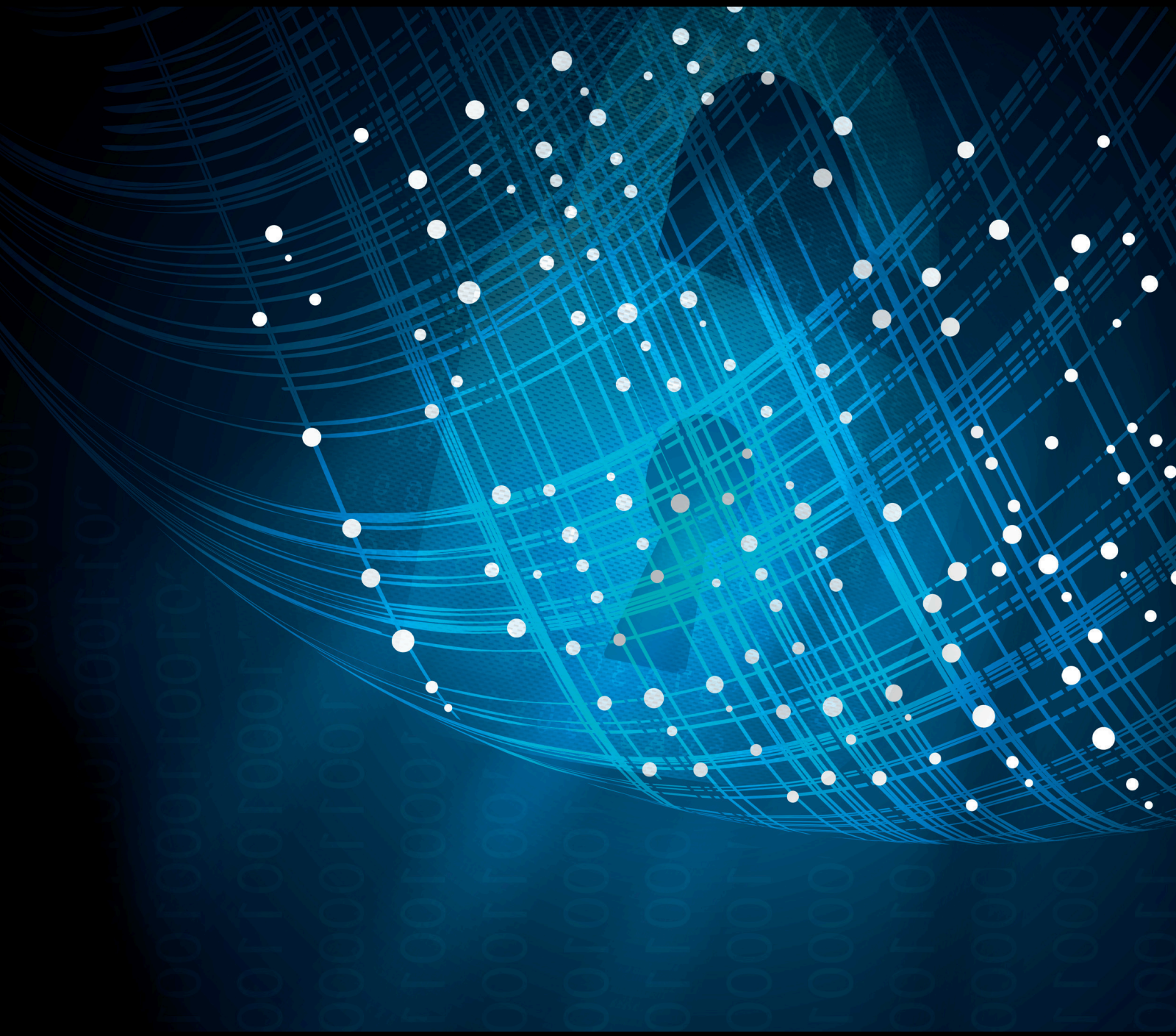
Security and Communication Networks

AI-Driven Cyber Security Analytics and Privacy Protection

Lead Guest Editor: Jiageng Chen
Guest Editors: Chunhua Su and Zheng Yan



AI-Driven Cyber Security Analytics and Privacy Protection

Security and Communication Networks

AI-Driven Cyber Security Analytics and Privacy Protection

Lead Guest Editor: Jiageng Chen
Guest Editors: Chunhua Su and Zheng Yan

Editorial Board

Contents

Hindawi
Security and Communication Networks
Volume 2019, Article ID 1859143, 2 pages
https://doi.org/10.1155/2019/1859143

Editorial

AI-Driven Cyber Security Analytics and Privacy Protection

Jiageng Chen,[1] **Chunhua Su**,[2] **and Zheng Yan**[3]

[1]*Central China Normal University, China*
[2]*University of Aizu, Japan*
[3]*Xidian University, China*

Correspondence should be addressed to Jiageng Chen; chinkako@gmail.com

Received 5 November 2019; Accepted 5 November 2019; Published 30 November 2019

The cyber security protection has gone through a rapid development in today's internet connected world. With the wide application of the booming technologies such as the Internet of Things (IoT) and the cloud computing, huge amount of data are generated and collected. While the data can be used to better serve the corresponding business needs, they also pose big challenges for the cyber security and privacy protection. It becomes very difficult if not impossible to discover the malicious behavior among the big data in real time. Thus, this gives rise to the cyber security solutions which are driven by AI-based technologies, such as machine learning, statistical inference, big data analysis, deep learning, and so on. AI-driven cyber security analytics has already found its applications in the next generation firewall which includes the automatic intrusion detection system, encrypted traffic classification, malicious software detection, and so on. In the area of cryptography, AI-driven solution starts to help the researchers optimize the algorithm design and can largely reduce the cryptanalysis effort such as searching the differential trails which is crucial in differential cryptanalysis. Recently, the idea of generative adversary network was applied to building the automatic encryption algorithm, which makes a first move towards making an intelligent protection solution without the interference of the human effort. On the contrary, individual's privacy is under threat given the AI-based systems. The rise of AI-enabled cyberattacks is expected to cause an explosion of network penetrations, personal data thefts, and an epidemic-level spread of intelligent computer viruses. Thus, another future trend is to defend AI-driven attacks by using AI-driven techniques, which will possibly lead to an AI arms race. AI-driven security solution is one of the fastest growing fields which bring together researchers from multiple areas such as machine learning, statistics, big data analytics, and cryptography to fight against the advanced cyber security threats. The purpose of this special issue is to present the cutting-edge research progress from both academia and industry, with a particular emphasis on the new tools, techniques, concepts, and applications concerning the AI-driven cyber security analytics and privacy protection. A brief summary of all the accepted papers is provided as follows.

In the paper by Y. Zhao et al., a novel feature extraction method of hybrid gram (H-gram) with cross entropy of continuous overlapping subsequences was proposed based on the dynamic feature analysis of malware, which implemented semantic segmentation of a sequence of API calls or instructions. The experimental results showed that the H-gram method can distinguish the malicious behaviors and is more effective than the fixed-length n-gram in all four performance indexes of the classification algorithms such as ID3, Random Forest, AdboostM1, and Bagging.

The paper by T. Hu et al. proposed a user authentication method based on mouse biobehavioral characteristics and deep learning, which can accurately and efficiently perform continuous identity authentication on current computer users to address insider threats. An open source dataset with ten users was applied to carry out experiments, and the experimental results demonstrated the effectiveness of the approach. The proposed approach can complete a user authentication task approximately every 7 seconds, with a false acceptance rate of 2.94% and a false rejection rate of 2.28%.

In the paper by G. Huang et al., the algorithm MFS_AN (mining fault severity of all nodes) was proposed to mine the

key nodes from the software network. A weighted software network model was built by using functions as nodes, with relationships as edges, and times as weight. By exploiting the recursive method, a fault probability metric FP of a function is defined according to the fault accumulation characteristic, and a fault propagation capability metric FPC of a function is proposed according to the fault propagation characteristic. Based on the FP and FPC, the fault severity metric FS was put forward to obtain the function nodes with larger fault severity in the software network. Experimental results on two real software networks showed that the algorithm MFS_AN can discover the key function nodes correctly and effectively.

The paper by H. Park proposed the Secure Information Sharing System (SISS) model with the main method as a group key cryptosystem. SISS figured out important problems of group key systems. (1) The newly developed equations for encryption and decryption can eliminate the rekeying and redistribution process for every membership change of the group, keeping the security requirements. (2) The new 3D stereoscopic image mobile security technology with AR (Augmented Reality) solved the problem of conspiracy by group members. (3) SISS used the reversed one-way hash chain to guarantee Forward Secrecy and Backward Accessibility (security requirements for information sharing in a group). It showed that the security analysis of SISS according to the Group Information-sharing Secrecy and experiment on the performance of SISS. As a result, SISS made it possible to securely share sensitive information from collaborative works.

The paper by Y. Zhao et al. addressed the problem of CCA secure public key encryption against after-the-fact leakage without NIZK proofs. To obtain security against chosen ciphertext attack (CCA) for PKE schemes against after-the-fact leakage attack (AFL), previous works followed the paradigm of "double encryption" which needs non-interactive zero knowledge (NIZK) proofs in the encryption algorithm. This paper presented an alternative way to achieve AFL-CCA security via lossy trapdoor functions (LTFs) without NIZK proofs. Formalization of definition of LTFs secure against AFL (AFLR-LTFs) and all-but-one variants (ABO) was given. Then, it showed how to realize this primitive in the split-state model. This primitive can be used to construct an AFLR-CCA-secure PKE scheme in the same way as the method of "CCA from LTFs" in traditional sense.

In the paper by J. Ren et al., a software buffer overflow vulnerability prediction method by using software metrics and a *decision tree* algorithm was proposed. First, the software metrics were extracted from the software source code, and data from the dynamic data stream at the functional level were extracted by a data mining method. Second, a model based on a *decision tree* algorithm was constructed to measure multiple types of buffer overflow vulnerabilities at the functional level. Finally, the experimental results showed that the method ran in less time than *SVM, Bayes, adaboost,* and *random forest* algorithms and achieved 82.53% and 87.51% accuracy in two different data sets.

In the paper by S. Zhao et al., a three-layer classifier using machine learning to identify mobile traffic in open-world settings was proposed. The proposed method had the capability of identifying the traffic generated by unconcerned apps and zero-day apps; thus, it can be applied in the real world. A self-collected dataset that contains 160 apps was used to validate the proposed method. The experimental results showed that the classifier achieved over 98% precision and produced a much smaller number of false positives than that of the state-of-the-art.

Conflicts of Interest

The guest editors declare that there are no conflicts of interest regarding the publication of the special issue.

Acknowledgments

We would like to express our gratitude to all authors who made this special issue possible. We hope this collection of articles will be useful to the scientific community. The launch of this special issue was supported in part by the National Natural Science Foundation of China under Grant no. 61702212 and the Fundamental Research Funds for the Central Universities under Grand no. CCNU19TS017.

