryption algorithms. Manual malware analysis, however, is highly labor intensive and cannot cope up with the increase in number of encrypted malware binaries. The Storm and Kraken bots are the most recent and widely publicized examples of malware that encrypt their communication.

Today a large number of malware analysis platforms exists which successfully hides the cryptographic present in it. Now before analyzing deep into various malwares, one need to initially over throw the shield created by the cryptographic functions. Through cryptographic identifier, this system tries to reach out to the infected code. So here the first phase is to develop a crypto identifier referring to various existing techniques and then based on pattern matching, the signatures of malwares have to be analyzed.

Mostly two types of encryptions are incorporated, they include the one using keys and the one without keys. Most of the malware writers transmit the key for encryption, along with the encrypted code [by storing it to the very end of the file, with less recognition probability]. Hence once the code is delivered to the destination, it could use this hidden key to uncover the scrambled malicious code and carryout the respective malicious actions. So normally to hide the key most of the hackers prefer the footer end location of the file. This is normal scenario undertaken by the hackers of this era.

In the past the recovery of crypto routines a lot of manual effort was required for this reverse engineering and analysis. But here the approach is to automate this process of finding encrypted malware portions that contains crypto functions as well. The knowledge about these parts helps analysts to extract the detail functionality of tools and create decryption add-ons for monitoring tools. These add-ons include signature based malware checker which could possibly confirm the presence of malwares. Most of the anti-malwares identifies malwares by checking for the presence of signatures of the malwares. But they fail if such signatures are scrambled and made into unreadable form or meaningless form.

Most of the existing systems recognizes cryptographic presence by analyzing the execution of the encrypted software. Normally dynamic binary analysis tools are used to analyze the execution traces of the program under execution. One such tool called Valgrind is used to monitor the execution. The online analysis result of valgrind is taken as the analysis log. Once the cryptographic presence is confirmed then comes the malware checker based on signatures. Most of the malwares poses a signature which refers to the identity of the malware. The most important fact which was revealed through this work is that none of the current software implementation of already existing cryptographic algorithms achieves perfect secrecy if their execution is viewed.

II. MALWARE ANALYSIS

In the world of malware detection analysis of the existing malwares holds a major prerequisite for any anti malware. To analyze malwares many techniques exist. Among them the most successful one is the binary analysis. As this paper tries to introduce an encrypted malware identifier,

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

the cryptography has to be identified first. There exist many tools which performs the analysis based on monitoring of binary analysis and various other heuristics. Let us ponder through some of the mostly accepted and widely used cryptography analysis techniques and cryptographic identifiers.

A focus on cryptography alone exists only in few approaches. Finding of cryptography can be categorized into static and dynamic analysis techniques. Many works [2]-[8] addresses the task of binary analysis. One such paper was Inspector Gadget [6] which presented an algorithm to automatically extract details from a given binary executable. Thus from the details collected one has to come to a conclusion of cryptographic presence. So next target to be achieved was to identify the cryptographic primitives from the so called details collected. Paper which tried to accomplish such a strategy is discussed in [7]. Next came a publicly available infrastructure for performing program verification and analysis tasks on binary code, explained as BAP [8]. Likewise among the few systems introduced the most successful one was CIS [10], which proposed a system to identify and extract crypto algorithms in binary code.

Most of the existing systems tried to identify the presence through monitoring of the execution traces. But none of them could actually reveal the presence of malicious code. And only the existence of the encryption was identified and confirmation of malicious presence was left unclear.

III. SYSTEM STUCTURE AND DESIGN

Figure 1 shows the structure of the entire system in a capsule format. This structure shows a proof of concept model.

