Detection of Cryptographic Operations in Malware Binaries

[1] Anitha Abraham,[2] Dr. Varghese Paul,[3] Chinju.K
[1]Final Year M Tech Student School of Computer Sciences Mahatma Gandhi University
[2] Associate Professor Department of IT, CUSAT,
[3]Assistant Professor, School of Computer Sciences, Mahatma Gandhi University

Abstract- **Malware, short for malicious software, is any software used to disrupt computer operation, gather sensitive information, or gain access to private computer systems. Nowadays malwares are becoming increasingly stealthy, most of malwares are using cryptographic algorithms to protect themselves from being analyzed. The use of cryptographic algorithms and truly transient cryptographic secrets inside the malware binary imposes a key obstacle to effective malware analysis and defense. CipherXRay is a novel binary analysis framework that can be used for effective malware analysis and defense. It can automatically identify and recover the cryptographic operations and transient secrets from the execution of potentially obfuscated binary executables. CipherXRay is based on the avalanche effect, which is a desirable property of cryptographic functions. The avalanche effect means that, a slight change in the input causes a significant change in the output, ie flipping a single bit in the input changes half of the output bits. Another feature of the avalanche effect is that it allows us to accurately pinpoint the location, size and boundary of both the input and output buffers. Using avalanche effect, CipherXRay is able to accurately pinpoint the boundary of cryptographic operation and recover truly transient cryptographic secrets that only exist in memory for one instant in between multiple nested cryptographic operations.**

Keywords: **Binary analysis, Cryptographic Operations, Key Recovery, Transient secret, Avalanche effect**

I. INTRODUCTION

Malware analysis, forensics and reverse engineering seek to understand the inner workings of malware, which are invaluable to defending against malware. Most of malwares are using cryptographic algorithms to prevent themselves from being analyzed. To prevent the in-memory cryptographic secrets from being recovered by key searching tools, sophisticated malware can make the cryptographic secrets truly transient in memory by encrypting or destroying the secrets right after using them at run-time. The use of cryptographic algorithms and truly transient cryptographic secrets inside the malware binary executable imposes a key obstacle to effective malware analysis and defense.

A number of methods [2], [3], [4] have been proposed to automatically recover the cryptographic keys from process memory or file. However, these methods require the cryptographic keys to be statically stored in plain text form. These methods are not effective when the cryptographic keys are stored encrypted or transient.

Recently proposed binary analysis approaches [5],[6],[7] are able to automatically detect the existence of cryptographic operations from a given binary execution. These methods are based on instruction profiling and signature of particular cryptographic implementations. However, they are not effective in the presence of multiple rounds of cryptographic operations.

In order to recover such transient cryptographic secrets, one needs to reliably identify not only exactly where but also exactly when those transient cryptographic secrets will be in memory. This requires one to accurately pinpoint the boundary of each of the multiple rounds of cryptographic operations. No existing binary analysis could accurately pinpoint the boundary between multiple rounds of cryptographic operations and recover truly transient cryptographic secrets from the execution of a given binary executable.

CipherXRay is a novel binary analysis framework that can be used to accurately pinpoint the boundary of individual cryptographic operation from multiple rounds of cryptographic operations and recover truly transient secrets from the execution of a potentially obfuscated binary executable. It is build upon avalanche effect, which refers to the desirable property of cryptographic function. Avalanche effect means that one bit change in the input or key would cause significant change in the output.

II. LITERATURE REVIEW

An exhaustive literature survey has been conducted to identify related research works conducted in this area. Abstracts of some of the most relevant research works are included below.

A. ReFormat: Automatic Reverse Engineering of Encrypted Messages

ReFormat, is a system that aims at deriving the message format even when the message is encrypted. According to this approach an encrypted input message will typically go through two phases: message decryption and normal protocol processing. With the help of data lifetime analysis of run-time buffers, it can pinpoint the memory locations that contain the decrypted message generated from the first phase and are later accessed in the second phase.

In particular, by intercepting system calls that are used to read from and write to file descriptors and/or network sockets, ReFormat taints the input message and applies taint analysis technique to keep track of the instructions that access tainted memory space. By dynamically instrumenting the program execution, the taint information can be properly propagated and a trace of the instructions that operate on tainted data will be collected.

After collecting an execution trace, phase profiler divides it into different execution phases. It need to recognize the boundary between the first two phases. The first phase is recognized as the "message decryption" phase, and the last three phases aggregately is referred as the "normal protocol processing" phase.

In order to divide an execution trace into these two phases, it need to search for the transition point between them. The first step is to use the cumulative percentage of arithmetic and bit-wise instructions to narrow down the search range where the transition point is located. Then it needs to identify the maximum and minimum instructions based on the cumulative percentage. The next step is to compute the percentage of arithmetic and bitwise instructions for each function fragment between them.

