2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) | 979-8-3503-8199-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/TrustCom60117.2023.00116

Hydamc: A Hybrid Detection Approach for Misuse of Cryptographic Algorithms in Closed-Source Software

Haoling Fan[a,b,c], Fangyu Zheng[d✉], Jingqiang Lin[e],
Lingjia Meng[a,b,c], Mingyu Wang[a,b,c], Qiang Wang[a,b,c], Shijie Jia[a,b,c], Yuan Ma[a,b,c]
[a] *State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences,* Beijing, China
[b] *Data Assurance and Communication Security Research Center, Chinese Academy of Sciences,* Beijing, China
[c] *School of Cyber Security, University of Chinese Academy of Sciences,* Beijing, China
[d] *School of Cryptography, University of Chinese Academy of Sciences,* Beijing, China
[e] *School of Cyber Security, University of Science and Technology of China,* Hefei, China
{fanhaoling, zhengfangyu, menglingjia, wangmingyu, wangqiang3113, jiashijie, mayuan}@iie.ac.cn, {linjq}@ustc.edu.cn

***Abstract*—Cryptographic algorithms are fundamental to secure software development, but security vulnerabilities can arise during implementation, usage, and when calling third-party libraries. As security standards continue to evolve, software updates have become an inevitable trend, and detecting cryptographic algorithm misuse is crucial to ensure compliance with these standards during the update process. However, closed-source software presents challenges in detecting cryptographic algorithm misuse. To enhance the security ecosystem of software, we designed a hybrid detection approach for detecting misuses in closed-source software related to weak cryptographic algorithms, short keys, insecure working modes, and insecure padding modes. Our hybrid detection tool uses both static and dynamic detection methods to collect log information through a logging mechanism in binary executable files. The collected data is cleaned using a data cleaning strategy and analyzed to extract key features, generating test reports to help developers and experts identify cryptographic algorithm security issues. We tested 24 software applications from app stores and found that 62.5% had weak algorithm implementations or usage, 83.3% supported short keys, and 50% supported insecure padding modes. Finally, we provided actionable recommendations to mitigate identified issues.**

***Index Terms*—Cryptography algorithm, Closed source software, Misuse detection, Instrumentation**

I. Introduction

Due to the increasing number of attacks, growing vulnerabilities, and advancements in computing power, information security has become dire. Cryptographic algorithms play a vital role in information security as they ensure confidentiality through encryption techniques, maintain data integrity, authenticate participants, and enable non-repudiation [1].

Software security protection widely applies cryptographic algorithms, whether open source or closed source. Closed-source software accounts for a considerable proportion in quantity, mainly due to the influence of multiple factors such as business confidentiality requirements, profit motives, security considerations, and intellectual property protection. With the continuous iteration of security standards, such as the Commercial National Security Algorithm Suite 2.0 (CNSA 2.0) released by the US National Security Agency in September 2022 in response to the threats posed by quantum computing, software updates have become an inevitable trend. Detection of cryptographic algorithms becomes critical to ensure compliance with these standards during the update process.

In addition to the need to detect cryptographic algorithms due to software updates, it is important to pay attention to the misuse scenarios of cryptographic algorithms, including the following three aspects:

- **Insecure algorithm implementations:** Some algorithm implementations have insecure default padding modes, compromising security [2].
- **Usage risks:** Security risks arise if short keys, insecure padding modes, insecure modes of operation, or deprecated cryptographic algorithms are employed when using cryptographic algorithms [3].
- **Third-party cryptographic algorithm library risks:** Insufficient attention to the security of these libraries by developers can lead to security vulnerabilities [4]. Suppose there are defects or vulnerabilities in the third-party cryptographic algorithm library, and developers fail to update the library version promptly on time. They may be exposed to known vulnerabilities. Furthermore, malicious developers may intentionally insert backdoors or malicious code into cryptographic algorithm libraries. In that case, thereby jeopardizing system security [5].

A. Motivation

It is necessary to detect the misuse of cryptographic algorithms in the software. In the case of open-source software, direct code review and analysis of algorithm implementation can identify potential vulnerabilities and weaknesses since the source code is publicly available and often accompanied by detailed documentation and explanations. There have been many methods for detecting misuse of cryptographic algorithms for open-source software, such as using tools like Gerrit for web-based code review [6] and CodeScene, which conducts code analysis on repositories over time [7].

This work is supported in part by the National Key R&D Program of China under Award (No.2017YFB0802103) and National Natural Science Foundation of China (No.61902392). The corresponding author is Fangyu Zheng.