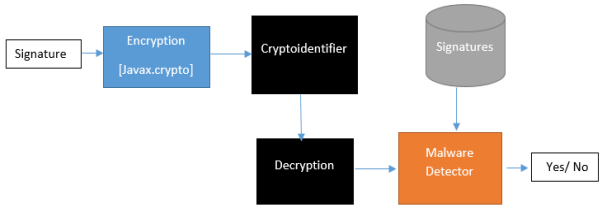

Figure 1: Overview of the entire system

The system contains a pre-requisite section or the first stage. This section is viewed from a black hat hacker's view. The system performs encryption on the signature [eg. bc356bae4c42f19a3de16e333ba3569c] of malware using in build java class. It then stores the key within the file with malicious code in a location which is noticed very rarely. Instead of signature one could also encrypt any program code which is containing the malicious related activity. Signature is being taken here because almost all anti-malwares are compromising most of the malwares by identifying their respective signatures. So the malicious code writer will surely do his maximum to avoid the recognition of signature. Once the required data is ready [i.e., the encrypted malicious code], it is then forwarded to the proposed crypto identifier. Wherein the data is scrutinized under regular scan at the OS level. On a practical view every program snippets undergo this check irrespective of any of the visible malicious nature [like file extensions]. Once the presence of cryptography is confirmed, it is then forwarded to second stage of analysis. In this stage only the malware presence is fully confirmed. A database containing signatures is being used to cross check with the output of the previous stage.

A. Encrypting stage

Encryption stage is the pre-requisite for further analysis. A model of the encrypted malicious content is made explicitly. Here the signature is taken for encryption. Many numbers of encryptions can be embedded into the signature. In the initial stages the malicious code writers simply tried to scramble the sensitive data i.e., the signature using any one encryption. But nowadays most of them employ more than one encryption methods over to the signature one after the other. Here also 3 encryptions are basically incorporated. They include RSA, AES and SHA-1. RSA and AES are used back to back in a single stretch in any order. Either AES first followed by RSA or vice versa. That is if AES is used initially for encryption then the result obtained after encryption is used as the input for the second encryption RSA. Whereas SHA-1 is used completely isolated. No other encryption is mixed with it. This encryption strategy can be varied based on one's taste for encryption. This thesis work follows this strategy of encryption. Encrypting the malware signature with three main crypto algorithms with a hacker's mind set is the main preprocessing task. Then this particular section as a whole undergoes the monitoring process.

B. Analysis of Cryptography: Valgrind

There exists much system which recognizes cryptographic presence by analyzing the execution of the encrypted software. A dynamic binary analysis tool is used to analyze the execution traces of the malicious encrypted code. Dynamic binary analysis (DBA) tools are tools that analyze programs at run-time at the level of machine code. Such a tool called valgrind is used to collect the log regarding the execution traces of this module [11]. Based on the log details obtained the following objectives were accomplished.

1. Determining the presence of any cryptographic functions.
2. Analysis of Cryptography functions.
3. Finding the hidden encryption keys.
4. Confirming malware signature match.

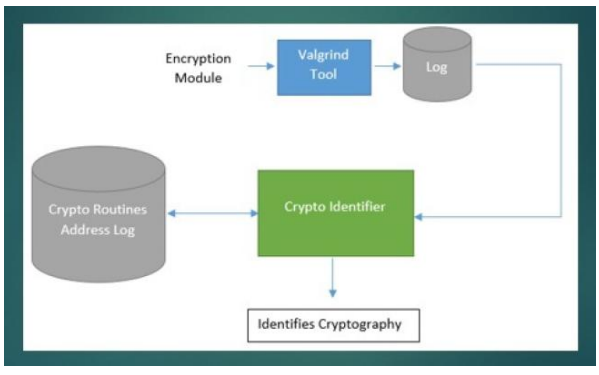

Figure 2: Overview of Encryption analysis section

DOI 10.17148/IJARCCE.2015.41134

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

After completing the phase of preprocessing the model is ready to look forward for the encryption analysis phase. As explained earlier, this phase is accomplished with the help of valgrind. Here the valgrind acts as an intermediate who provides the malware catcher with the details of the currently running program at the memory level in a detail form. Here every code under execution is under monitoring of the valgrind tool. As an initial step to the model, many samples were verified using valgrind tool. Both infected and good program code were verified using this famous DBI tool. The output generated from this tool is shown in figure 3.