Jiageng Chen
Chunhua Su
Zheng Yan

Hindawi
Security and Communication Networks
Volume 2019, Article ID 8357241, 8 pages
https://doi.org/10.1155/2019/8357241

WILEY | Hindawi

Research Article

CCA Secure Public Key Encryption against After-the-Fact Leakage without NIZK Proofs

Yi Zhao,[1,2] **Kaitai Liang**,[3] **Bo Yang**,[1,2] **and Liqun Chen**[3]

[1]*Computer Science, Shaanxi Normal University, 710119 West Changan Street 620, Xi'an, Shaanxi, China*
[2]*State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China*
[3]*Computer Science, University of Surrey, GU27XH Guildford, UK*

Correspondence should be addressed to Bo Yang; byang@snnu.edu.cn

Received 10 August 2018; Accepted 3 September 2019; Published 31 October 2019

Academic Editor: Francesco Gringoli

In leakage resilient cryptography, there is a seemingly inherent restraint on the ability of the adversary that it cannot get access to the leakage oracle after the challenge. Recently, a series of works made a breakthrough to consider a postchallenge leakage. They presented achievable public key encryption (PKE) schemes which are semantically secure against after-the-fact leakage in the split-state model. This model puts a more acceptable constraint on adversary's ability that the adversary cannot query the leakage of secret states as a whole but the functions of several parts separately instead of prechallenge query only. To obtain security against chosen ciphertext attack (CCA) for PKE schemes against after-the-fact leakage attack (AFL), existing works followed the paradigm of "double encryption" which needs noninteractive zero knowledge (NIZK) proofs in the encryption algorithm. We present an alternative way to achieve AFL-CCA security via lossy trapdoor functions (LTFs) without NIZK proofs. First, we formalize the definition of LTFs secure against AFL (AFLR-LTFs) and all-but-one variants (ABO). Then, we show how to realize this primitive in the split-state model. This primitive can be used to construct AFLR-CCA secure PKE scheme in the same way as the method of "CCA from LTFs" in traditional sense.

1. Introduction

In the past two decades, physical attacks which are capable of getting access to partial information of the secret state have become a serious threat to the security of cryptographic algorithms in practice. These attacks have moved far beyond the scope of traditional cryptography with an inherent assumption that no information of the secret key is leaked. Up till now, the branch of cryptography to treat this issue is highly motivated, which is called leakage resilient cryptography.

The first step to address leakage resilience systematically is formalizing the leakage attack in the traditional security notion. There are already several models in the existing works which describe leakage in different ways. Akavia et al. [1] modeled the leakage as the bounded output of an arbitrary function of secret states (bounded/relative leakage). Naor and Segev [2] presented an alternative description to allow leakage without length restriction. They measured the leakage by the induced decrease of the minimum entropy of the secret (noisy leakage). Under these formulations, some leakage resilient primitives are successfully designed, including signature schemes [3–5] and key agreement protocol [6, 7].

However, in the area of public key encryption (PKE), there is an inherent restriction in the security notion. Semantic security is always defined to be the indistinguishability of the challenge ciphertext issued by an adversary in a game with a challenger answering different types of queries from the adversary. The full-fledged definition for leakage resilience allows the adversary to query the leakage oracle after the challenge. This means an adversary could design its leakage function via the information of challenge ciphertext. For instance, in the bounded leakage model, an adversary could encode the challenge and decryption algorithm together to recover the

whole message via leakage queries if its length is shorter than the bound. Most existing works, such as [1, 2, 8–10], beg this technical difficulty with a weaker security definition, which only admits prechallenge leakage queries. But in practice, after-the-fact leakage is really feasible because many cryptographic devices are portable so that the attack can be launched at any time.

Halevi and Lin [11] made an effort to treat after-the-fact leakage (AFL) directly. As classic semantic security is impossible under postleakage attack, they choose to put another limitation instead of ignoring it. They require that the adversary can only get access to different parts of the secret via leakage independently, not as a whole. This "split-state" leakage was also defined and applied in the setting "computation leaks only" [12]. This restriction is meaningful because it is feasible to store secret fractions in different locations. They introduced the notion of "entropy leakage" to capture after-the-fact leakage. This concept states that the leakage should not be used to obtain more information than itself. This is an essential property for a postchallenge leakage. They showed that constructions from the hash proof system like that in [2] meet the requirement of security against entropy leakage. And they gave the first after-the-fact leakage resilient (AFLR) encryption scheme secure against chosen plaintext attack (CPA) by combining two instances of entropy leakage resilient schemes. Then, Li et al. presented identity-based encryption secure against postchallenge leakage attack [13]. Yang and Li considered this problem for the key exchange protocol [14].

Since security against chosen ciphertext attack (CCA) is a well-accepted standard for encryption schemes, some subsequent works aimed to achieve this goal against AFL. Zhang et al. [15] followed the classic Naor–Yung paradigm [16] to give a construction with simulation sound noninteractive zero knowledge (NIZK) proof. Chakraborty et al. [17] presented a more efficient construction with true simulation extractable NIZK proof. Fujisaki et al. [18] considered the multichallenge setting as well as the leakage from randomness. There are indeed more techniques to obtain traditional CCA security, but few existing works secure against AFL attacks have been proposed.

Lossy trapdoor functions: besides double encryption paradigm [16] and hash proof system [19], there is another approach to achieve CCA security, via a powerful primitive called lossy trapdoor functions (LTFs). Since its appearance [20], this primitive has been widely applied in many areas. The CCA secure encryption schemes based on LTFs get rid of the burden from NIZK proofs so that it is more efficient than those which need NIZK proofs. Also, LTFs have brilliant properties to extract statistical entropy from computational indistinguishability between two working modes. So LTFs have its nature to play an important role in leakage resilient cryptography. Some prior works already tried this way. Qin et al. [8] designed an invariant called the lossy filter to replace the universal-2 part in HPS-based schemes and achieved better leakage rate. More directly, Qin et al. [21] attempted to construct LR-LTFs, but their result can only be proven secure in a weaker model in which the adversary can get access to entire public key after leakage queries. Chen et al. proposed an advanced version of lossy function with its application in leakage resilience [22].

1.1. Our Contribution. In this work, we demonstrate that AFLR-LTFs and ABO invariants can be constructed in the split-state model and then can achieve AFLR-CCA security without NIZK proofs either. First, we formulate the notion of AFL secure LTFs. Then, we realize this primitive from AFL CPA secure PKE schemes. To overcome the technical difficulty that most randomness extractors and the underlying PKE schemes do not have homomorphic property which is essential for this use, we refine a AFLR randomness extractor from the BHHO PKE scheme [21, 23] with this property. Thus, with an AFLR-LTF and an AFL-ABO-LTF, we can follow the approach in [20] to achieve CCA security. Furthermore, our construction is easy to be used to construct chameleon AFL-ABO-LTFs [24] for a more efficient CCA secure realization.

1.2. Organization. The remaining part of this paper is organized as follows: the basic definitions and tools we need is shown in Section 2. In Section 3, we build a step stone before arriving to the final step: a two-source extractor in the 2 split-state model. Then, we present AFLR-LTFs in Section 4 and an AFLR PKE scheme based on them in Section 5, respectively. The final scheme is interpreted in a black box manner from AFLR-LTFs. The security of the final scheme can be reduced to the security of AFLR-LTFs.

2. Preliminaries

2.1. ABO Lossy Trapdoor Functions. A collection of LTFs is a collection of publicly computable functions which are indexed by a set of public key $\{s\}$. Every public key is associated with a branch which is used to generate the key. There are two kinds of public keys. Functions indexed by one kind are injective, while functions indexed by the other have a smaller size of image than that of domain. We called the branch according to the former "injective branch" and the other "lossy branch." "Lossy" means the image of the function working on these branches loses part of the information of the preimage. We use a generalized notion to incorporate exponential lossy branches. Let $\{B_n\}$ denote a collection of branch sets and $\{B_n^*\}$ denote the corresponding collection of lossy branch sets. We recall the definition of ABO-LTFs [20] below. If $\{B_n\}$ contains two elements only, it is just the standard LTF.

Definition 1. A collection of (n, k) ABO-LTFs is composed of 3 probabilistic polynomial time (PPT) algorithms:

G_{abo}: take $\lambda \in \mathbb{N}$ and $b^* \in B_\lambda$ as input and output $(s, \text{td}, B_\lambda^*)$, where s is a function index, td is its trapdoor, and B_λ^* is the set of lossy branches that $b^* \in B_\lambda^*$.

F_{abo} and F_{abo}^{-1}: for any $b \in B_\lambda / B_\lambda^*$, $F_{\text{abo}}(s, b, \cdot)$ computes an injective function $f_{s,b}(\cdot)$ over the domain $\{0,1\}^n$ and $F_{\text{abo}}^{-1}(s, b, \cdot)$ computes $f_{s,b}^{-1}(\cdot)$. For any $b \in B_\lambda^*$, $F_{\text{abo}}(s, b, \cdot)$

computes a function $f_{s,b}(\cdot)$ over the domain $\{0,1\}^n$ whose image size is at most 2^k.

There are two security requirements for ABO-LTFs. Index indistinguishability: the ensemble $s \longleftarrow G_{\text{abo}}(\lambda, b_0^*)$ and $s \longleftarrow G_{\text{abo}}(\lambda, b_1^*)$ are computationally indistinguishable. Lossy branch hidden: any PPT adversary $\mathcal{A}$ which takes (s, b) as input, where $(s, \text{td}, B^*) \longleftarrow G_{\text{abo}}(\lambda, b^*)$ has only a negligible probability to find a b' such that $b' \neq b$ and $b' \in B^*$. And even $b \in B^*$, and the adversary could not find one either.

2.2. Randomness Extractor

2.2.1. One Source

Definition 2. A randomized algorithm $\text{Ext1}: \mathcal{K} \longrightarrow \{0,1\}^\nu$ is a (μ, ε) extractor if for all (X, Z) that is distributed on $\mathcal{K}$ and $\tilde{H}_\infty(X \mid Z) \geq \mu$, $(Z, S, \text{Ext1}(X; S)) =_s (Z, S, U_\nu)$, where U_ν is a uniform distribution over $\{0,1\}^\nu$ and S is called seed which is the coin of Ext1.

The parameters of the concrete extractor used need to satisfy the condition that $\nu \leq \mu - 2\log(1/\varepsilon) - 1$. Generally, pair-wise independent hash functions are used to realize extractors.

2.2.2. Two Source

Definition 3. A two-source extractor does not rely on random seeds but extracts randomness from two independent sources. A randomized algorithm $\text{Ext2}: (\mathcal{K})^2 \longrightarrow \{0,1\}^\nu$ is a (μ, ε) extractor if for all (K_1, K_2, Z) where K_1 and K_2 are distributed on $\mathcal{K}$ and have minimum entropy μ conditioned on Z, $(Z, \text{Ext}(K_1, K_2)) =_s (Z, U_\nu)$.

2.3. AFLR-CPA Secure PKE

2.3.1. Entropy Leakage Resilient PKE. The definition of entropy leakage resilience stresses that the leakage after challenge cannot be amplified. This fact is captured by a simulator, which interacts with the adversary in an indistinguishable manner to the real setting. Formally, we first set some parameters: k is the minimum entropy that the message source M has, l_{post} denotes the leakage after challenge, and δ is an overhead parameter which comes from the statistical distance that the extractor deviates from uniform distribution.

Definition 4. A PKE scheme Π is entropy leakage resilient if there exists a simulator Sim such that, for every PPT adversary $\mathcal{A}$, the following two conditions hold:

(1) $(M^{\text{real}}, \text{View}_{\mathcal{A}}^{\Pi}) =_c (M^{\text{sim}}, \text{View}_{\mathcal{A}}^{\text{Sim}})$

(2) $\tilde{H}_\infty(M^{\text{sim}} \mid \text{View}_{\mathcal{A}}^{\text{Sim}}) \geq k - l_{\text{post}} - \delta$

2.3.2. After-the-Fact Leakage Resilience. Semantic security against AFL is defined by a game between a challenger and an adversary just the same as normal CPA game, except that the adversary is allowed to issue leakage query before and after challenge. The semantic security requires that the adversary can still not win with nonnegligible advantage in this setting. The CCA security is define analogously.

