e-ISSN: 2349-9745
p-ISSN: 2393-8161
Scientific Journal Impact Factor (SJIF): 1.711
International Journal of Modern Trends in Engineering and Research
www.ijmter.com

Multi-Secure Message For Exposing Cryptographic Processes

Saranya.B[1],Chinnadurai.S[2],Srimadevi.S[3],Divya.K[4]
[1]*Department of ME-CSE, Srinivasan Engineering College*
[2]*Assistant Professor/CSE, Srinivasan Engineering College*
[3]*Department of ME-CSE, Srinivasan Engineering College*
[4]*Department of ME-CSE, Srinivasan Engineering College*

Abstract—Nowadays, malwares remain fetching gradually crafty, further more malwares be there with cryptographic algorithms toward defend themselves after actuality investigated. The use of crypto judgment algorithms and certainly fleeting crypto logic secrets within the malware dualistic executes a key problem toward operational malware examination and protection. In this paper CipherXRay an innovative dualistic examination structure is planned to allow additional operative malware examination. CipherXRay can spontaneously recognize and get well the cryptographic processes and temporary confidences from the implementation of possibly obscured dualistic executable. CipherXRay can able to precisely locate the border of cryptographic process additional classify convinced process modes of the recognized block cipher and express whether the recognized block cipher operation is encryption or decryption.

Keywords: Binary analysis, avalanche effect, key recovery attack on cryptosystem, transient cryptographic secret recovery, reverse engineering

I. INTRODUCTION

Malware analysis, forensics, and reverse engineering seek to understand the inner workings of malware, which are invaluable to defending against malware. To prevent themselves from being analyzed and reverse engineered, more and more malwares (e.g., Agobot, MegaD, Kraken, Conficker) are using cryptographic algorithms (e.g., packing , encrypting C&C communication) to protect the malicious code and communication . To prevent the in-memory cryptographic secrets (e.g., key, IV) from being then recovered by key searching tools (e.g., rsakeyfind), sophisticated malware can make the cryptographic secrets truly transient in memory by encrypting or destroying the secrets right after using them at runtime. 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. In order to recover the true logic of packed malware and the plaintext of the encrypted malware C&C communication, we have to be able to effectively analyse the cryptographic operations and recover their secrets from the malware binary executable.

A number of methods 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 plaintext form and they are not effective when the cryptographic keys are stored encrypted or transient. Recently proposed binary analysis approaches are able to automatically detect the existence of cryptographic operations from a given binary execution based on instruction profiling (i.e., calculate the percentage of bitwise and arithmetic instruction) and signature (e.g., specific constants and sequence of mnemonics) of particular cryptographic implementations. However, they are not effective in the presence of multiple rounds of cryptographic operations. For example, sophisticated security software

such as all the sensitive data (e.g., passwords, master keys) right after they have been decrypted and used at runtime. As illustrated in Fig. 1, the plaintext form cryptographic secrets are transient in that they exist in memory for only one instant between cipher 1 and cipher 2 operations. KeePass has claimed [4] "even if you would dumpthe KeePass process memory to disk, you couldn't find the passwords."

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. Unfortunately, existing instruction profiling and signature- based binary analysis approaches tend to think multiple rounds of cryptographic operations as one big cryptographic operation thus they are not able to recover the transient secrets in between multiple rounds of cryptographic operations. To the best of our knowledge, 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—a novel binary analysis framework that can 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. Instead of using instruction profiling, we build CipherXRay upon one of the defining characteristics of all (good) cryptographic algorithms—the avalanche effect, which refers to the desired property of cryptographic function such that 1 bit change in the input or key would cause significant change in the output.