2324-9013/23/$31.00 ©2023 IEEE
DOI 10.1109/TrustCom60117.2023.00116

Closed-source software challenges information security in the industrial sector due to legacy systems, technologies, and cost considerations. Cryptographic algorithm misuse detection is difficult since the source code is inaccessible and needs more transparency. Furthermore, developers of closed-source software often conceal algorithm details and use technical protection measures to prevent reverse engineering and code analysis, increasing detection challenges.

Detecting issues in closed-source software can be challenging due to the unavailability of source code, making it difficult to access complete information. While static analysis techniques using characteristic data matching can identify potential vulnerabilities and security issues without executing the software, this approach has limitations, such as high false negative and false positive rates. We propose a hybrid approach that combines static and dynamic analysis techniques to address these challenges.

Dynamic analysis complements the limitations of static analysis by providing a powerful supplementary method. By monitoring software behaviour during execution, dynamic analysis can collect comprehensive and accurate information on function calls and their parameters related to cryptographic algorithms used in the software. This enables precise behaviour detection of misuse of the encryption algorithms used. Dynamic analysis is precious for closed-source software incorporating third-party cryptographic algorithm libraries, as it can identify specific algorithms used and potential security vulnerabilities.

The hybrid detection approach for closed-source software utilizes a logging mechanism to collect static and dynamic logs related to implementing cryptographic algorithms in the binary executable file. These logs capture successful matches of cryptographic algorithm implementations, function calls associated with cryptographic algorithms, parameter data, and other critical information. The log data is then processed using data filtering strategies to remove redundant information and extract key features. The results of cryptographic algorithm misuse detection can aid software developers and security experts in understanding and addressing the identified security issues and enhance overall software security.

B. Contributions

We have made the following contributions to the detection of cryptographic algorithms in closed-source software:

- We proposed a hybrid detection method Hydramc that combines static and dynamic analysis techniques to improve the accuracy of cryptographic algorithm misuse detection.
- We implemented a hybrid detection system. This system extends our previous black-box detection method based on feature data and achieves dynamic detection of calls to third-party cryptographic algorithm libraries.
- We analyzed 24 software applications using the hybrid detection system. The results revealed security issues when implementing and using cryptographic algorithms in all tested software applications, as well as when calling third-party encryption algorithm libraries.

C. Paper Structure

Section 1 introduced the research problem, significance, and objectives. Section 2 conducted a comprehensive literature review to examine relevant theories and previous research to support the present study. Section 3 outlined the research methods and data analysis techniques. Section 4 presented the research findings based on the research objectives. Section 5 summarizes the significance and conclusions of the study and makes suggestions for future research.

II. Preliminaries

This section explores instances of cryptographic algorithm misuse scenarios as well as existing cryptographic algorithm misuse detection methods, and highlights the pivotal role of binary detection methods in identifying and addressing such misuse, providing a comprehensive understanding of the challenges and effective detection techniques in ensuring software security.

A. Cryptographic Algorithm Misuse

Cryptographic algorithms play a crucial role in ensuring software security. Various international organizations and institutions have developed comprehensive cryptographic standards to promote the secure utilization of cryptographic algorithms. Notable examples include the National Institute of Standards and Technology (NIST) [8] and the International Organization for Standardization (ISO) [9] [10], each providing its own set of cryptographic standards. Building upon these established standards, the present study aims to identify instances of cryptographic algorithm misuse by focusing on vulnerabilities associated with weak algorithms, short key lengths, insecure working modes, and insecure padding methods.

1) Weak Cryptographic Algorithms: Weak cryptographic algorithms are encryption algorithms that exhibit poor security and are vulnerable to attacks. The insecurity of these algorithms may be due to a minor key space or overly simplistic encryption methods. In this article, we identified weak cryptographic algorithms listed in TABLE I.

2) Short Key Length: Key length is a crucial parameter for ensuring security in symmetric cryptography, as longer key lengths typically correspond to higher levels of security. In practical applications, the choice of key length should consider both security requirements and system performance. Different cryptographic algorithms have varying key length requirements.

For symmetric cryptographic algorithms, the key length primarily determines security. NIST recommends a key length of at least 192 or 256 bits for transmitting confidential information, with AES-256 being the current standard due to advances in computational power [16].

In public-key cryptographic algorithms, key length affects security and algorithm performance. Generally, longer key lengths provide enhanced security but may also impact performance. The choice of key length is based on specific requirements. For instance, RSA recommends a key length of at least 2048 bits [17], while ECC suggests a key length of at least 256 bits [18].