An AFLR-CPA-secured PKE scheme in the 2 split-state model can be constructed via combination of two instances of an entropy leakage resilient PKE scheme and a two-source extractor. Specifically, given two entropy leakage resilient PKE schemes $\Pi_1 = (\text{Gen}_1, \text{Enc}_1, \text{Dec}_1)$ and $\Pi_2 = (\text{Gen}_2, \text{Enc}_2, \text{Dec}_2)$, a semantic secure scheme against a posthallenge leakage can be defined as $\text{Gen} = (\text{Gen}_1, \text{Gen}_2)$, $\text{Enc}(m; r_1, r_2) = (c_1 = \text{Enc}_1(r_1), c_2 = \text{Enc}_2(r_2), \text{Ext2}(r_1, r_2) \oplus m) = c$, $\text{Dec}(c) = c \oplus \text{Ext2}(\text{Dec}_1(c_1), \text{Dec}_2(c_2))$. The security proof for this construction is in [11].

2.4. Homomorphism.

A function is called homomorphism if the operation between elements in the domain preserves its structured functionality between elements in the range. For instance, let "+" denote the operation in the domain, "·" denote the operation in the range, and f: $A \longrightarrow B$ be the function. The property can be represented as $f(x + y) = f(x) \cdot f(y)$, which can induce $f(ax) = f(x)^a$.

2.5. DDH Assumption.

Given a cyclic group G with order q which is a big prime number, f and h are random elements in G and then (f, h, f^r, h^r) and $(f, h, f^r, h^{r'})$ are computationally indistinguishable for randomly chosen r and r'. Following a hybrid argument, this result can be extended to vector situation: $(g, f_1^r, \ldots, f_l^r, h_1^r, \ldots, h_l^r)$ and $(g, f_1^r, \ldots, f_l^r, h_1^{r'}, \ldots, h_l^{r'})$ are computationally indistinguishable for randomly chosen r and r'.

2.6. 2 Split-State Model.

This model is introduced in [11] to incorporate postchallenge leakage resilience. This model puts one more restriction than the ordinary security model against leakage attack that an adversary cannot issue leakage queries on the whole secret state but two separate parts. This means, instead of a leakage function f on sk, the adversary can only issue queries f_1 on sk_1 and f_2 on sk_2.

2.7. Notations.

Throughout this paper, we build our concrete construction on quadratic residue subgroup of the cyclic group with order N^2. So we present all the parameter settings here. Let $\mathbb{G}$ denote a group of order N^2 where N is a Blum integer, $\mathbb{G}_r$ the subgroup of $\mathbb{G}^*$ with order $(p-1)(q-1)/4$, n the security parameter, λ the length of leakage, and set $l = 2 + (\lambda + 2(\log 1/\varepsilon)/\log N - 3)$ for some negligible ε.

Note that DDH assumption also holds in $\mathbb{G}_r$.

Also, we define the multiple computation and exponential computation of a vector as $xy = (x_1, \ldots, x_n)(y_1, \ldots, y_n) = (x_1 y_1, \ldots, x_n y_n)$ and $x^r = (x_1, \ldots, x_n)^r = (x_1^r, \ldots, x_n^r)$.

3. Homomorphic and Leakage Resilient Randomness Extraction

In general, the keyed randomness extractor in leakage resilient setting is initiated with universal hash functions which do not incorporate homomorphic property. However, it is quite vital in our scheme. So we refine an extractor from a variant of BHHO scheme [21] which meets our requirement and leads to a construction of homomorphic two-source extractor.

3.1. One-Source Leakage Resilient Extractor. An extractor can be constructed as follows with abovementioned parameters:

Gen: choose $f_1, f_2, \ldots, f_l \in G_r$. The evaluation key is set to be $\mathrm{pk} = (N, f_1, f_2, \ldots, f_l)$.

Ext: for any $x = (x_1, x_2, \ldots, x_l)$ sampled from source $\mathbb{Z}_{N-1/4}$, choose $r \in \mathbb{Z}_{N-1/4}$, compute public random seed $u_1 = f_1^r, u_2 = f_2^r, \ldots, u_l = f_l^r$, and $f = \prod_{i=1}^{l} f_i^{x_i}$. Then, the extracted randomness is $\mathrm{Ext}(x; r) = \prod_{i=1}^{l} u_i^{x_i}$.

Following [21], R is distributed negligibly close to uniform even subject to λ bits leakage and published f.

Homomorphic property: we observe that $\mathrm{Ext}(x; r)^a = \mathrm{Ext}(x; ar)$. So this extractor has homomorphic random seed.

3.2. Two Sources in 2 Split-State Model. Given the same parameters as above, we can present our publicly computable two-source extractor in 2 split-state model.

Gen: choose $f_1, f_2, \ldots, f_l, h_1, h_2, \ldots, h_l \in G_r$. The evaluation key is set to be $\mathrm{pk} = (N, f_1, f_2, \ldots, f_l, h_1, h_2, \ldots, h_l)$.

Ext2: for any $x = (x_1, x_2, \ldots, x_l)$ and $y = (y_1, y_2, \ldots, y_l)$ sampled from source $\mathbb{Z}_{N-1/4}$, choose $r \in \mathbb{Z}_{N-1/4}$, compute public random seed $u_1 = f_1^r, u_2 = f_2^r, \ldots, u_l = f_l^r$, $v_1 = h_1^r, v_2 = h_2^r, \ldots, v_l = h_l^r$, and $f = \prod_{i=1}^{l} f_i^{x_i}$ and $h = \prod_{i=1}^{l} h_i^{y_i}$. Then, the extracted randomness is $\mathrm{Ext2}(x, y) = \prod_{i=1}^{l} u_i^{x_i} v_i^{y_i}$.

Theorem 1. *The construction above is a* $(\log N - 3 - \lambda, \varepsilon)$ *two-source extractor against* λ *bits leakage under DDH assumption.*

Proof. We prove this theorem via hybrid argument through games between a challenger and an adversary as follows:

Game0: the game proceeds as the real game. The challenger chooses $\mathrm{pk} = (N, f_1, f_2, \ldots, f_l, h_1, h_2, \ldots, h_l)$ and responds queries from the adversary as the algorithm.

Game1: in this game, the only change is that challenger computes the public random seed with two randomness r and r', which is $u_1 = f_1^r, u_2 = f_2^r, \ldots, u_l = f_l^r$, $v_1 = h_1^{r'}, v_2 = h_2^{r'}, \ldots, v_l = h_l^{r'}$ and $f = \prod_{i=1}^{l} f_i^{x_i}$ and $h = \prod_{i=1}^{l} h_i^{y_i}$. □

Lemma 1. *The view of adversary is indistinguishable between Game0 and Game1 assuming DDH problem is hard.*

Given a DDH instance $(g_1, g_2, A = g_1^r, z)$, the challenger can simulate the game by letting $(f_1 = g_1^{x_1}, f_2 = g_1^{x_2}, \ldots, f_l = g_1^{x_l})$ and $(h_1 = g_2^{y_1}, h_2 = g_2^{y_2}, \ldots, h_l = g_2^{y_l})$ where $(x_1, \ldots, x_l, y_1, \ldots y_l)$ are chosen randomly. In the challenge query, the challenger computes the public randomness as $u_1 = A^{x_1}, u_2 = A^{x_2}, \ldots, u_l = A^{x_l}$, $v_1 = z^{y_1}, v_2 = z^{y_2}, \ldots, v_l = z^{y_l}$. The challenger can answer leakage queries because it chooses secret key itself.

If the adversary can tell which game he is playing with nonnegligible advantage, then we can conclude that $z = g_2^r$, which breaks the DDH assumption.

Lemma 2. *The output is distributed negligibly close to uniform.*

In Game1, the output can be seen as the multiplication of two-independent leakage resilient one-source extractors in the 2 split-state model. For $l = 2 + (\lambda + 2(\log(1/\varepsilon))/\log N - 3)$ where ε is negligible, the output is the multiplication of two variables which are both distributed ε close to uniform. Thus, it is at least distributed ε close to uniform itself.

Combing lemma 1 and lemma 2, the construction above is a two-source extractor against λ bits leakage under DDH assumption in the 2 split-state model.

4. AFLR-LTFs in 2 Split-State Model

In this section, we formulate the notion of AFLR-LTFs in the 2 split-state model and give concrete constructions of its own and ABO variants.

4.1. Definition. In this model, the secret is divided into 2 parts for storage and leakage attack can only get access to each part independently but not a function of whole state as before. This restriction provides the possibility to achieve AFL resilience.

Definition 5. A collection of 2 split-state ABO-LTFs are composed of specified algorithms as follows:

G_{abo}: the generated trapdoor td is divided into two parts $(\mathrm{td}_1, \mathrm{td}_2)$, as well as the index $s = (s_1, s_2)$. The lossy branch set B_λ^* is the same as before.

F_{abo}^{-1}: the inversion algorithm consists of two subroutines inv_1 and inv_2 which take two parts of the secret as input, respectively. And a combing subroutine f_{-1} takes as input the output of the two subroutines and outputs the preimage.

The security notion requires that index indistinguishability and lossy branch hidden hold even subject to leakage attack. Note that this requirement is just the same as AFLR PKE because the adversary could issue leakage queries to check the lossy branch after it sees the index.

4.2. A Homomorphic AFLR PKE Scheme in 2 Split-State Model. Homomorphism is essential to the underlying PKE schemes for LTFs and CCA security [25]. However, the generic construction in [11] does not incorporate this property. But [11] indicated that variants of hash proof system-based schemes are entropy leakage resilient. So we use the extractors mentioned in Section 3. We start from a basic scheme in [21] which is a variant of the BHHO scheme (and thus hash proof system-based scheme) and then construct the scheme we need via this one.

The basic scheme is as follows:

Gen: choose $x_1, x_2, \ldots, x_l \in \mathbb{Z}_{N-1/4}$, $f_1, f_2, \ldots, f_l \in G_r$. Let $f = \prod_{i=1}^{l} f_i^{x_i}$. The public key is set to be $\mathrm{pk} = (N, f_1, f_2, \ldots, f_l, f)$, and the secret key is $\mathrm{sk} = (x_1, x_2, \ldots, x_l)$.

Enc: given the message $m \in \mathbb{Z}_{N-(1/4)}$, Choose $r \in \mathbb{Z}_{N-(1/4)}$ and compute $(c_1 = (u_1 = f_1^r, u_2 = f_2^r, \ldots, u_l = f_l^r), c_2 = f^r (1+N)^m)$.

Dec: given $c = (c_1, c_2)$. Compute $K = \prod_{i=1}^{l} u_i^{x_i}$ and $m = \log(c_2/K)$.

The construction is as follows:

Given two instances of the basic entropy leakage scheme $(\mathrm{Gen}_1, \mathrm{Enc}_1, \mathrm{Dec}_1)$ and $(\mathrm{Gen}_2, \mathrm{Enc}_2, \mathrm{Dec}_2)$, we define an AFLR PKE scheme in the 2 split-state model as follows:

GEN: it includes two subroutines of $(\mathrm{Gen}_1, \mathrm{Gen}_2)$. The outputs are a public key pair $\mathrm{pk} = (\mathrm{pk}_1, \mathrm{pk}_2)$ and a secret key pair $\mathrm{sk} = (\mathrm{sk}_1, \mathrm{sk}_2)$.

ENC: given a message m, it chooses randomness x_1, x_2, r as the input of two subroutines $(\mathrm{Enc}_1, \mathrm{Enc}_2)$ (we use the same randomness r in both encryption algorithm), and the ciphertext is computed as $C = (C_1, C_2, C_3) = ((c_1^1, c_2^1), (c_1^2, c_2^2), \mathrm{Ext2}(x_1, x_2)(1+N)^m)$.