CipherXRay has the following nice features:
1. It is able to reliably detect, pinpoint, and distinguish the operations of public key cryptographic algorithms (e.g., RSA), block cipher (e.g., AES), and hash (e.g., SHA-1), even if they are mingled with each other or noncryptographic operations, from the execution of a given binary executable.
2. It can accurately pinpoint the location, size, and
boundary of the input, the output, and the key buffers of each of the multiple rounds of cryptographic operations and determine the exact time when the input, the output, the IV, and the key of each identified cryptographic operation will be ready. This allows us to recover the data input, the data output, the secret key (e.g., 256-bit AES key) or the private key (e.g., 1,024-bit RSA key) used in every identified cryptographic operation even if multiple cryptographic operations are nested (e.g., encryption first and hash second) and the cryptographic secrets are transient.
3. CipherXRay can further identify certain operation nodes (e.g., ECB, CBC, CFB) of the identified block cipher and tell whether the identified cipher operation is encryption or decryption in certain cases.
4. Since the binary code obfuscation (e.g., self-modifying code) does not change or remove the avalanche effect of any cryptographic operations in the original binary executable, CipherXRay could be effective on obfuscated binary executable as long as the avalanche effect can be detected.
5. It is quite generic in that it is not dependent on specific implementation.

We have empirically evaluated the effectiveness of CipherXRay with OpenSSL, popular KeepassX password safe, malware Stuxnet, Kraken and Agobot, and a number of third party softwares with built-in chechsum and compression. Despite that KeePassX reencrypts all the sensitive data within 21 microseconds after they have been decrypted and used at runtime, CipherXRay is able to recover not only all the protected entry passwords but also the 256-bit master key and the 128-bit IV that enable one to directly decrypt the KeePassX password file using OpenSSL; CipherXRay is also able to recover all block cipher secret keys from the binary executables obfuscated by a number of packers (e.g., UPX

,ASPack, PECompact). CipherXRay has successfully recovered the secret key used by Agobot and the secrets of proprietary ciphers used by Kraken and Stuxnet malware.

To the best of our knowledge, CipherXRay is the first binary analysis framework that can accurately 1) pinpoint the boundary between multiple rounds of cryptographic operations; 2) recover truly transient cryptographic secrets and keys that exist in runtime memory for only a few micro seconds; and 3) recover the type of cryptographic operations and certain modes of operation of block ciphers. Our results demonstrate that the current software implementation of existing cryptographic algorithms achieves virtually no secrecy if their execution can be monitored. The rest of this paper is organized as follows: Section 2 overviews CipherXRay. Section 3 describes how to detect the avalanche effect. Section 4 discusses how to recover cryptographic secrets based on the avalanche effect. Section 5 presents how to recover the type, mode of operation of the identified cryptographic operation. Section 6 presents the empirical evaluation of CipherXRay. Section 7 discusses CipherXRay's implication, limitation, and potential countermeasures. Section 8 overviews related works. Section 9 concludes the paper.

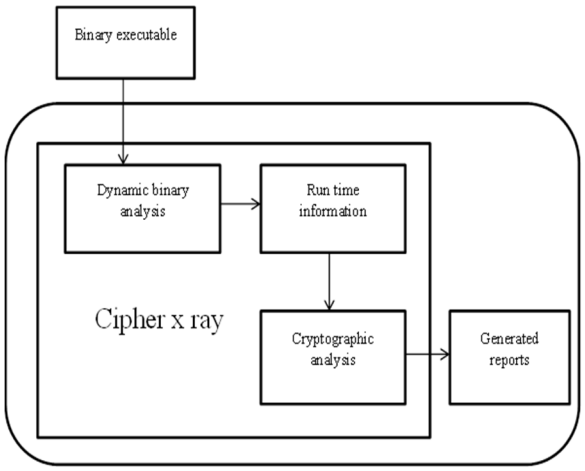

Fig 1.1 Architecture

II. RELATED WORKS

CipherXRay is designed upon the avalanche effect, which refers to the desirable property of all cryptographic algorithms (e.g., public key cryptographic algorithms, symmetric cryptographic algorithms, hash functions) such that a slight change (e.g., flipping a single bit) in the input would cause significant changes (e.g., half the output bits flip) in the output. Specifically, cryptographic functions are designed to exhibit the avalanche effect despite any obfuscation might be used in the implementation. On the other hand, noncryptographic code almost never has the avalanche effect. Therefore, the avalanche effect is an 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. The avalanche effect on the output buffer by changing two different bits in the input buffer. 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 $m \geq 1$ and $n \geq 1$ 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 4n bits

changed in the output. Here, we call those changed 4n bits in the output "touched" by the bit change in the input. Assume the 4n bits in the output touched by any bit in the input are random and independent from each other, then no more than 8n 2x bits in the output would remain untouched afterx > 1 different bit changes in the input. In other words, changing eight different bits in the input of a good cryptographic function would leave no more than half bit untouched in a 128-bit output.