TABLE I
LIST OF WEAK CIPHER ALGORITHMS

Algorithm Name	Attack Category	Security Description
MD2	1, 2	IETF released RFC 6149 in 2011, announcing that MD2 is no longer recommended.
MD4	1, 2	IETF released RFC 6150 in 1996, announcing that MD4 algorithm has serious security loopholes.
MD5	1, 2	IETF released RFC 6151 in 2017, declaring that MD5 is not suitable for specific security applications.
SHA-1	1, 2	NIST announced in 2011 that SHA-1 is no longer suitable for applications using digital signatures [12].
DES	3, 4, 5	NIST announced in 2005 that DES was no longer considered a secure encryption algorithm [13].
3DES	3, 4	NIST announced in 2017 that 3DES is no longer considered a secure encryption algorithm [14].
RC2	4, 5	NIST announced in 2017 that the RC2 algorithm is no longer recommended [14].
RC4	6, 7	IETF released RFC 7465 in 2015, announcing that the RC4 algorithm is not recommended.
RC5	4, 5	NIST announced in 2017 that the RC5 algorithm is no longer recommended [15].
CAST-128	4, 5	NIST announced in 2012 that the CAST-128 algorithm was deprecated.
IDEA	4, 5	NIST announced in 2017 that IDEA is no longer considered a secure encryption algorithm [14].
Blowfish	4, 5	NIST announced in 2017 that Blowfish is no longer considered a secure encryption algorithm [14].
SEED	4, 5	KISA announced in 2013 that the SEED algorithm is not recommended [28].

1: Collision attack. 2: Preimage attack. 3: Exhaustive attack. 4: Differential attack.
5: Linear attack. 6: Key stream deviation attack. 7: statistical analysis attack.

Hash algorithm security depends on hash value length, with longer hash values providing greater security but potentially affecting algorithm performance. SHA-1 is insufficiently secure for digital signatures below 80 bits. NIST and ISO have established standards and guidelines for evaluating and selecting hash functions to ensure hash algorithm security. Secure hash algorithms commonly include SHA-256 and SHA-3 [19].

3) Insecure Working Modes: Insecure working modes refer to operational ways in block ciphers that may lead to security issues, such as leakage of ciphertext. The NIST SP 800-38A [14] standard describes the encryption and decryption processes, security requirements, and usage considerations for each working mode. The Electronic Codebook (ECB) mode is vulnerable due to the generation of identical ciphertext blocks for the same plaintext blocks, making it susceptible to analysis and attacks. In contrast, Cipher Block Chaining (CBC) mode offers better security, albeit with limited support for parallel encryption and decryption operations. Cipher Feedback (CFB) and Output Feedback (OFB) modes allow encryption at the byte level and support partial block encryption operations but have the risk of error propagation. By comparison, Counter (CTR) mode supports stream ciphers, parallel encryption and decryption operations and provides higher security.

4) Insecure Padding Modes: Padding modes are used in encryption algorithms to pad the plaintext to meet the block length requirements of the algorithm. Many cryptographic algorithms, such as RSA, AES, and DES, use padding modes in practical applications. Selecting an appropriate padding mode is crucial, as an insecure padding pattern can provide an attacker with an opportunity to attack. Several common padding modes include Zero padding, PKCS#5/PKCS#7 padding, ANSI X.923 padding, ISO 10126 padding, ISO/IEC 7816-4 padding, OAEP padding, PSS padding, and RSA's No Padding mode. Each padding mode has different characteristics and levels of security.

For a clearer understanding of the characteristics and safety of each fill mode, TABLE II below provides a concise summary.

B. Binary Detection Methods

Commonly used methods for detecting algorithms in binary executable files include matching based on characteristic data and instrumentation techniques.

Matching based on characteristic data is a method of detection that involves searching for known patterns of cryptographic algorithms in binary files [20]. These patterns can be specific function call sequences, constant values, working modes, etc. Detection using matching based on characteristic data can be performed through static analysis tools such as disassemblers, pattern-matching tools, or custom scripts. This method is suitable for detecting known cryptographic algorithms but may not be effective for customized or modified algorithms.

Instrumentation techniques involve inserting additional code into binary files to monitor and record the execution process and data flow of cryptographic algorithms [21]. By inserting logging code, key tracking code, or detection code into the cryptographic algorithm functions, details of the algorithm's execution can be obtained. Instrumentation techniques can be implemented using static or dynamic analysis tools, such as dynamic debuggers, dynamic instrumentation frameworks, or custom binary modification tools. Instrumentation techniques provide more detailed information, including input/output data, encryption/decryption processes, and the keys used, which helps in analyzing and evaluating the correctness and security of cryptographic algorithms.

These methods can be combined, and the appropriate method can be chosen based on specific requirements and scenarios for detecting cryptographic algorithms in binary executable files.

1) Cryptographic Algorithm Misuse Detection: Misuse detection of cryptographic algorithms is a crucial research area in computer science. To detect and evaluate the security of cryptographic algorithms in software applications, researchers employ static and dynamic analysis methods.