DEC: $m = \log(C_3/\mathrm{Ext2}(\mathrm{Dec}_1(C_1), \mathrm{Dec}_2(C_2)))$.

Note: discrete logarithm in this case can be easily computed.

Homomorphic property: $\mathrm{ENC}^a(m; r) = \mathrm{ENC}(am; ar)$.

Theorem 2. *The construction above is a CPA secure scheme against* $(1 - O(1))|sk|$ *bits prechallenge leakage and* λ *bits postchallenge leakage under DDH assumption in the 2 split-state model.*

Proof. Let $\mathcal{A}$ denote the adversary. We prove the theorem via a sequence of hybrid experiments as follows.

Game0: the challenger and the adversary proceed as the normal game. The challenger chooses the secret to generate public key and respond leakage queries.

Game1: the only difference from Game0 happens in the challenge phase. The challenger chooses two independent randomness r_1, r_2 for $(\mathrm{Enc}_1, \mathrm{Enc}_2)$ instead of the same one for both.

Game2: in this game, the challenger generates half of public key by itself and runs the simulator of one-entropy leakage resilient instance to get the rest. In detail, the challenger execute Gen to generate $x_1, x_2, \ldots, x_l \in \mathbb{Z}_{N-(1/4)}$, $\mathrm{pk}_1 = (f = \prod_{i=1}^{l} f_i^{x_i}, f_1, f_2, \ldots, f_l)$ and runs a simulator of another instance of the basic scheme to receive $\mathrm{pk}_2 = h, h_1, h_2, \ldots, h_l \in G_r$. The public key is $(N, f_1, f_2, \ldots, f_l, h_1, h_2, \ldots, h_l, f, h)$. When the adversary issues a leakage query $(\mathrm{leak}_1, \mathrm{leak}_2)$, the challenger forward leak_2 to the simulator and receives the answer. The answer can be merged with the output of leak_1 which can be calculated by itself. In the challenge phase, the challenger chooses x_2 and sends it to simulator to get ciphertext (c_1^2, c_2^2). Then, it computes the challenge ciphertext by $C = (C_1, C_2, C_3) = ((c_1^1, c_2^1), (c_1^2, c_2^2), \mathrm{Ext2}(x_1, x_2)(1+N)^m)$. When the adversary issues a postchallenge leakage query, the challenger handles like the way in the prechallenge phase.

Game3: the challenger interacts with $\mathcal{A}$ via two entropy leakage resilient simulators. In this game, all the leakage queries are forwarded to simulators. The challenger computes $\mathrm{Ext2}(x_1, x_2)(1+N)^m$ itself but receives the rest part of ciphertext from simulators. □

Lemma 3. *The views of* $\mathcal{A}$ *in Game0 and Game1 are indistinguishable under DDH assumption.*

This lemma is the same as Theorem 1.

Lemma 4. *The views of* $\mathcal{A}$ *in Game1 and Game2 are indistinguishable following Defintion 4.*

Lemma 5. *The views of* $\mathcal{A}$ *in Game2 and Game3 are indistinguishable following Defintion 4.*

The above two lemmas hold assuming the property of simulator.

Lemma 6. *In Game3, the challenge ciphertext has distribution negligible close to uniform distribution against* $(1 - O(1))|sk|$ *bits prechallenge leakage and* λ *bits postchallenge leakage.*

This can be concluded by the property of the two-source extractor.

4.3. AFLR-LTFS. Following [25], AFLR-ABO-LTFS can be constructed as follows given a homomorphic AFLR-CPA secure encryption scheme $(\mathrm{GEN}, \mathrm{ENC}, \mathrm{DEC})$ which we present above.

PP: choose a branch b as the lossy branch and then run GEN and $\mathrm{ENC}(b) = C$. The public key is (pk, C), and the secret key is sk (we do not put b here because the security is not guaranteed with leaked b, and it can actually be obtained by decrypting C).

Evaluation f: for any input x, choose an evaluation branch b', $f(x) = (C/\mathrm{ENC}(b'))^x = \mathrm{ENC}((b - b')x)$. Output $(f(x), b')$.

Inversion f^{-1}: decrypt C to get b and then compute $f^{-1} = \mathrm{DEC}(f(x))/(b - b')$.

Security analysis: our construction achieves pre- and postchallenge leakage resilience more than [25]. Due to

the use of AFLR encryption scheme as the building block, we can handle leakage query before and after challenge, which makes the proof similar to the one in [25]. So we omit the details here.

Indistinguishability: adversary cannot tell the computation is lossy or not with nonnegligible advantage because the branch is encrypted with the AFLR encryption scheme. If the branch set consists only two elements 0 and 1, this construction can lead to a standard AFLR-LTF which will be used to achieve CCA security later. If the branch set contains many branches, the lossy one is also hidden from adversary.

Lossiness: the output has entropy at most $\log N - 2$. So the lossiness is at least $\log N - (\log N - 2) = 2$. These results can be extended if we use N^a as a module for the basic encryption scheme.

5. Constructions of AFLR-CCA Secure PKE

AFLR-CCA security can be obtained in a classic way with a standard AFLR-LTF, an AFLR-ABO-LTF, and an unforgeable one-time signature scheme. But we prefer another approach via chameleon AFLR-ABO-LTFs. Chameleon ABO-LTFs are introduced in [24] which can avoid using one-time signature. In this variant of LTFs, the lossy set is denoted as a line rather than points to incorporate exponential lossy branches. So we give the construction of chameleon AFLR-ABO-LTFs first.

5.1. Chameleon AFLR-ABO-LTFs

PP: choose $d, e, j \in \mathbb{Z}_{N-1/4}$ and then run GEN and $\mathrm{ENC}(d) = D$, $\mathrm{ENC}(e) = E$, and $\mathrm{ENC}(j) = J$. The public key is (pk, D, E, J), and the secret key is $\mathrm{sk}_{\mathrm{CH}}$.

Evaluation f_{CH}: for any input x, choose an evaluation branch (x_d, x_e), $f_{\mathrm{CH}}(x) = (D^{x_d} E^{x_e} J)^x = \mathrm{ENC}((dx_d + ex_e + j)x)$. Output $(f_{\mathrm{CH}}(x), (x_d, x_e))$.

Inversion f_{CH}^{-1}: decrypt D, E, J to get d, e, j and then compute $f_{\mathrm{CH}}^{-1} = \mathrm{DEC}(f_{\mathrm{CH}}(x))/(dx_d + ex_e + j)$.

The lossy branches are all pairs (x_d, x_e) that satisfy the condition $dx_d + ex_e + j = 0$.

5.2. AFLR-CCA Secure PKE Scheme. We can build our AFLR-CCA secure encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ by combining standard AFLR-LTFs and chameleon AFLR-ABO-LTFs as [24].

$\mathcal{G}$: first generate public parameters for LTF and chameleon ABO-LTF with an AFLR-CPA secure PKE scheme $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$. Let H be a universal hash function and h a collision-resistant hash function. The public key is $(\mathrm{pk}, \mathrm{pk}_{\mathrm{CH}}, H, h, A = \mathrm{Enc}(1), D = \mathrm{Enc}(d), E = \mathrm{Enc}(e), J = \mathrm{Enc}(j))$ where d, e, j are randomly chosen and independently encrypted. Thus, the standard LTF $f(x) = A^x$ and chameleon AFLR-ABO-LTF $f_{\mathrm{CH}}(x; (x_d, x_e)) = (D^{x_d} E^{x_e} J)^x$ are well defined. The secret key is $(\mathrm{sk}, \mathrm{sk}_{\mathrm{CH}})$.

$\mathcal{E}$: for a message m, choose a randomness x, evaluation branch b randomly, and compute $C = (c_0 = H(x) \oplus m, c_1 = f(x), c_2 = f_{\mathrm{CH}}(x; b, h(c_0, c_1)), b)$.

$\mathcal{D}$: given a ciphertext $C = (c_0, c_1, c_2, b)$, compute $x = f^{-1}(c_1)$ and check whether $c_2 = f_{\mathrm{CH}}(x; b, h(c_0, c_1))$. If the output is not $\perp$, then output $m = c_0 \oplus H(x)$.

Theorem 3. *Given AFLR-LTFs and chameleon AFLR-ABO-LTFs, the construction above is an AFLR-CCA secure PKE scheme in the 2 split-state model against λ bits after-the-fact leakage.*

Proof sketch: the case without leakage attack are proven secure in [24]. The proof goes with a sequence of indistinguishable games between challenger and adversary. The first step is to reject all the decryption queries with lossy computation by chameleon ABO-LTF. Then, change the working mode of LTF to be lossy and the decryption queries can be responded by chameleon LTF on injective branches. Finally, CCA security is achieved statistically with appropriate parameter.

As the underlying AFLR primitives we propose in Sections 4 and 5.1 can handle leakage queries in both pre- and postchallenge phase, we can preserve AFLR security if we use these primitives instead of ones in ordinary case naturally. Readers can check every step and see the proof strategy above can still work with additional leakage attack. □

6. Efficiency in Practice

The generic constructions in previous works [15, 17] need NIZK system to prove the language that two encryptions contain the same plaintext. In practice, NIZK proofs secure in standard model concerns the Groth–Sahai system [26] which suffers from heavy burden of computations via bilinear mappings. Specifically, proving a commitment of exponential which is only a step stone for proving equal plaintext requires 4 group elements and verified by 9 pairing operations. The cost of NIZK for same plaintext may be dozens of group elements and pairing operations. That is why existing works did not even give concrete construction for NIZK-based solutions. This situation is just like "two-key" generic construction in [16] which is convincing but not practical until [19] appeared. Our construction comes from a leakage resilient extension of [19] and achieves CCA security against postleakage without NIZK just like Cramer and Shoup [19] did in classic environment.

Specifically, the evaluation key in our scheme can be processed in precomputation and the encryption algorithm works by $8l$ exponential computations. If we want to achieve 80 bit security ($\varepsilon = 2^{-80}$) with 1024 bit N, $l = 2 + ((\lambda + 160)/7)$ against λ bit leakage. If we want to encrypt longer plaintext, we can use larger modulus like $N^a, a > 2$.

We implement our scheme to evaluate its efficiency, which is based on JPBC 2.0.0 library (http://gas.dia.unisa.it/projects/jpbc/index.html#.VTDrLSOl_Cw) and coding

language Java. We select type A1 pairings are constructed on the curve $y^2 = x^3 + x$ over the field $\mathbb{Z}_N$ for some Blum integer N. The following experiments are based on Dell laptop (Windows 7 operation system with Intel(R) Core(TM) i5-2450M CPU 2.50 GHz, 4.00 GB RAM, and 500 G disk storage). The time cost in real-world experiment for one encryption is 0.042 s with 1024 bit N.

7. Conclusion and Future Direction

Our work removes the use of zero knowledge proofs which is not efficient in the construction of AFLR-CCA secure PKE encryption schemes via the approach of lossy trapdoor functions. We also present instances of AFLR-LTF and its variants. An interesting open problem is finding more efficient PKE schemes with both homomorphic property and leakage resilience.

Data Availability

The simulation data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Key R&D Program of China (2017YFB0802000), the National Natural Science Foundation of China (61572303 and 61772326), the Natural Science Basic Research Plan in Shaanxi Province of China (2018JQ6088), the National Cryptography Development Fund during the 13th Five-year Plan Period (MMJJ20170216), the Foundation of State Key Laboratory of Information Security (2017-MS-03), the Fundamental Research Funds for the Central Universities (GK201702004 and GK201803064), and the Project of Basic Research of Qinghai Province (2016-ZJ-776).