To handle potential dynamic binary transformations (e.g., packing) inside a binary executable, we build our CipherXRay framework upon dynamic binary analysis (DBA). On the other hand, we leverage results from static binary analysis to help dynamic binary analysis whenever possible. Since the cryptographic operations inside the binary executable is generally independent from the underlying operating system, we focus on analyzing the user space binary executables in this proof-of-concept research.

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 (if any) will be in their corresponding buffers.

Assume we are able to effectively track the information flow across cryptographic functions, and we know the taint source (e.g., information received from the network), we want to identify if any part of the information flow from the taint source exhibits any avalanche effect. To the best of our knowledge, all existing cryptographic function implementations store the input, the output, and the key of the cryptographic function in continuous memory buffers due to efficiency considerations. Therefore, we need ways to effectively and efficiently identify the avalanche effect between any two given continuous memory buffers of certain sizes.

III. SCOPE AND OUTLINE OF THE PAPER

To evaluate CipherXRay's capability in detecting cryptographic operations and revealing the internals of the cryptographic operations, we designed test program which reads the plaintext from a disk file and processes the plaintext with multiple rounds of block cipher encryption and decryption followed by a final round of hash function.

The two block ciphers, Blowfish and AES-256, use different secret keys. In this experiment, CipherXRay should be able to distinguish a block cipher from a hash function. For block ciphers, CipherXRay should further identify the block size, the operation mode, the chaining mode, the secret key, the size, and the location of the corresponding input buffer and output buffer; for hash functions, CipherXRay should be able to tell the size and the location of the input buffer and output hash buffer.

Node Deployment

In this module the sender and receiver's are configured first. Then the server selects the operations by using algorithm such as RSA, Block Cipher's and SHA hash algorithm in a random manner. After choosing cryptographic operations the server generated key for each cryptographic operations.

File Conversion

The selected file is sent encrypted using the selected cryptographic operations and generated keys. The input and output buffer size are extracted in each step. The cryptographic operations are performed based on the selection of the user.

Path Selection

After finding the total number of intermediate nodes, we have to find out the number of possible paths between the Transmitter and receiver .To find the number of possible path and identify the best path we use Swarm Optimization method.Then After finding the best path we will transmit the data to the receiver.

Extract Cryptographic Operation

The sender transfer the encrypted file to the receiver. After receiving the file the receiver determines the cryptographic operation in its execution. The receiver pinpoints the location of all the cryptographic functions, their respective mode and the order of execution.Extract the input and output of each cryptographic functions.

Offline Analysis

The extracted cryptographic operations are executed. Here the keys are extracted for each cryptographic operation and their input and output buffers size are verified. The module the mode of operation for the cipher encryption also extracted which in turn decrypts the file based on the mode of operation executed earlier on server side.

IV. CONCLUSION

A novel framework related to binary analysis called CipherXRay to extract the transient secrets and the cryptographic operations. In this proposed framework they are utilizing the characteristic of the avalanche effect. It is mainly used to detect the public key cryptography, block cipher and also the hash operations. The proposed framework is able to accurately pinpoint the cryptographic input, output and the keys which are available in the memory buffers. We have practically evaluated the effectiveness of the proposed CipherXRay technique by this we get the better analyze typical malwares protected by strong cryptographic algorithms.