Common static analysis tools such as x3chun-Crypto-Searcher [22], HCD [23], draca [24], and Krypto Analyzer [25] analyze binary or source code to identify patterns or instruction sequences related to specific cryptographic algorithms. These

TABLE II
CRYPTOGRAPHIC ALGORITHM PADDING MODE SECURITY DESCRIPTION

Padding Mode	Characteristics	Security
Zero Padding	Simple padding, adds zero bytes to the end of the block	Insecure, may lead to information leakage
PKCS#5/PKCS#7 Padding	Widely used, ensures integrity and uniqueness of padding	Secure, with provable security properties
ANSI X.923 Padding	Adds zero bytes at the end, with second-to-last byte indicating the number of padding bytes, relies on random numbers	Security risks due to reliance on random numbers
ISO 10126 Padding	Adds random bytes at the end, with last byte indicating the number of padding bytes, relies on random numbers	Security risks due to reliance on random numbers
ISO/IEC 7816-4 Padding	Adds non-zero and zero bytes at the end, with second-to-last byte indicating the number of padding bytes, relies on random numbers	Security risks due to reliance on random numbers
OAEP Padding	Used for public key encryption, introduces randomness and complexity, provides higher security	Provides higher security
PSS Padding	Used for digital signatures, introduces randomness and complexity, provides higher security	Provides higher security
No Padding (RSA)	No padding applied, requires additional security measures for data protection	Insecure, may lead to information leakage

tools detect the presence and properties of cryptographic algorithms based on predefined signatures or features, offering high efficiency and fast detection.

Dynamic analysis tools use simulation techniques or program instrumentation to track the runtime behaviour of a program. Felix Gröbert's work [26], for instance, employs the PIN tool for dynamic tracing and combines signature-based search with simple memory reconstruction techniques to identify specific encryption algorithms by observing program behaviour and data flow. The DynamoRIO framework [27] provides a flexible, dynamic binary instrumentation platform that can be used to analyze program execution behaviour and data flow.

To address cryptographic algorithm detection in closed-source software, we propose a hybrid detection tool that combines static and dynamic techniques. This tool leverages the strengths of both static and dynamic approaches to provide a more comprehensive analysis and identification of the presence and characteristics of cryptographic algorithms. It aims to deliver a more accurate detection of cryptographic algorithm misuse.

III. DESIGN AND IMPLEMENTATION

We have designed and implemented a hybrid detection system combining static and dynamic analysis to achieve security detection of cryptographic algorithms in closed-source software. This system can detect weak cryptographic algorithms, short keys, insecure working modes, and insecure padding modes, which is divided into the following three modules:

- **Data Collection Module:** This module collects static logs (Log-S) using an extended black-box static tool and dynamic logs (Log-D) using an instrumentation-based dynamic detection method. The log data serves as the data source for the entire system.
- **Data Filtering Module:** This module is responsible for processing the log data collected by the Data Collection Module. The minimum unit of cryptographic algorithm-related data in the log data is defined as a *log primitive*. The log primitives are classified, and redundant data is removed according to the data filtering strategy to obtain key information.
- **Report Generation Module:** This module generates security detection reports, presenting the detection results to developers and researchers in specific content and format.

A. Data Collection Module

By executing the data collection module, static logs (Log-S) acquired through black-box static detection and dynamic logs (Log-D) obtained through dynamic detection with instrumentation are collected as the system's data sources.

1) Black-box Static Detection: Using static analysis techniques, executable binary files of software can be examined to determine the implementation of cryptographic algorithms. As a static analysis tool that relies on characteristic data of cryptographic algorithms, the cryptographic algorithm black-box testing tool can detect the implementation of cryptographic algorithms even in closed-source software [20]. We extended the capabilities of the cryptographic algorithm black-box detection tool by:

- Incorporating the characteristic data of post-quantum cryptography algorithms into the configuration file of the black-box detection tool, enabling the detection of post-quantum cryptography algorithms in response to the increasing computational capabilities. Post-quantum cryptography aims to develop secure cryptographic systems against quantum and classical computers and can interoperate with existing communication protocols and networks. The "CRYSTALS" (Cryptographic Suite for Algebraic LatticeS) package contains Kyber and Dilithium. Kyber is a key encapsulation mechanism resistant to adaptive chosen-ciphertext attacks (IND-CCA2). At the same time, Dilithium is a highly unforgeable digital signature algorithm that is secure against existential forgery under a chosen message attack (EUF-CMA). We have added detection for the post-quantum cryptographic algorithms Kyber and Dilithium. Kyber uses a tree table for bit reverse indexing during NTT transformation, which can be used as characteristic data for detection. Dilithium uses a zetas