References

[1] A. Akavia, S. Goldwasser, and V. Vaikuntanathan, "Simultaneous hardcore bits and cryptography against memory attacks," *Theory of Cryptography*, pp. 474–495, 2009.

[2] M. Noar and G. Segev, "Public-key cryptosystems resilient to key leakage," *SIAM Journal on Computing*, vol. 41, no. 4, pp. 772–814, 2012.

[3] E. Boyle, G. Segev, and D. Wichs, "Fully leakage-resilient signatures," *Journal of Cryptology*, vol. 26, no. 3, pp. 513–558, 2013.

[4] A. Faonio, J. B. Nielsen, and D. Venturi, "Fully leakage-resilient signatures revisited: graceful degradation, noisy leakage, and construction in the bounded-retrieval model," *Theoretical Computer Science*, vol. 660, pp. 23–56, 2017.

[5] J.-D. Wu, Y.-M. Tseng, and S.-S. Huang, "Leakage-resilient ID-based signature scheme in the generic bilinear group model," *Security and Communication Networks*, vol. 9, no. 17, pp. 3987–4001, 2016.

[6] J. Alwen, Y. Dodis, and D. Wichs, "Leakage-resilient public-key cryptography in the bounded-retrieval model," *Advances in Cryptology—CRYPTO 2009*, vol. 5677, pp. 36–54, 2009.

[7] J. Alawatugoda, "On the leakage-resilient key exchange," *Journal of Mathematical Cryptology*, vol. 11, no. 4, pp. 215–269, 2017.

[8] B. Qin, K. Chen, and S. Liu, "Efficient chosen-ciphertext secure public-key encryption scheme with high leakage-resilience," *IET Information Security*, vol. 9, no. 1, pp. 32–42, 2015.

[9] Y. Zhou, B. Yang, W. Zhang, and Y. Mu, "CCA2 secure public-key encryption scheme tolerating continual leakage attacks," *Security and Communication Networks*, vol. 9, no. 17, pp. 4505–4519, 2016.

[10] S.-F. Sun, D. Gu, and S. Liu, "Efficient chosen ciphertext secure identity-based encryption against key leakage attacks," *Security and Communication Networks*, vol. 9, no. 11, pp. 1417–1434, 2016.

[11] S. Halevi and H. Lin, "After-the-fact leakage in public-key encryption," in *Theory of Cryptography*, pp. 107–124, Springer, Berlin, Germany, 2011.

[12] S. Micali and L. Reyzin, "Physically observable cryptography," in *Theory of Cryptography*, pp. 278–296, Springer, Berlin, Germany, 2004.

[13] J. Li, Y. Guo, Q. Yu, Y. Lu, and Y. Zhang, "Provably secure identity-based encryption resilient to post-challenge continuous auxiliary input leakage," *Security and Communication Networks*, vol. 9, no. 10, pp. 1016–1024, 2016.

[14] Z. Yang and S. Li, "On security analysis of an after-the-fact leakage resilient key exchange protocol," *Information Processing Letters*, vol. 116, no. 1, pp. 33–40, 2016.

[15] Z. Zhang, S. S. M. Chow, and Z. Cao, "Post-challenge leakage in public-key encryption," *Theoretical Computer Science*, vol. 572, pp. 25–49, 2015.

[16] M. Noar and M. Yung, "Public-key cryptosystems provably secure against chosen ciphertext attacks," in *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing—STOC'90*, pp. 427–437, Baltimore, MD, USA, May 1990.

[17] S. Chakraborty, G. Paul, and C. P. Rangan, "Efficient compilers for after-the-fact leakage: from CPA to CCA-2 secure PKE to AKE," in *Information Security and Privacy*, pp. 343–362, Springer, Berlin, Germany, 2017.

[18] E. Fujisaki, A. Kawachi, R. Nishimaki, K. Tanaka, and K. Yasunaga, "Post-challenge leakage resilient public-key cryptosystem in split state model," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98.A, no. 3, pp. 853–862, 2015.

[19] R. Cramer and V. Shoup, "Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption," in *Advances in Cryptology—EUROCRYPT 2002*, pp. 45–64, Springer, Berlin, Germany, 2002.

[20] C. Peikert and B. Waters, "Lossy trapdoor functions and their applications," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1803–1844, 2011.

[21] B. Qin, S. Liu, K. Chen, and M. Charlemagne, "Leakage-resilient lossy trapdoor functions and public-key encryption," in *Proceedings of the First ACM Workshop on Asia Public-Key Cryptography—AsiaPKC'13*, pp. 3–12, Hangzhou, China, May 2013.

[22] Y. Chen, B. Qin, and H. Xue, "Regular lossy functions and their applications in leakage-resilient cryptography," *Theoretical Computer Science*, vol. 739, pp. 13–38, 2018.

[23] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky, "Circular-secure encryption from decision diffie-hellman," in *Proceedings of the CRYPTO*, pp. 108–125, Santa Barbara, CA, USA, August 2008.

[24] S. Liu, J. Lai, and R. H. Deng, "General construction of chameleon all-but-one trapdoor functions," *Journal of Internet Services and Information Security*, vol. 1, no. 2-3, pp. 74–88, 2011.

[25] B. Hemenway and R. Ostrovsky, "Homomorphic encryption over cyclic groups implies chosen-ciphertext security," *IACR Cryptology*, vol. 99, 2010.

[26] J. Groth, "Simulation-sound NIZK proofs for a practical language and constant size group signatures," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 444–459, Springer, Shanghai, China, December 2006.

Hindawi
Security and Communication Networks
Volume 2019, Article ID 7541269, 16 pages
https://doi.org/10.1155/2019/7541269

Research Article

Secure Information Sharing System for Online Patient Networks

Hyun-A Park

Department of Medical Health Sciences, Kyungdong University, Republic of Korea

Correspondence should be addressed to Hyun-A Park; kokokzi@naver.com

Received 9 August 2018; Revised 31 January 2019; Accepted 14 February 2019; Published 12 March 2019

Guest Editor: Chunhua Su

Recently, privacy emerged as a hot issue again, as the General Data Protection Regulation (GDPR) of EU has become enforceable since May 25, 2018. This paper deals with the problem of health information sharing on a website securely and with preserving privacy. In the context of patient networks (such as 'PatientsLikeMe' or 'CureTogether'), we propose the model Secure Information Sharing System (SISS) with the main method of group key cryptosystem. SISS addresses important problems of group key systems. (1) The new developed equations for encryption and decryption can eliminate the rekeying and redistribution process for every membership-change of the group, keeping the security requirements. (2) The new 3D Stereoscopic Image Mobile Security Technology with AR (augmented reality) solves the problem of conspiracy by group members. (3) SISS uses the reversed one-way hash chain to guarantee forward secrecy and backward accessibility (security requirements for information sharing in a group). We conduct a security analysis of SISS according to group information sharing secrecy and an experiment on its performance. Consequently, although current IT paradigm is changing to be more and more 'complicated', 'overlapped', and 'virtualized', SISS makes it possible to securely share sensitive information from collaborative work.

1. Introduction

Patients around the world want to connect to people who are suffering from the same symptoms and try to find the best treatments. These days, there are some online patient websites for health information sharing such as "PatientsLikeMe"[1] or "CureTogether"[2], where patients talk about their symptoms and successful (or failed) experiences. Researchers can also discover new and better solutions based on the patient-contributed data.

1.1. Problem Identification. The problem is that these kinds of health data may be sensitive and private information. Therefore, patients want the sensitive health data to be protected and managed with safety, and the data to be revealed to only limited persons. That is, the individuals' right to privacy and control over the circulation of their information [3]. In particular, privacy emerged as a hot issue again, as the General Data Protection Regulation (GDPR) of EU has become enforceable since May 25, 2018.

One of the substantial ways to share patients' information and to protect privacy is a group key management system. However, a group key system has some peculiar characteristics; the group key should be updated whenever members leave or join the group. This is called rekeying and redistribution. These processes need a complicated and high level of security. Another feature that differentiates an information sharing system from other general group key management systems is that leaving members cannot access the group's information anymore, but joining members can access all the previous group's information to get more information for better treatment. Moreover, current members may conspire with leaving members or other people by revealing their group key [4].

In this paper, the proposed model SISS (Secure Information Sharing System) addresses the above problems in the context of online patient networks such as PatientsLikeMe or CureTogether. The main methods are group key management system, including the newly developed encryption/decryption algorithm and reversed hash key chain, and 3D Stereoscopic Image Mobile Security Technology

(augmented reality technique). Namely, that is the secure version for the websites which can make people with the same symptoms share their health information safely and securely.

1.2. Main Goals. The SISS's principle to manage users' information with safety is that general information should be presented in plaintexts but sensitive information should be managed in ciphertexts. The solution for sharing ciphertexts with users is group key management system. The following is the main goals of SISS's group key system.

There Should Be No Processes of Rekeying and Redistribution. Whenever membership-changes (users leaving or joining groups) happen in the general group key system, the system should generate a new group key and distribute the new key to all members very quickly. This process needs more complicated and secure level of techniques, so 'rekeying and redistribution' falls under the hard problem in group key systems. To solve this problem, we develop new equations of the encryption and decryption with the group key. Therefore, SISS has no process of rekeying and redistribution for membership-changes. To the best of my knowledge, this is the first trial to eliminate the process of rekeying and redistribution keeping the security of the group key.

SISS Guarantees Forward Secrecy and Backward Accessibility. The representative security requirements for general group key systems are forward secrecy and backward secrecy. However, the security requirements of SISS to share information with group members are forward secrecy and backward accessibility, not backward secrecy. The reversed hash chain can guarantee the security properties of SISS.

SISS is Collusion-Resistant with New Technology. In every group member system, one of the important security problems is the potential for conspiracy between users and illegal members. As a solution, SISS proposes a new concept of 3D Stereoscopic Image Mobile Security Technology using VR/AR technique.

1.3. Methods and Contributions. The main methods and contributions are listed in detail as follows:

(1) The group key system for encryption and decryption: Only valid users can share the secret information with their group keys and pseudonyms. Others (invalid users) cannot know the contents and ownership for the information: 'what about' and 'whose information'.

(2) SISS addresses the security problem with reversed hash chain and 3D Stereoscopic Image Mobile Security Technology. And, the newly developed encryption and decryption algorithms are used for efficiency. It is because the information sharing has been and will be highly increased in the networked collaborative computing environments.

(2.1) *Equations for Encryption and Decryption*: SISS has no need for rekeying and redistribution for membership-changes. The principle of group key generation for each member is that a fixed master group key is assigned to each group, and random numbers for each user and every session are newly generated by applying random numbers to hash function respectfully, reversedly, and repeatedly s times. Thereafter, each random number and the master group key are combined according to the developed equation algorithm. Then, a total of five subgroup keys are generated as each member's group session key for every session. Hence, every member has completely different group keys for each session and each member because of generating different random number each time. However, one of the most important things is that the results after calculation of all other group keys for the developed equation (for encryption and decryption) are the same as the result of master group key for encryption and decryption, which makes it possible for SISS not to have rekeying and redistribution processes whenever there are membership-changes.

(2.2) *3D Stereoscopic Image Mobile Security Technology*: It is a new concept of a security solution using VR/AR techniques against conspiracy. The combination of human's facial expressions and gestures which are identified at the registration time of a legitimate group member (LGM) should be rendered as a 3D image in the login process to authenticate a legitimate user. Consequently, the rendering of users' own facial expressions and gestures can prohibit conspiracy between users and invalid persons.