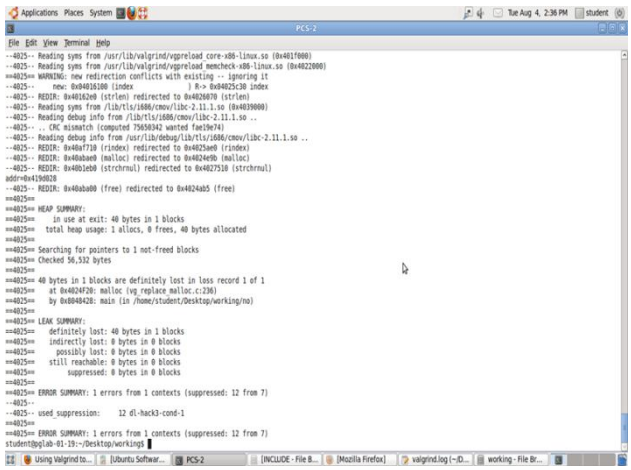

Figure 3: Valgrind Output

The output thus generated is manually verified for the presence of certain code word which represents the crypto routines used in a Linux executable. Names of the main crypto routines identified are shown following table I.

Address	Name of Routine
04153e4	AES_CFB256_encrypt()
040af710	SHA_Final()
040abae0	AES_cfb128_cipher()
040b1eb0	RSA
04025c30	SHA1()

Table I: Crypto Routines

This crypto routines table is referred from the base paper [1].

Once the presence of these routines are identified it could come to the conclusion of presence of encryption. With this new step one thing is very much clear that to date no proper mechanism is discovered to keep the encrypted materials hidden, when its execution is being monitored.

C. Decryption and Signature Identification

Once the encryption presence is confirmed, the next step for further confirmation of malwares has to be performed. According to this proof of concept model three main encryptions are being tested. Among them two of them are based on encryption keys [AES and RSA] and one is a hashing function.

Most of the virus writers tries to hide their code using encryption which is very complicated and on a series fashion. That is the actual virus content will be hidden first and will be followed by next set of encryption methods. So with this thesis work the focus is to exploit this nature of malware creators. This model's preprocessing stage have tried to perform such a similar approach of encryption.

Now for decryption of symmetric encryption the initial process is to identify the hidden key for decryption. Here again the common trend shown among the black hat hackers for hiding the keys are exploited. Most of the hackers of this era tries to generate an encrypted unreadable version of the code with a large content. And they try to make it even larger which makes the security checker less interested to go further through the unreadable data content. But here comes the actual brain work of the attacker, that, he/ she saves the key ,required for the encryption ,to the footer location of this particular file, which will be very deep end of the file.

Once the keys are identified the decryption process is carried out in a series fashion. Since the hashing function does not possess a reversal of encrypted code, the only option is to perform hashing once more on the final output of the second decryption step and compare it with the actual encrypted data. The decrypted content is then verified with a set of malware signatures. If matching is found in the comparison then the malware presence is confirmed. Malware signature is like a fingerprint that can be used to detect and identify specific viruses. Normally anti-virus software uses the database of virus signature to identify malicious code. E.g. Signatures - bc356bae4c42f19a3de16e333ba3569c. Such a collection of signatures is made in order to compare the results.

IV. PERFORMANCE EVALUATION

The major goal of this work is to identify encrypted malwares from the given code. For the performance evaluation process false positive and negative rates are taken. The goal of testing was to find out false positive and false negative. Based on signatures, malware detector detects the malware presence.