The basic idea is to identify the buffers that must be written in the former phase and read in the latter phase. To identify such buffers, it needs to analyze the lifetime of memory buffers. It identifies the memory buffers that contain the decrypted message based on the liveness definition. Initially it searches for all the buffers that were written to in the message decryption phase and are still live when the application enters the subsequent phase. Second, it searches for all the buffers that are live when they are being first read from in the second phase. The buffers in the intersection of the two sets are identified as those that contain the decrypted message.

ReFormat can only handle applications where there exists a single boundary between decryption and normal protocol processing. However, multiple such boundaries may exist. Reformat was not designed to identify the buffers holding the unencoded data before encoding. However it is not effective in the presence of multiple rounds of cryptographic operations.

B. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse engineering

Automatic protocol reverse-engineering is important for the analysis and defense against botnets. A zombie army/botnet is a number of Internet computers that, have been set up to forward spam or viruses to other computers on the Internet. Security analysts often need to rewrite messages sent and received by a bot in order to provide the botmaster with an illusion of successful and unhampered operation. Dispatcher is used to extract the format of protocol messages sent by an application.

Reverse-engineering of encrypted protocols needs to address the following problems. In the case of received messages, it needs to identify the buffers holding the unencrypted data at the point that the decryption has finished. In the case of sent messages, it needs to identify the buffers holding the unencrypted data before the encryption. After identifying the buffers holding the unencrypted data, protocol reverse engineering techniques can be applied on them.

Dispatcher makes a forward pass over the input execution trace and replicates the call-stack of the application. It does this by monitoring the call and return instructions. For each function it computes the ratio between the number of arithmetic and bit-wise operations over the total number of instructions in the function. Instructions belonging to any invoked sub functions are excluded from this ratio. It is computed for each of the occurrence of the function in the trace. A function is identified as encoding function if it executes a minimum number of instructions and has a ratio larger than a pre-defined threshold.

Dispatcher computes the read set to identify the buffers holding the unencrypted data before encryption. Read set includes the set of locations read inside the encryption routine before being written. The buffers holding the unencrypted data, the encryption key, and any hard-coded tables used by the routine are included in the read set for the encryption routine. Dispatcher computes the write set to identify the buffers holding the unencrypted data after decryption. Write set is the set of locations written inside the decryption routine and read later in the trace. However these methods are not effective in the presence of multiple rounds of cryptographic operations.

X. Automated Identification of Cryptographic Primitives in Binary Programs

This paper, presents several methods to identify cryptographic primitives within a given binary program in an automated way. They perform fine-grained dynamic binary analysis and use the collected information as input for several heuristics that characterize specific, unique aspects of cryptographic code. These methods can successfully extract cryptographic keys from a given malware binary. The system implementation is divided in two stages, which are performed for each analysis of a binary sample. In the first stage, it performs fine-grained binary instrumentation, and the second stage implements several heuristics to identify cryptographic code from the data gathered by the first stage.

During the controlled execution of the target binary program it uses the technique of dynamic binary instrumentation (DBI) to gain insight on the program flow. DBI is used to collect an execution trace. Execution trace also includes the memory areas accessed and modified by the program. DBI framework Pin is used which supports fine-grained instruction-level tracing of a single process. This implementation creates a run trace of a software sample to gather the relevant data for the second stage.

In the second stage, the instruction and data trace is used to detect cryptographic algorithms, and their parameters. To detect the algorithms and their parameters, it first elevates the trace to high-level structured representations, i.e., loops, basic blocks, and graphs. Then, it employs different identification methods and utilize the high-level representation of the trace to inspect the execution for cryptographic primitives. Based on the findings, the tool generates a report that displays the results, especially the identified algorithms and their parameters.

Methods for identification of cryptographic primitives are of mainly two classes: signature-based and generic. The main differentiation is the knowledge needed for the identification algorithm. For signature-based identification, a priori knowledge about the specific cryptographic algorithm or implementation is needed. On the other hand, for generic identification it uses characteristics common to all cryptographic algorithms and therefore do not need any specific knowledge. But this method was not effective in the presence of multiple nested cryptographic operations.

III. CIPHERXRAY

A. *Goals and Assumptions*

Given an obfuscated binary executable, we want to uncover the cryptographic operations and their secrets from the execution. Specifically, CipherXRay can

- Determine if there is any cryptographic operation in its execution. If present it can further pinpoint the location of all the cryptographic functions, their respective mode and the order of execution.
- Pinpoint the location, size and boundary of the input and the output buffers used by each cryptographic function identified.
- Determine exactly when the input and the output of each cryptographic function will be at which buffers. This enables us to recover those truly transient input and output of each cryptographic operation that will be immediately destroyed or re-encrypted after run-time use.
- Determine if there is any key used in each cryptographic operation. If present it can recover the key even if it will be destroyed right after run-time use.