(2.3) *Reversed Hash Chain*: SISS should guarantee forward secrecy and backward accessibility, along with group key secrecy [5], which are security requirements in group information sharing system. In SISS, every member's group key is generated based on reversed one-way hash chain. Due to one-way properties of reversed hash function[6], a leaving member cannot know the next group key (forward secrecy) but a joining member can know all the previous keys and information (backward accessibility). Therefore, SISS is suitable for secret collaborative work and sensitive information sharing among group members.

(2.4) *All Different Random Numbers for Each Member and Each Session*: In SISS system, every member's group session key looks like private key because their group key has completely different values with different random numbers for each member. Nevertheless, the key plays a role of a group key, with which every group member can share their sensitive health information with other members in a group.

(3) Stronger authentication for login: The proposed model uses LGM (legitimate group member) for stronger authentication. Moreover, the authentication processes are mutual, so that the proposed model is secure against spoofing or masquerading attack.

(4) Scalability to other group project systems and mobile phone applicability: SISS is scalable to other group project systems on websites. Although application scenario is about patient networks on the web, SISS is extendable to other secure group projects. Furthermore, it is applicable to even LGM of mobile phones, because the next smartphone will feature a front-facing 3D laser scanner for facial recognition, which was the expectation of upcoming iPhone 8 in early 2018 [7] (but Apple released it without 3D laser scanner in late 2017).

(5) Privacy preserving system: SISS can meet the privacy requirements: pseudonymity (partial anonymity), unlinkability, and unobservability.

(6) Blinding: In every flow, SISS uses newly generated random numbers. This masking method does not allow an attacker to know or to guess real contents correctly.

2. Related Works and Application

2.1. Related Works. In this section, we introduce our main solution's research area of group key.

Many researchers have worked on group key in various views such as group key agreement, exchange, revocation, and multicast/broadcast. However, this paper only focuses on the group key application for sharing information among multiple users. In particular, SISS has a little different property from the general group key in that the security requirement of SISS is not backward secrecy (a joining member cannot know all the previous keys and information) but backward accessibility (a joining member can know all the previous keys and information). It is caused by the different application's goal from other group key systems, which can enable current group members to share all their information securely. That is the reason why SISS is related to the research area of keyword search schemes for multiuser setting.

At first, we address the researches about 'multiuser setting'. In [8], Park et. al firstly proposed privacy preserving keyword-based retrieval protocols for dynamic groups (multiuser setting) based on reversed hash key chain. In multiuser setting environments like companies and municipal offices, a server contains heterogeneous documents accessible by different groups or persons. A leaving member from a group should not have access to any documents of the group anymore, that is, forward secrecy. Because a newly joining member should perform the tasks of the group to which he belongs, all documents accessible to the group must be still available and he should be able to obtain all the group keys, that is, backward accessibility. They accomplished both security properties with reversed hash key chain. Thereafter, they made the formal definition for 'backward accessibility' firstly in [5], where they proposed two practical group search schemes in the respect of efficiency in cloud datacenter. As for the researches not based on reversed hash key chain, Wang et. al. [9] analyzed Park et. al's scheme[8] on various views including security and efficiency. They achieved backward accessibility for multiple users with public key and Pallier's cryptosystem instead of reversed hash key chain. The next year, Wang et. al proposed another scheme of keyword field-free conjunctive keyword searches on encrypted data in the dynamic group setting [10]. In [11], Li et al. suggested a new effective fuzzy keyword search in a multiuser system over encrypted cloud data. This system supports differential searching and privileges based on the techniques attribute-based encryption and Edit Distance.

In respect of key-updating and redistribution, there have been also many researches until now, because group key's rekeying and redistribution to all group members are complicated and hard tasks. Generally, this research area is divided into four categories: a centralized distribution scheme, a distributed key agreement scheme, a hybrid group key management scheme, and a self-healing group key distribution scheme (SGKD). The significant point of centralized key distribution schemes [12–20] is that a centralized key management center as a trusted party generates, updates, and distributes the group keys to all members. As to distributed key agreement schemes [21–28], all group members participate in generation, rekeying, and redistribution of their group keys. Hybrid group key management schemes [29–32] are about the best use of both schemes: a centralized distribution scheme and a distributed key agreement scheme. Lastly, a self-healing group key distribution scheme (SGKD) is for wireless networks including sensor networks [33–50]. In this paper, we focus on SGKD schemes because SISS's application includes both networks of wired and wireless and SISS has more relations with SGKD schemes than other schemes.

The main point of self-healing schemes is that group members can recover the missing session keys without retransmission of the missing messages from the GM (group manager). A SGKD scheme is largely divided into four categories again: polynomial based SGKD (P-SGKD) schemes [33–37], vector space secret sharing based SGKD schemes [38–41], bilinear pairings based SGKD schemes [42, 43], and exponential arithmetic based SGKD (E-SGKD) schemes [44–51]. P-SGKD and vector space secret sharing based SGKD schemes are generally known as "not efficient". A polynomial secret sharing scheme is the most common technique, where two types of polynomials are constructed: the revocation polynomial and the access polynomial. In [44], Rams et al. pointed out that almost all of the polynomial based SGKD schemes can be converted to E-SGKD schemes. P-SGKD schemes can be divided into two types based on using Lagrange Interpolation or not. In [33, 45–47] E-SGKD schemes are constructed from P-SGKD schemes with Lagrange Interpolation. The other P-SGKD schemes without Lagrange Interpolation can be classified into two types again based on using hash chains or not. Scheme 3 in [48] and Scheme 2 in [49] are E-SGKD schemes transformed from revocation polynomial based P-SGKD schemes without hash chains. In [50], Gou et. al first proposed E-SGKD scheme with high efficiency and backward secrecy by combining dual chains: a traditional hash chain and a key chain.

2.2. Entities and Application Scenario. SISS has three parties: users, SM (security manager), and SISS server. SM (security manager) is a kind of a client, which is granted a special role of a security manager. SM is assumed as a TTP (trusted third party) and it is located in front of the SISS server. SM controls group key and key-related information, all sensitive information, and all other events with powerful computational and storage abilities. Figure 1 shows the system configuration of SISS.

The main participants of SISS are group members who want to get help through information sharing. The information scope is health conditions and patient profile. Mostly, the health information could be shared but some secret personal data in patient profile should be revealed to the allowed people only.

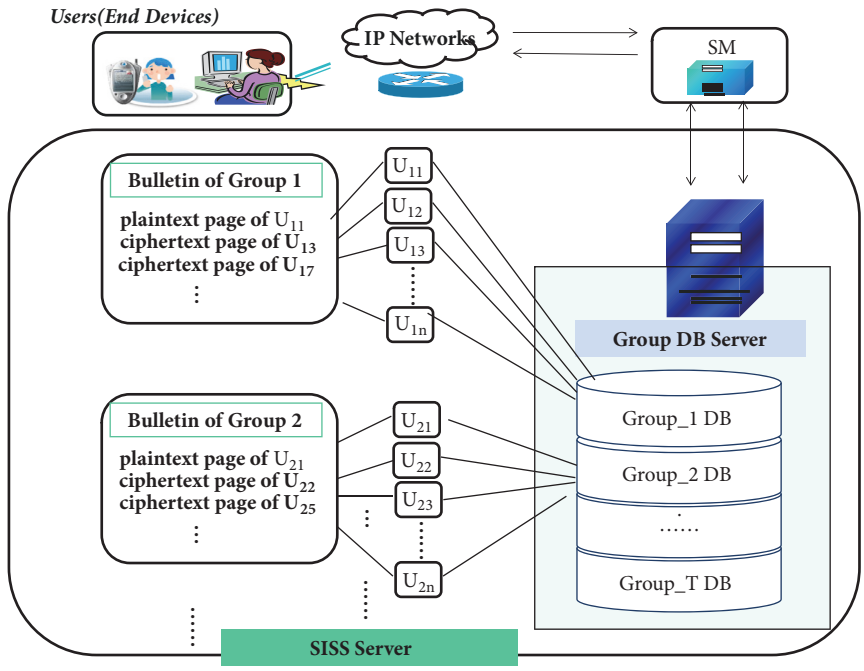


Figure 1: The configuration of SISS.

With the mobile devices or PC, Secure Information Sharing System (SISS) is constructed for online website patients with any disease all over the world. The needs are as follows: Many patients want to get in such kind of patient networks and to be helped more easily and securely. Each member uploads his/her conditions or information to his/her personal ciphertext pages (for sensitive data) or plaintext pages (for public data).

One more important thing is that the augmented reality (AR) technique of 3D image rendering is applied to external devices. The rendered image is selected from the contents-list consisting of the randomly repeated and rearranged human's facial expressions and gestures. As the first process, every user should register at SM; thereafter they should get through the authentication processes every session and then they start some actions. When some sensitive information is shared with other patients (it means that the ciphertext pages are generated), we know the page is encrypted by the group's encryption key. Only the legitimate users (who registered at SM and kept the information given by SM at their devices for authentication) can pass authentication processes and know the sharing information. In the last step of the authentication, 3-dimensional image is rendered. This image can be called a legitimate group member.

3. Preliminaries

3.1. Notations. The notations for SISS are explained in Table 1.

3.2. Algorithms for SISS Model. A SISS model consists of the following eight algorithms.

(1) $SysPrm(1^k)$: Parameter Generation algorithm $SysPrm$ takes as an input a security parameter k and produces a system parameter λ.

(2) $KeyGen(\lambda)$: Taking λ as input, Key Generation algorithm $KeyGen$ produces group session random numbers set RH, group member keys set K, and member pseudonym keys set P.

(3) $InfGenStr(\lambda, RH, K, P)$: Taking λ, RH, K, P as input, Information Generation and Storage algorithm $InfGenStr$ produces LGM (legitimate group member) and other information for authentication.

(4) $Q_r(RH, K, P)$: Given RH, K, P, Query algorithm produces Query Value Q_v.

(5) V_r-$Q_r(Q_v)$. Given Q_v, Verification algorithm verifies Q_v, and Query algorithm produces Query Value Q_v.

(6) $V_r(Q_v)$: Given Q_v, Verification algorithm verifies Q_v.

(7) $EncUp(M)$: This algorithm encrypts and uploads message M.

(8) $DnDec(M)$: This algorithm downloads and decrypts message M.

3.3. Security Building Blocks and Model

Definition 1 (one-way hash chain). It is generated by selecting the last value at random and applying it to one-way hash function h repeatedly. The initially chosen value is the last value of the key chain. Following are two properties of one-way hash chain.

Property 2. Anybody can deduce the earlier value k_i belonging to the one-way key chain with the later value k_j by checking $h^{j-i}(k_j)$ which equals k_i with the later value k_j.

Property 3. Given the latest released value k_i of one-way key chain, an adversary cannot find a later value k_j such that $h^{j-i}(k_j)$ equals k_i. Even when value k_{i+1} is released, the second preimage collision resistant property prevents an adversary from finding k'_{i+1} different from k_{i+1} such that $h(k_{i+1})$ equals k_i [6].

Remark. We call property 1 of one-way hash key chain 'backward accessibility' and property 2 'forward security'.

Definition 4 (PRF (pseudorandom function)). We say that '$F : K_f \times X \longrightarrow Y$ is (t, q, e)-secure pseudorandom function' if every oracle algorithm A making at most q oracle queries and with running time at most t has advantage $Adv_A < e$. The advantage is defined as $Adv_A = |Pr[A^{F_k} = 1] - Pr[A^{g} = 1]|$ where g represents a random function selected uniformly from the set of all maps from X to Y, and where the probabilities are taken over the choice of k and g [51].