REFERENCES

[1] C.K. Andreas Moser and E. Kirda, "Exploring Multiple Execution Paths for Malware Analysis," Proc. IEEE Symp. Security and Privacy (S&P '07), pp. 231-245, May 2007.
[2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (Im)Possibility of Obfuscating Programs (Extended Abstract)," Proc. 21st Ann. Int'l Cryptology Conf. Advances in Cryptology (CRYPTO '01), pp. 1-18, 2001.
[3] D. Brumley and D. Boneh, "Remote Timing Attacks Are Practical," Proc. 12th USENIX Security Symp., pp. 1-14, 2003.
[4] J. Caballero, N.M. Johnson, S. McCamant, and D. Song, "Binary Code Extraction and Interface Identification for Security Applications," Proc. 17th Network and Distributed System Security Symp. (NDSS '10), Feb. 2010.
[5] J. Caballero, P. Poosakam, S. McCamant, D. Babi_c, and D. Song, "Input Generation via Decomposition and Re-Stitching: Finding Bugs in Malware," Proc. 17th ACM Conf. Computer and Comm. Security (CCS '10), pp. 413-425, Oct. 2010.
[6] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-engineering," Proc. 16th ACM Conf. Computer and Comm. Security (CCS '09), pp. 621-634, Oct. 2009.
[7] J. Caballero and D. Song, "Polyglot: Automatic Extraction of Protocol Format Using Dynamic Binary Analysis," Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), pp. 317-329, Oct.
2007.
[8] L. Cavallaro, P. Saxena, and R. Sekar, "On the Limits of Information Flow Techniques for Malware Analysis and Containment," Proc. Fifth Int'l Conf. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '08), pp. 143-163, July 2008.
[9] L. Cavallaro, P. Saxena, and R. Sekar, "On the Limits of Information Flow Techniques for Malware Analysis and Containment," Proc. Fifth Int'l Conf. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '08), pp. 143-163, July 2008.

[10] P.M. Comparetti, G. Salvaneschi, E. Kirda, C. Kolbitsch, C. Kruegel, and S. Zanero, "Identifying Dormant Functionality in Malware Programs," Proc. IEEE Symp. Security and Privacy (S&P '10), pp. 61-76, May 2010.
[11] P.M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol Specification Extraction," Proc. IEEE Symp. Security and Privacy (S&P 2009), pp. 110-125, May 2009.
[12] W. Drewry and T. Ormandy, "Flayer: Exposing Application Internals," Proc. First USENIX Workshop Offensive Technologies (WOOT '07), Aug. 2007.
[13] T. Duong and J. Rizzo, "Cryptography in the Web: The Case of Cryptographic Design Flaws in ASP.NET," Proc. IEEE Symp. Security & Privacy (S&P '11), pp. 481-489, May 2011.
[14] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song, "Dynamic Spyware Analysis," Proc. USENIX Ann. Technical Conf. (ATC '07), pp. 233-246, June 2007.
[15] K. Gandolfi, C. Mourtel, and F. Olivier, "Results of Electromagnetic Analysis," Proc. Third Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01), May 2001.
[16] F. Gro¨ bert, "Automatic Identification of Cryptographic Primitives in Software," Deplima thesis, Ruhr-Univ. Bochum, Germany, Feb.2010.
[17] T. Pettersson, "Cryptographic Key Recovery from Linux Memory Dumps," Presentation, Chaos Communication Camp, Aug. 2007.
[18] I. Popov, S. Debray, and G. Andrews, "Binary Obfuscation Using Signals," Proc. 16th USENIX Security Symp., pp. 275-290, Aug. 2007.
[19] A. Shamir and N. van Someren, "Playing Hide and Seek with Stored Keys," Proc. Third Int'l Conf. Financial Cryptography (FC '99), pp. 118-124, Feb. 1999.
[20] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, "Impeding Malware Analysis Using Conditional Code Obfuscation," Proc. 15th Network and Distributed System Security Symp. (NDSS '08), Feb. 2008.
[21] S.K. Udupa, S.K. Debray, and M. Madou, "Deobfuscation: Reverse Engineering Obfuscated Code," Proc. 12th Working Conf. Reverse Eng. (WCRE '05), Nov. 2005.
[22] T. Wang, T. Wei, G. Gu, and W. Zou, "TaintScope: A Checksum- Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection," Proc. IEEE Symp. Security and Privacy (S&P '10), pp. 497-512, May 2010.
[23] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace, "ReFormat: Automatic Reverse Engineering of Encrypted Messages," Proc. 14th European Symp. Research in Computer Security (ESORICS '09),
pp. 200-215, Sept. 2009.
[24] W. Yan, Z. Zhang, and N. Ansari, "Revealing Packed Malware," IEEE Security and Privacy, vol. 6, no. 5, pp. 65-69, Oct. 2008.
[25] H. Yin, D. Song, M. Egele, E. Kirda, and C. Kruegel, "Panorama: Capturing System-Wide Information Flow for Malware Detection and Analysis," Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), pp. 497-512,Oct.2007