It is assumed that CipherXRay monitor the execution of the binary executable. Moreover, the input and the output of any called cryptographic functions is assumed to reside in some continuous memory buffer at run-time. The private key, used in the cryptographic function could be stored in a transformed form, and it will be derived at run-time.

B. *The Principle of CipherXRay*

CipherXRay is designed upon the avalanche effect, which refers to the desirable property of all cryptographic algorithms. A slight change in the input would cause significant changes in the output. This property is referred to as avalanche effect. Cryptographic functions are designed to exhibit the avalanche effect. Non-cryptographic code almost never has the avalanche effect. Therefore, the avalanche effect is a fairly unique and defining characteristic of all good cryptographic functions. This enables us to reliably identify the cryptographic operations from potentially obfuscated executables.

Another nice feature of the avalanche effect is that it allows us to accurately pinpoint the location, size and boundary of both the input and output buffers. While changing different bits in the input buffer results different bits changed in the output buffer, the changed bits are cohesive within the fixed output buffer.

Let $p \geq 1$ and $y \geq 1$ be the number of bytes of the input and the output of the cryptographic function respectively. Because of the avalanche effect, any single bit change in the input would cause $4y$ bits changed in the output. Those changed $4y$ bits in the output are said to be “touched” by the bit change in the input.

Given the location of buffers p and *q*, one can measure the impact of buffer p to buffer q by repeatedly changing different bits in buffer p and comparing the corresponding results in buffer *q*. This method is however not practical when the location of the input and the output buffers is unknown. Due to potential inherent randomness, different runs of one executable may generate different internal states and outputs with exact the same input. Therefore, it is desirable to be able to measure the impact between any two chosen buffers with only one run of a given binary executable.

If any bit in memory impacts any other bit in memory during the binary execution, then there must exist information flow between those two bits. If we can track the information flow from a given source, we could measure the impact between memory buffers and detect the avalanche effect with a single run of the binary executable.

C. *Overall CipherXRay Architecture*

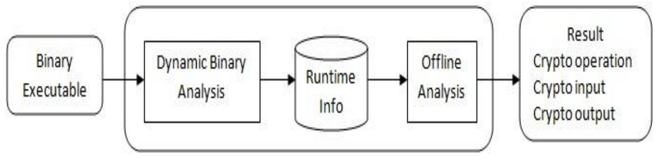

Fig.1 CipherXRay Architecture

To handle potential dynamic binary transformations inside a binary executable, CipherXRay framework is build upon dynamic binary analysis (DBA). Since the cryptographic operations inside the binary executable are generally independent from the underlying operating system, the proposed system analyzes only the user space binary executables.

Figure 1 shows the overall architecture of CipherXRay. CipherXRay dynamically intercepts and instruments the runtime instructions of the binary executable and collect valuable run-time information about the binary's execution. CipherXRay tracks and records the taint propagation, the address and value of the bytes involved in the taint propagation. It further analyzes the recorded runtime information and checks the patterns of instruction execution and memory access for any avalanche effect. Based on the detected avalanche effect patterns, CipherXRay identifies cryptographic operations and determines the exact location, size and boundary of the input buffer, the output buffer and any key buffer involved in each of the identified cryptographic operations and the exact time when the input, the output and the key will be in their corresponding buffers.

IV.CONCLUSIONS AND FUTURE ENHANCEMENT

Based on the defining characteristic of all good cryptographic algorithms – avalanche effect, CipherXRay has been shown to be able to detect block cipher and hash operations and pinpoint exactly when and where the cryptographic input, output, IV and keys will be in memory even if they exist only for a few microseconds. Current software implementations achieve hardly any secrecy if their execution can be monitored

REFERENCES

[1] Xin Li, Xinyuan Wang, 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] A. Shamir and N. van Someren. Playing Hide and Seek with Stored Keys. In *Proceedings of the Third International Conference on Financial Cryptography (FC 1999)*, pages 118 – 124, February 1999.

[3] T. Pettersson. Cryptographic Key Recovery from Linux Memory Dumps.In *Presentation, Chaos Communication Camp*, August 2007.

[4] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proceedings of the 17th USENIX Security Symposium*, pages 45–60. USENIX, August 2008.

[5] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. ReFormat: Automatic Reverse Engineering of Encrypted Messages. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS 2009)*, pages 200–215, September 2009.

[6] F. Gr¨obert, C. Willems, and T. Holz. Automated Identification of Cryptographic Primitives in Binary Programs. In *Proceedings of the 14th* International Symposium on Recent Advances in Intrusion Detection *(RAID 2011)*, September 2011.

[7] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse engineering. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009)*, pages 621–634. ACM, October 2009.