Definition 5 (PRG G_r (pseudorandom generator)). We say that '$Gr : K_{Gr} \longrightarrow S$ is a (t, e)-secure pseudorandom generator' if every algorithm A with running time at most t has advantage $Adv_A < e$. The advantage is defined as $Adv_A = |Pr[A(Gr(U_{K_{Gr}})) = 1] - Pr[A(U_S) = 1]|$, where $U_{K_{Gr}}, U_S$ are random variables distributed uniformly on K_{Gr}, S [51].

Definition 6 (DDH (decisional Diffie-Hellman)). Let G be a group of prime order q and g a generator of G. The DDH problem is to distinguish between triplets of the form (g^a, g^b, g^{ab}) and (g^a, g^b, g^c), where a, b, c are random elements of $\{1, \ldots, q-1\}$.

Definition 7 (collusion resistance). The leaving member $U_i^l \in G^l$ $(1 \le i \le n, 1 \le l < l' \le s)$ colluding with the members in the after sessions $U_{i'}^j \in G^j$ $(1 \le i' \le n, j \ge l', i \ne i')$ cannot recover $K_{Gi}^{\ j}$ even knowing $\{\alpha_i^l, K_{Gi}^{\ l}\}$ and $\{\alpha_{i'}^j, K_{Gi'}^{\ j}\}$ [50], where G^l is a group G at l-th session.

Definition 8 (security game ICR-IS). (Indistinguishability of Ciphertexts from Random Bit Strings in Information Sharing)

Setup. The challenger C creates a ciphertext set B of m pages $\in \{CP_{it}\} \in \{B_t\}$ $(1 \le i \le n, 1 \le t \le T)$ and gives this to the adversary A. A chooses a polynomial number of subsets from B. This collection of subsets is called B^*; C runs algorithm $SysPrm, KeyGen, InfGenStr$ and encrypts each subset running algorithm $Q_r, V_rQ_r, V_r, EncUp$. Finally, C sends A all ciphertexts with their associated subsets.

Queries. A may request the encryption $EncUp(B^*)$) of any B and any verification V_r.

Challenge. A chooses a B_0 and its subsets such that none of the algorithms (Q_r, V_rQ_r, V_r) given in the step Queries distinguishes B_0 from $B_1 = \text{rand}(B_0)$. The challenger C chooses a random bit b and gives $EncUp(B_b)$ to A. A again asks for encrypted pages and their verifications with the restriction that A may not ask for the algorithm that distinguishes B_0 from B_1. The total number of ciphertexts and verifications is in k.

Response. A outputs $b_A \in \{0, 1\}$. If $b_A = b$, A is successful. In security game ICR-IS, adversary's advantage is defined as $Adv(1^k) = |Pr[b_A = b] - 1/2|$.

Table 1: Notations.

SM	Security Manager
K_G	the group keys set of group G
G_t	the t-th group
T	the total number of groups in a SISS server
mK_g	the master group key of group G
$CP_{i,t}^j$	the j-th ciphertext-page of a member i in a group t
n	the total number of group G's members
sn	session number
m	the total number of ciphertext-page CP
$U_{t,i}^j$	a member i of the t-th group in the j-th session
$K_{Gt,i}^{\ j}$	a group session key for each member i of t-th group G_t in the j-th session $sK_{i,1}^j, sK_{i,2}^j, sK_{i,3}^j, sK_{i,4}^j, sK_{i,5}^j$: five subkeys for a group session key $K_{Gi}^{\ j}$ of a member i in group G
α_i^j	random number of member i in the j-th session
p_i^j	pseudonym of member i in the j-th session
$h(\cdot)$	hash function
$h^t(x)$	the value hashed t-times for x
$f(\cdot)$	pseudorandom function
S_i^j	stereoscopic image information for 3D real model of member i in the j-th session
R_{S_i}	a rendered image of S_i
E	Encryption function
D	Decryption function
M	a message

3.4. Legitimate Group Member (LGM) . Every user (member) registers at SM with the contents-list which is the combination set for gestures and facial expressions of the user. All the gestures and facial expressions are randomly repeated and rearranged in the contents-list. Then, the user keeps the contents-list in their device for later authentication. $S_{t,i}^j$ is put as the stereoscopic image information for the gesture and facial expression of the member i of group t at the j-th session. Every session, SM selects one of the combinations of gestures and facial expressions from the contents-list and challenges the member of the group. Then, the member renders his own gesture and facial expression for $S_{t,i}^j$.

4. Construction of SISS Model

In this section, SISS is constructed by using the eight algorithms described before. This SISS model is divided largely

into four processes: system setting, registration, authentication for login, and action. The whole process is shown in Table 2 and the details are addressed in Section 4.1.

4.1. System Setting

4.1.1. SysPrm(1^k) Construction. The basis of security system SISS is established.

(i) *Input*; k: a security parameter.

(ii) *Output*;$\lambda = f(\cdot), h(\cdot), G, g, G_r, n, j, i, E, D$: system parameters' set.

$f : \{0,1\}^k \times \{0,1\}^* \longrightarrow \{0,1\}^k$is a pseudorandom function and $h : \{0,1\}^* \longrightarrow \{0,1\}^k$ is a one-way hash function. G_r is a pseudorandom generator. G is a group of order q which is a large prime. g is a generator of a group G, n is the total members of group G, j is the session number, and i is each member of group G. E and D are encryption and decryption function.

4.2. Registration.

The registration process consists of two algorithms: $KeyGen(\lambda)$, $InfGenStr(\lambda, RH, K, P)$.

4.2.1. KeyGen(λ) Construction. Key materials are generated.

(i) *Input*; λ.

(ii) *Output*;RH(*group session random numbers set*), K(*group members keys set*), P(*pseudonym keys set*).

(1) Group Session Random Numbers RH: Reversed One-Way Hash Chain. It is assumed that the total number of sessions is s. For every member i, each different random number α_i^s $(1 \le i \le n)$ is generated for the last session. Here, each α_i^s is applied to one-way hash function $(s-1)$ times repeatedly to generate all sessions' random numbers and, respectively, for each user as follows.

$$\begin{aligned}
&\alpha_i^s, \ \text{(randomly generated)}\\
&h\left(\alpha_i^s\right) = \alpha_i^{s-1}\\
&h\left(\alpha_i^{s-1}\right) = \alpha_i^{s-2} = h^2\left(\alpha_i^s\right)\\
&h\left(\alpha_i^{s-2}\right) = \alpha_i^{s-3} = h^3\left(\alpha_i^s\right)\\
&\cdots\\
&h\left(\alpha_i^4\right) = \alpha_i^3 = h^{s-3}\left(\alpha_i^s\right)\\
&h\left(\alpha_i^3\right) = \alpha_i^2 = h^{s-2}\left(\alpha_i^s\right)\\
&h\left(\alpha_i^2\right) = \alpha_i^1 = h^{s-1}\left(\alpha_i^s\right)
\end{aligned} \tag{1}$$

Therefore, the first session's random number of member i is α_i^1 and the *t-th* session's random number of member i is α_i^t : $h(\alpha_i^{t+1}) = \alpha_i^t = h^{s-t}(\alpha_i^s)$.

With these different random numbers, we can make all different group keys for each member and each session, respectively.

One-way hash function $h(\cdot)$ plays the important role in group information sharing. One-Way hash chain is generated by randomly selecting the last value, which is repeatedly applied to one-way hash function $h(\cdot)$. The initially selected value is the last value of the hash chain. One-way hash chain has two properties as mentioned in Definition 1 in Section 3. Therefore, the two properties make it possible that a leaving member cannot compute new keys after leaving the group and any newly joining member can obtain all previous keys and information through applying the current key to hash function $h(\cdot)$ repeatedly.

(2) Group Keys Set K. It is assumed that there are 'n' members of the group 'G', and the group session keys for each member i of group G are $\{K_{Gi}^j\}$ $(1 \le i \le n, 1 \le j \le s)$. Here, j is a session number and s is the last session. The each member i's group key K_{Gi}^j consists of totally five subkeys; $sK_{i,1}^j, sK_{i,2}^j, sK_{i,3}^j, sK_{i,4}^j, sK_{i,5}^j$. SM selects the master group key mK_g of group $G \in \{G_t\}$, $t \ge 1$ and generates a random number to blind the master key in five subkeys, which is the way to construct a group member's session key and, therefore, the last session group key K_{Gi}^s of user i.

$$\begin{aligned}
sK_{i,1}^s &= h\left(mK_g\right)\alpha_i^s,\\
sK_{i,2}^s &= h\left(mK_g\right) f_{mK_g}\left(mK_g\right)\left(1-\alpha_i^s\right),\\
sK_{i,3}^s &= g^{f_{mK_g}(mK_g)},\\
sK_{i,4}^s &= -\left(h\left(mK_g\right)+\alpha_i^s\right),\\
sK_{i,5}^s &= f_{mK_g}\left(mK_g\right)\alpha_i^s
\end{aligned} \tag{2}$$

The generation principle of group keys is that every different random number for each member and each session ($n \times s$ random numbers) is combined to the master group key mK_g. Table 3 shows the random numbers and group keys for each member and each session which belong to the group G. The group G is one of the groups $\{G_t\}, t \ge 1$.

(3) Group Members' Pseudonym Keys Set P: Reversed One-Way Hash Chain. For stronger security and privacy, SISS uses each member's pseudonyms, which are generated with the reversed one-way hash chain in the same way as group session keys. Thus, each member has also s pseudonyms which are denoted as p_i^j (for each member i, $1 \le j \le s$).

$$\begin{aligned}
&p_i^s, \ \text{(randomly generated)}\\
&h\left(p_i^s\right) = p_i^{s-1}\\
&h\left(p_i^{s-1}\right) = p_i^{s-2} = h^2\left(p_i^s\right)\\
&h\left(p_i^{s-2}\right) = p_i^{s-3} = h^3\left(p_i^s\right)\\
&\cdots
\end{aligned}$$

$$h\left(p_i^4\right) = p_i^3 = h^{s-3}\left(p_i^s\right)$$
$$h\left(p_i^3\right) = p_i^2 = h^{s-2}\left(p_i^s\right)$$
$$h\left(p_i^2\right) = p_i^1 = h^{s-1}\left(p_i^s\right) \tag{3}$$

4.2.2. $InfGenStr(\lambda)$ Construction. The values to be used for authentication are generated and saved.

(i) *Input*; λ: system parameters' set.

(ii) *Output*; $S_{t,i}^j$, $h(E_{K_{Gi}^1}(p_i^1 \parallel S))$, $\{h(E_{K_{Gi}^j}(p_i^j))\}$, p_i^1, K_{Gi}^1, α_i^q, $\{h(p_i^j), p_i^j\}$ $(1 \le j \le s)$.

At the registration process, every user is given some information from SM and stores them in one's own device such as in smartphone or PC: $S_{t,i}^j$, p_i^1, K_{Gi}^1, $\{h(E_{K_{Gi}^j}(p_i^j))\}$, $h(E_{K_{Gi}^1}(p_i^1 \parallel S))$, $(1 \le i \le n, 1 \le j \le s)$. SM also stores some information for each member i: α_i^s, $\{h(p_i^j), p_i^j\}$, mK_g, $S_{t,i}^j$, $(1 \le j \le s)$.

As such, the output of $InfGenStr(\lambda)$ is the values created by the SM and each user during the registration process, which means the values are stored in advance for later use in the authentication process.