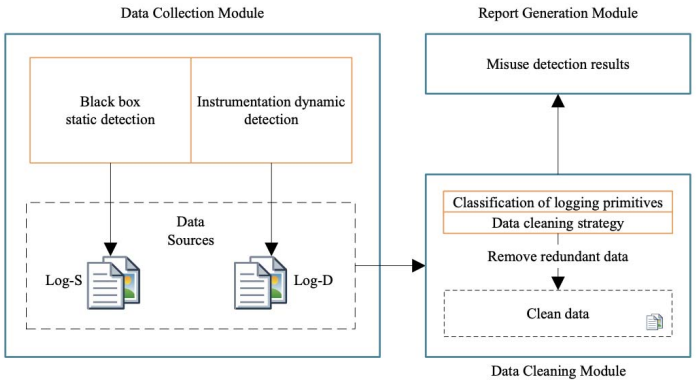

Fig. 1. Security Evaluation System Architecture

table during NTT transformation, which can also be used for detection.

These extensions broaden the detection scope of the cryptographic algorithm black-box detection tool, thereby enhancing its effectiveness in identifying the presence of quantum cryptography algorithms and analyzing the characteristics of different padding modes in cryptographic algorithm implementations.

2) Instrumentation Dynamic Detection: In dynamic instrumentation, we conduct source code instrumentation on a third-party cryptographic algorithm library by inserting additional code into its source code. This enables the instrumented dynamic link library to be invoked during software execution, allowing for dynamic monitoring and analysis of the software's interaction with the cryptographic algorithm library. As a result, relevant data regarding the software's utilization of the cryptographic algorithm library can be retrieved.

OpenSSL is a widely used and trusted cryptographic library for instrumentation purposes due to its long history, high functionality, cross-platform compatibility, and open source nature that allows for thorough scrutiny. We selected OpenSSL as the third-party cryptographic algorithm library for instrumentation, and below are the specific details of the instrumentation process:

- **Insertion position:** We select the cryptographic algorithm implementation files involved in the cryptographic algorithm library, and the insertion position is the initial position of the function body of each cryptographic algorithm-related function in the file.
- **Insertion method:** We have written an automatic instrumentation tool, which first traverses each cryptographic algorithm-related function in the implementation file of all cryptographic algorithms, then locates the initial position of each function body, and finally inserts the output statement at the instrumentation position, so that the function information is output when the function is executed.
- **Insertion content:** The instrumentation content includes the output function execution time statement, the output function name statement, and the output function parameter statement.
- **Deployment:** We recompile and install the instrumented OpenSSL, and if the software calls functions related to cryptographic algorithms in OpenSSL during software execution, the information will be output as the log content.

B. Data Filtering Module

We process the log data collected through black-box static and dynamic detection with instrumentation to obtain cryptographic algorithm-related data necessary for generating the detection report. Examples of the static log data (Log-S.txt) and dynamic log data (Log-D.txt) are as follows:

```
Log-S.txt
File Name: test/test.so
NO.1    SHA1
NO.2    SHA256
NO.3    SHA512
NO.4    SHA3
NO.5    MD4
NO.6    MD5

Log-D.txt
Start Timestamp: 0.006446943 s
Function: ERR_load_EC_strings
Start Timestamp: 0.007866163 s
Function: AES_set_encrypt_key
Param (const int): bits = 256
Param (AES_KEY): *key = 289616208
```

In Log-S.txt, the "File Name" field indicates the name of the currently analyzed binary executable file. This is followed by a list of cryptographic algorithms obtained through characteristic data matching. We have modified the black-box detection tool to output the matched characteristic data, thereby obtaining a collection of successful characteristic data matches for each cryptographic algorithm.

In Log-D.txt, the "Start Timestamp" denotes the start time of function execution when calling the third-party library. "Function" represents the name of the called function. The content within parentheses after "Param" indicates the parameter types of the function. Parameter names are listed after the colon, and

parameter values are provided after the equal sign. Parameters marked with an asterisk (*) can only represent addresses.

We define the log primitive as the smallest unit of cryptographic algorithm-related data in the log data. Log primitives can be classified into characteristic data, cryptographic algorithm names, function names, parameter data types, parameter names, and parameter values.

In the data filtering module, we propose a data filtering strategy to remove redundancy from the log data:

- **Weight**: In the log primitives, we assign high weights to function names and parameter values, medium weights to cryptographic algorithm names and characteristic data, and low weights to parameter data types. This is because dynamic detection provides information about function calls during software execution. In contrast, static detection can only detect cryptographic algorithms implemented in the binary executable file, which may not necessarily be executed. Therefore, function names and parameter values from dynamic logs have higher weights than characteristic data and cryptographic algorithm names from static logs. Parameter data types, as auxiliary log primitives for detection, are assigned low weights.
- **Grouping**: Grouping setting involves grouping function name data with relational dependencies in dynamic logs and establishing a group whitelist. When a specific cryptographic algorithm is called, a certain number of function calls within the same group are expected to be invoked simultaneously. If at least one function within the group is not reached, we consider that the corresponding cryptographic algorithm in that group has not been invoked. For example, when SHA256 is called, it requires the functions SHA256_Init, SHA256_Update, and SHA256_Final to be called.
- **Sequence**: Sequence setting involves sorting a group of function names with sequential dependencies in dynamic logs based on their call order. If a group of functions is not called according to the sequence setting, we consider that the corresponding cryptographic algorithm in that group has not been invoked. For example, when AES in CBC mode is called, it should first gather AES_set_encrypt_key and then AES_cbc_encrypt. If the order is reversed, we conclude that AES in CBC mode has not been invoked.

During the data filtering process, with the log data as input, we first classify the log primitives and then apply the data filtering strategy to remove redundant data. The specific process is illustrated in Fig. 2.

C. Report Generation Module

The primary function of the report generation module is to compile the misuse detection results into a comprehensive report, enabling developers and researchers to understand the issues related to the implementation, usage, and integration of third-party cryptographic algorithm libraries. The report provides detailed security explanations, specific problem descriptions, and recommendations for resolution. The exact contents are as follows:

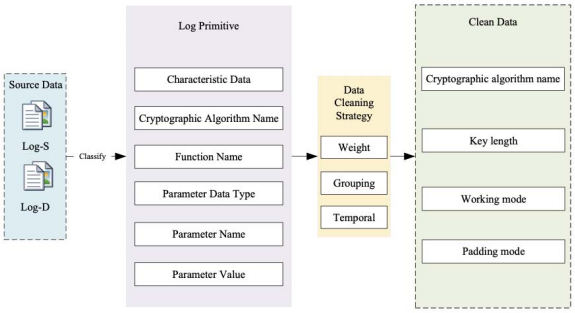

Fig. 2. Data Filtering Process

- **Comprehensive Security Explanations:** The hybrid detection system comprehensively evaluates the software to address factors such as weak cryptographic algorithms, short key lengths, insecure modes of operation, and padding schemes.
- **Analysis of Security Factors:** Manually analyze the potential risks and threats to the software by each security factor. Evaluate the effectiveness of existing security measures and protective mechanisms and their impact on overall security.
- **Recommendations for Resolution:** Provide corresponding solutions and improvement suggestions manually based on the specific problem descriptions listed in the report.

IV. Evaluation

In this section, we assess the security of various software applications by employing a hybrid detection tool that combines static and dynamic analysis techniques. We developed a Python-based instrumentation tool to perform instrumentation on the OpenSSL 1.1.1 cryptographic library. Subsequently, we deployed and executed a cryptographic algorithm hybrid detection tool on the Ubuntu 18.04 operating system. We harnessed Python's flexibility and Ubuntu's stability for tool development and maintenance. Given the critical reliance of numerous applications and systems on OpenSSL for secure communication and data protection, we selected OpenSSL as the cryptographic library to be instrumented.

A. Test Set Description

The test set used for evaluation is derived from keyword-based searches conducted within software marketplaces, utilizing terms such as "secure storage," "secure communication," and "encryption." This search approach enables us to identify and categorize software applications into five distinct types: password managers, disk encryption software, compression tools, cloud storage encryption software, and instant messaging tools. To provide a comprehensive overview, we present TABLE III that includes the name, serial number, and category information for each software application.

TABLE III
TEST SET DESCRIPTION

Category	No.	Name	Category	No.	Name
Compression tool	1	7-Zip	Password manager	13	KeePassXC
	2	WinZip		14	Bitwarden
	3	WinRAR		15	Password Safe
	4	PeaZip		16	Buttercup
	5	BitZipper		17	LastPass
	6	HaoZip	Encrypted cloud storage service	18	Cryptomator
	7	Bandizip		19	Boxcryptor
	8	B1 Free Archiver		20	SpiderOak
	9	Bitser	Instant messaging tool	21	Telegram
Disk encryption tool	10	VeraCrypt		22	Pidgin
	11	DiskCryptor		23	Signal
	12	FreeOTFE		24	Jitsi

TABLE IV
CLASSIFICATION OF TEST RESULTS

Misuse Category	Risky software serial number
Implement or use weak cryptographic algorithms	1, 2, 3, 4, 5, 6, 7, 10, 13, 14, 15, 16, 17, 20, 22
Support for short keys	1, 2, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 23, 24
Support for unsafe work modes	1, 2, 3, 4, 7, 8, 9, 14, 15, 17, 18, 19