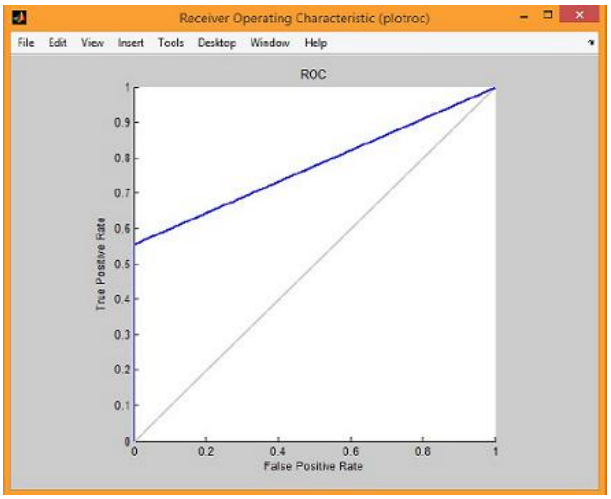

Figure 4: ROC Curve of malware detection

DOI 10.17148/IJARCCE.2015.41134

Normally antivirus programs compare their database of virus signatures with the files. The antivirus vendor updates the signatures frequently and makes them available to customers via Web. Nowadays, malware detection system requires high detection rate and low false alarm rate.

For this proof of concept model many signatures were collected. The major source for this collection process was from web.

V. CONCLUSION AND FUTURE WORK

Finding and extracting cryptographic functions from binary executable is often a hard reverse engineering task that requires a lot of manual effort. Still, it is an essentially important analysis step in the fight against malware. This work tries to make a first step into this direction by deriving important requirements any crypto detection framework should fulfil. The proposed prototype have successfully shown that it is capable to detect public key cryptography, block cipher and hash operations. Initially the prerequisite of the prototype was accomplished by providing three encryptions. Then with the help of crypto identifier the system tried to identify the encryptions incorporated. It is then followed by decryption and finally ensuring the malicious presence using signature based malware identification. One proposed extension of this work could be to build a dynamic binary analysis tool without depending on Valgrind.

REFERENCES

[1] Xin Li,Xinyuan Wang and Wentao Chang, \CipherXRay: Exposing Cryptographic Operations and Transient Secrets from Monitored Binary Execution",IEEE Transactions on dependable and secure computing, VOL. 11, NO.2, MARCH/APRIL 2014.
[2] Chi-Keung Luk , Robert Cohn , Robert Muth , Harish Patil Artur Klauser ,Geo_ Lowney , StevenWallace , Vijay Janapa Reddi and Kim Hazelwood , \Pin - Building Customized Program Analysis Tools with Dynamic Instrumentation,"Proc. Intl Conf. World Wide Web (WWW), ACM 2005.
[3] Nicholas Nethercote and Julian Seward, \Valgrind - A Framework for Heavy-weight Dynamic Binary Instrumentation ,"Proc. 16th Intl Conf. World Wide Web(WWW),ACM,2007.
[4] Johannes Kinder and Helmut Veith, \Jakstab - A Static Analysis Platform for Binaries ,"SIGIR ACM,2008.
[5] Juan Caballero, Noah M. Johnson, Stephen McCamant and Dawn Song,\Binary Code Extraction and Interface Identification for Security Applications,"IEEE , 2009.
[6] Clemens Kolbitsch , Thorsten Holz , Christopher Kruegel and Engin Kirda, \Inspector Gadget-Automated Extraction of Proprietary Gadgets from Malware Binaries,"IEEE Symposium on Security and Privacy,2010.
[7] Grobert F , Willems C and Holz T, \Automated Identification of Cryptographic Primitives in Binary Programs,"Transactions on Secure Computing,2010.
[8] David Brumley, Ivan Jager, Thanassis Avgerinos and Edward J.Schwartz, \BAP - A Binary Analysis Platform ,"ACM, 2012.
[9] Joan Calvet , Jos M. Fernandez and Jean-Yves Marion, \Cryptographic Function Identification in Obfuscated Binary Programs,"ACM, 2012
[10] Felix Matenaar, Andre Wichmann, Felix Leder and Elmar Gerhards-Padilla ,\CIS The Crpto Intelligence System for Automatic Detection and Localization of Cryptographic Functions in Current Malware," IEEE, 2012.
[11] [ONLINE] http://valgrind.org/docs/manual/QuickStart.html
[12] [ONLINE] http://www.findjar.com/jar/bouncycastle/bouncycastle-jce-jdk13/112/bouncycastle-jce-jdk13-112.jar.html
[13] [ONLINE] https://blog.malwarebytes.org/intelligence/2013/04/malware-in-a- jar/
[14] [ONLINE] https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html