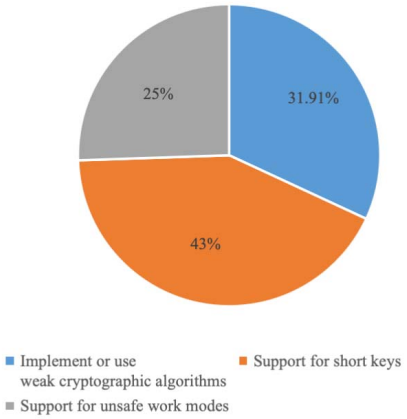

Fig. 3. Statistical Analysis of Software Quantity for Each Security Issue

B. Test Results

We tested the software in the test set using the hybrid detection tool for cryptographic algorithm misuse. In the hybrid detection approach, we employed fuzzing techniques to explore the runtime paths of the software and trigger potential cryptographic algorithm invocations. The testing results primarily focused on detecting and analyzing security issues encountered during the evaluation process. Specifically, we categorized the issues into the following classes: weak encryption algorithms, short encryption keys, insecure operation modes, and insecure padding schemes. For each problem category, we provide a TABLE IV containing software serial numbers and a statistical analysis graph of the number of software applications belonging to each category, as shown in Fig. 3. The statistical analysis indicates that short keys and weak cryptographic algorithms are more risky. Additionally, we provided detailed explanations for each identified security issue.

- **Weak encryption algorithms:** Our detection results indicate that SHA1, MD4, MD5, Blowfish, DES, XTEA and RC4 are frequently used in software applications, with 62.5% of the applications implementing at least one of them. These algorithms are vulnerable to various attacks, such as brute force attacks and encryption vulnerabilities, which pose significant security risks. It is crucial to replace these weak algorithms with more secure ones.
- **Short encryption keys:** We found that 83.3% of software applications use inappropriate key lengths, with a particular focus on 128-bit keys supporting AES, as only WinRAR has dropped the use of AES-128. As we consider the threat of quantum computing, choosing a 128-bit key may not be sufficient to protect message confidentiality from quantum attackers, making longer encryption keys necessary to ensure robust security.
- **Insecure operation modes:** We discovered that 50% of software applications use insecure modes of operation, with risks primarily arising from the use of CBC encryption mode. The primary security issue of CBC mode is Padding Oracle Attack, which can gradually deduce plaintext information by exploiting error information returned by the server during decryption. To provide better security and integrity protection, a more secure encryption mode, such as CTR mode, should be used.

Our hybrid detection tool has identified security vulnerabilities in various software applications. The test results revealed that all of the 24 applications in the test set had security issues. By categorizing and analyzing these issues, we have provided valuable insights into weaknesses in cryptographic implementations. We emphasize the importance of adopting secure algorithms, ensuring sufficient key lengths, using secure operation modes, and employing robust padding schemes in

software development to ensure data security and integrity.

To perform static analysis on the binary executable files in the test set, we used a black-box test tool [20] for cryptographic algorithms. False positive issues, which account for 16.7% of cases, may arise from the possibility of multiple cryptographic algorithms sharing common characteristic data. Additionally, false negative issues, which account for 75% of cases, may occur due to reduced discriminative power of characteristic data caused by different implementations of cryptographic algorithms and the inability to recognize cryptographic algorithms called from third-party libraries.

Compared to static analysis, the hybrid detection tool provides the following benefits:

- **More comprehensive detection results:** In addition to identifying weak cryptographic algorithms, the tool can detect issues related to short encryption keys, insecure operation modes, and insecure padding schemes.
- **Reduced false positive rate:** The tool can identify cryptographic algorithms that are invoked during software execution, rather than simply detecting which cryptographic algorithm implementations are present.
- **Reduced false negative rate:** The tool can also detect security issues in cryptographic algorithms called from third-party libraries.

V. Summary

We propose a hybrid cryptographic algorithm detection tool that combines static and dynamic analysis techniques for closed-source software. Our data filtering strategy has significantly improved the accuracy of the detection tool by removing redundant data. Evaluations conducted on software from a marketplace have revealed security vulnerabilities associated with third-party cryptographic libraries.

Developers and researchers should avoid using weak cryptographic algorithms like DES and MD5 and opt for validated alternatives such as AES and SHA-3. Balancing security and performance requirements is crucial when selecting key lengths and working modes. Future research may focus on security issues of other third-party cryptographic libraries, improving security alert systems, and exploring the impact of industry standards and policies on encryption algorithms. These recommendations support cryptographic library security, ensuring the construction of secure and reliable networks and information systems.

References

[1] Qadir A M, Varol N. A review paper on cryptography[C]//2019 7th international symposium on digital forensics and security (ISDFS). IEEE, 2019: 1-6.

[2] Nazaruks V, Rusakovs P. Implementation of Cryptographic Algorithms in Software: An Analysis of the Effectiveness[J]. Rigas Tehniskas Universitates Zinatniskie Raksti, 2010, 43: 97.

[3] Wickert A K, Reif M, Eichberg M, et al. A dataset of parametric cryptographic misuses[C]//2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). IEEE, 2019: 96-100.

[4] Wang Q, Li J, Zhang Y, et al. NativeSpeaker: Identifying crypto misuses in Android native code libraries[C]//Information Security and Cryptology: 13th International Conference, Inscrypt 2017, Xi'an, China, November 3–5, 2017, Revised Selected Papers 13. Springer International Publishing, 2018: 301-320.

[5] Duan Y, Gao L, Hu J, et al. Automatic Generation of Non-intrusive Updates for Third-Party Libraries in Android Applications[C]//RAID. 2019: 277-292.

[6] Jigin D, Gunnesson O. Using Already Existing Data to Answer Questions Asked During Software Change[J]. LU-CS-EX 2018-29, 2018.

[7] Mukadam M, Bird C, Rigby P C. Gerrit software code review data from android[C]//2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013: 45-48.

[8] Brown D R L, Gjøsteen K. A security analysis of the NIST SP 800-90 elliptic curve random number generator[C]//Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27. Springer Berlin Heidelberg, 2007: 466-481.

[9] Accerboni F, Sartor M. ISO/IEC 27001[M]//Quality Management: Tools, Methods, and Standards. Emerald Publishing Limited, 2019.

[10] Standard A. ISO/IEC27002[M]//Informationtechnology-security techniques-code of practice for information security controls,(AS ISO/IEC 27002: 2015). 2015.

[11] Sindhu S, Sindhu D. Cryptographic algorithms: applications in network security[J]. International Journal of New Innovations in Engineering and Technology, 2017, 7(1): 11.

[12] Barker E, Roginsky A. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths[J]. NIST Special Publication, 2011, 800: 131A.

[13] Biham E, Shamir A. Differential cryptanalysis of DES-like cryptosystems[J]. Journal of CRYPTOLOGY, 1991, 4: 3-72.

[14] Dworkin M. NIST Special Publication 800-38A 2001 Edition[J]. NIST Special Publication, 2001, 800: 38A.

[15] Charbathia S, Sharma S. A comparative study of rivest cipher algorithms[J]. International Journal of Information & Computation Technology. ISSN, 2014, 4: 0974-2239.

[16] Bisht N, Singh S. A comparative study of some symmetric and asymmetric key cryptography algorithms[J]. International Journal of Innovative Research in Science, Engineering and Technology, 2015, 4(3): 1028-1031.

[17] Somani N, Mangal D. An improved RSA cryptographic system[J]. International Journal of Computer Applications, 2014, 105(16).

[18] Abood O G, Guirguis S K. A survey on cryptography algorithms[J]. International Journal of Scientific and Research Publications, 2018, 8(7): 495-516.

[19] Cid C. Recent developments in cryptographic hash functions: Security implications and future directions[J]. Information security technical report, 2006, 11(2): 100-107.

[20] Fan H, Meng L, Zheng F, et al. Black-Box Testing of Cryptographic Algorithms Based on Data Characteristics[C]//Applied Cryptography in Computer and Communications: Second EAI International Conference, AC3 2022, Virtual Event, May 14-15, 2022, Proceedings. Cham: Springer Nature Switzerland, 2022: 153-169.

[21] Gröbert F, Willems C, Holz T. Automated identification of cryptographic primitives in binary programs[C]//Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011. Proceedings 14. Springer Berlin Heidelberg, 2011: 41-60.

[22] x3chun. Crypto searcher, 2004. https://web.archive.org/web/20050211180634/http://x3chun.wo.to/.

[23] Mr Paradox, AT4RE. Hash & crypto detector (hcd), 2009. https://web.archive.org/web/20091203010936/http://www.at4re.com/download.php?view.8.

[24] Literatecode. Draft crypto analyzer (draca). http://www.literatecode.com/draca, May 2013.

[25] snaker, Maxx. Kanal - krypto analyzer for peid, 2015. http://www.dcs.fmph.uniba.sk/zri/6.prednaska/tools/PEiD/plugins/kanal.htm.

[26] Gröbert F. Automatic identification of cryptographic primitives in software[J]. Ruhr-University Bochum, 2010, 115.

[27] Yan F, Xing Y, Zhang S, et al. Research on Cryptographic Algorithm Recognition Based on Behavior Analysis[C]//Trusted Computing and Information Security: 11th Chinese Conference, CTCIS 2017, Changsha, China, September 14-17, 2017, Proceedings. Springer Singapore, 2017: 370-382.

[28] Lu J, Yap W S, Henricksen M, et al. Differential attack on nine rounds of the SEED block cipher[J]. Information Processing Letters, 2014, 114(3): 116-123.