Additional Explanation for Encryption (E) and Decryption (D) with Group Keys. Here, $E_{K_{Gi}^j}(M)$ means ciphertext C with group members' group key '$K_G = \{K_{Gi}^j\}$' for a message M. The encryption with the master key mK_g is assumed $C = E_{mK_g}(M) = g^{h(mK_g)f(mK_g)}M$.

For simplicity, we put $sK_{i,1}^s, sK_{i,2}^s, sK_{i,3}^s, sK_{i,4}^s, sK_{i,5}^s$ as K_1, K_2, K_3, K_4, K_5, and $f_{mK_g}(mK_g)$ as $f(mK_g)$. Then, the encryption with each member's group key $\{K_{Gi}^j\}$, for example, in the last session (i.e., $j = s$, K_{Gi}^s) is as follows:

$$\begin{aligned} C &= E_{K_{Gi}^s}(M) = K_3^{K_1} g^{K_2} M \\ &= \left(g^{f(mK_g)}\right)^{h(mK_g)\alpha_i^s} g^{h(mK_g)f(mK_g)(1-\alpha_i^s)} M \\ &= g^{h(mK_g)f(mK_g)} M. \end{aligned} \tag{4}$$

The decryption method with the group master key 'mK_g' is $D = C \cdot g^{-h(mK_g)f(mK_g)} = M$. Then, the decryption method with each member's group key K_{Gi}^s in the last session is as follows:

$$\begin{aligned} D &= C \cdot K_3^{K_4} g^{K_5} \\ &= g^{h(mK_g)f(mK_g)} M \cdot \left(g^{f(mK_g)}\right)^{-(h(mK_g)+\alpha_i^s)} \cdot g^{f(mK_g)\alpha_i^s} \\ &= g^{h(mK_g)f(mK_g) - f(mK_g)h(mK_g) - f(mK_g)\alpha_i^s + f(mK_g)\alpha_i^s} \cdot M \\ &= M. \end{aligned} \tag{5}$$

We can check whether the result of encryption/decryption with the master group key 'mK_g' is the same as anything of each member's group key $K_{Gi}^j = \{sK_{i,1}^j, sK_{i,2}^j, sK_{i,3}^j, sK_{i,4}^j, sK_{i,5}^j\}$ $(1 \le i \le n, 1 \le j \le s)$. Because of the properties of this developed encryption and decryption algorithms, SISS has no need for rekeying processes whenever membership-changes happen.

4.3. Login by Authentication. The login process consists of four algorithms: $QrU1(RH, K, P)$, $VrSM1\_QrSM2(QvU1)$, $VrU2_QrU3(Qv - SM2)$, and $VrSM3(R_{S_i^j})$.

4.3.1. $QrU1(RH, K, P)$ Construction. A member i makes login-request to SM with the stored information.

(i) *Input*; RH, K, P.

(ii) *Output*; $QvU1$ (1st querying value of a user).

(1) *Compute*: $f_{p_i^1}(K_{Gi}^1), h(p_i^1)$; with the stored value p_i^1, K_{Gi}^1, a member i computes $f_{p_i^1}(K_{Gi}^1), h(p_i^1)$.

(2) *A member i queries SM with QvU1*:

$$\begin{aligned} &QvU1 \\ &\quad = 1(sn), h\left(p_i^1\right), f_{p_i^1}\left(K_{Gi}^1\right), h\left(E_{K_{Gi}^1}\left(p_i^1 \parallel S_i^1\right)\right). \end{aligned} \tag{6}$$

Here, $h(E_{K_{Gi}^1}(p_i^1 \parallel S_i^1))$ is also the stored value at registration time. Because K_{Gi}^1 is the member i's group key in the first session, $E_{K_{Gi}^1}(p_i^1 \parallel S_i^1)$ means

$$\begin{aligned} C &= E_{K_{Gi}^1}\left(p_i^1 \parallel S_i^1\right) = K_3^{1K_1^1} g^{K_2^1}\left(p_i^1 \parallel S_i^1\right) \\ &= \left(g^{f(mK_g)}\right)^{h(mK_g)\alpha_i^1} g^{h(mK_g)f(mK_g)(1-\alpha_i^1)}\left(p_i^1 \parallel S_i^1\right) \\ &= g^{h(mK_g)f(mK_g)}\left(p_i^1 \parallel S_i^1\right). \end{aligned} \tag{7}$$

Here, for simplicity, K_1^1, K_2^1, K_3^1 are denoted as the member i's subkeys for its group key K_{Gi}^1 in the first session. K_4^1, K_5^1 are also the subkeys (for decryption) of K_{Gi}^1.

4.3.2. $VrSM1\_QrSM2(QvU1)$ Construction. The SM verifies the login-request of member i and sends the next session information, the group key and the pseudorandom number, to member i.

(i) *Input*; $QvU1$.

(ii) *Output*; $QvSM2$ (2nd querying value of SM).

(1) *Find*: α_i^s, p_i^1.

SM checks $1(sn), h(p_i^1)$ and finds the corresponding values α_i^s, p_i^1 from its storage.

(2) *SM decrypts* with p_i^1 : $D(f_{p_i^1}(K_{Gi}^1)) = K_{Gi}^1$.

(3) *Compute and Verify*: $h^{s-1}(\alpha_i^s) = \alpha_i^{1\prime}$, $K_{Gi}^{1\prime} = K_{Gi}^1$.

Table 2: The whole process of SISS.

User	*SM*
1. System Setting	
1.1. SysPrm	
2. Registration	
2.1. KeyGen	
2.2. InfGenStr	
3. Log-in by Authentication	
3.1. QrU1 (1) Compute and Query: $f_{p_i^1}(K_{Gi}^{\ 1}), h(p_i^1)$ $\xrightarrow{\{1(sn), h(p_i^1), f_{p_i^1}(K_{Gi}^{\ 1}), h(E_{K_{Gi}^1}(p_i^1 \| S_i^1))\}}$	
	3.2. VrSM1_QrSM2 (1) Find: $1(sn), h(p_i^1) \longrightarrow \alpha_i^s, p_i^1$ (2) Decrypt: $D(f_{p_i^1}(K_{Gi}^{\ 1})) = K_{Gi}^{\ 1}$ (3) Compute: $h^{s-1}(\alpha_i^s) = \alpha_i^{1\prime}$, $K_{Gi}^{\ 1\prime} = \{K_1^1, K_2^1, K_3^1, K_4^1, K_5^1\}'$ Verify: $K_{Gi}^{\ 1\prime} = K_{Gi}^{\ 1}$ (4) Compute & Verify: $h(E_{K_{Gi}^1}(p_i^1 \| S_i^1))' =? h(E_{K_{Gi}^1}(p_i^1 \| S_i^1))$ (5) Compute: $\alpha_i^2, K_{Gi}^{\ 2}$ (6) Compute & Query: $\xleftarrow{f_{p_i^1}(K_{Gi}^2, p_i^2), f_{p_i^2}(p_i^1 \| S_i^1)}$
3.3. VrU2_QrU3 (1) Decrypt: $D(f_{p_i^1}(K_{Gi}^{\ 2}, p_i^2)) = K_{Gi}^{\ 2\prime}, p_i^{2\prime}$ (2) Compute & Verify: $h(E_{K_{Gi}^2}(p_i^2))' =? h(E_{K_{Gi}^2}(p_i^2))$, $h(p_i^{2\prime}) = p_i^1$ Then, $K_{Gi}^{\ 2\prime} \longrightarrow K_{Gi}^{\ 2}, p_i^{2\prime} \longrightarrow p_i^2$ (3) Decrypt: $D(f_{p_i^2}(p_i^1 \| S_i^1)) = p_i^1 \| S_i^1$ (4) Render at a page: $R'_{S_i^1}$	
	3.4. VrSM3 (1) Verify: $R'_{S_i^1} = R_{S_i^1}$
4. Action	
4.1. EncUp [*member* − *i*] (1) Encrypt & Upload M: $C_i^1 = E_{K_{Gi}^1}(M)$	$\xrightarrow{=K_{i,3}^{1\ K_{i,1}^1} \cdot g^{K_{i,2}^1} \cdot M = g^{h(K_G) f(K_G)} M}$
4.2. DnDec [*member* − *u*] (1) Download from SISS Server: (2) Decrypt $C_i^1 : D = C_i^1 \cdot K_{u,3}^{1\ K_{u,4}^1} \cdot g^{K_{u,5}^1} = M$	$\xleftarrow{C_i^1}$

Table 3: Random number and group key.

		Users		
		U_1	U_2	U_3...
session 1	random number	α_1^1	α_2^1	...
	group key	K_{G1}^1;	K_{G2}^1;	...
		$sK_{1,1}^1 = h(mK_g)\alpha_1^1$	$sK_{2,1}^1 = h(mK_g)\alpha_2^1$	...
		$sK_{1,2}^1 = h(mK_g) f_{mK_g}(mK_g)(1-\alpha_1^1)$	$sK_{2,2}^1 = h(mK_g) f_{mK_g}(mK_g)(1-\alpha_2^1)$	...
		$sK_{1,3}^1 = g^{f_{mK_g}(mK_g)}$	$sK_{1,3}^1 = g^{f_{mK_g}(mK_g)}$	...
		$sK_{1,4}^1 = -(h(mK_g) + \alpha_1^1)$	$sK_{2,4}^1 = -(h(mK_g) + \alpha_2^1)$	...
		$sK_{1,5}^1 = f_{mK_g}(mK_g)\alpha_1^1$	$sK_{2,5}^1 = f_{mK_g}(mK_g)\alpha_2^1$	...
session 2	random number	α_1^2	α_2^2	...
	group key	K_{G1}^2;	K_{G2}^2;	...
		$sK_{1,1}^2 = h(mK_g)\alpha_1^2$	$sK_{2,1}^2 = h(mK_g)\alpha_2^2$	...
		$sK_{1,2}^2 = h(mK_g) f_{mK_g}(mK_g)(1-\alpha_1^2)$	$sK_{2,2}^2 = h(mK_g) f_{mK_g}(mK_g)(1-\alpha_2^2)$	...
		$sK_{1,3}^2 = g^{f_{mK_g}(mK_g)}$	$sK_{1,3}^2 = g^{f_{mK_g}(mK_g)}$	...
		$sK_{1,4}^2 = -(h(mK_g) + \alpha_1^2)$	$sK_{2,4}^2 = -(h(mK_g) + \alpha_2^2)$	...
		$sK_{1,5}^2 = f_{mK_g}(mK_g)\alpha_1^2$	$sK_{2,5}^2 = f_{mK_g}(mK_g)\alpha_2^2$	...
session 3...	...	...	...	...

For the found value α_i^s, SM applies α_i^s to hash function repeatedly, up to $(s-1)$ times. If he obtains the result $\alpha_i^{1\prime}$, then SM computes $K_{Gi}^{1\prime}$;

$$\begin{aligned} K_1^{1\prime} &= h\left(mK_g\right)\alpha_i^{1\prime}, \\ K_2^{1\prime} &= h\left(mK_g\right) f_{mK_g}\left(mK_g\right)\left(1-\alpha_i^{1\prime}\right), \\ K_3^{1\prime} &= g^{f_{mK_g}(mK_g)}, \\ K_4^{1\prime} &= -\left(h\left(mK_g\right) + \alpha_i^{1\prime}\right), \\ K_5^{1\prime} &= f_{mK_g}\left(mK_g\right)\alpha_i^{1\prime}. \end{aligned} \tag{8}$$

Then, SM verifies $K_{Gi}^{1\prime} = K_{Gi}^1$ or not.

(4) *Compute and Verify* with K_{Gi}^1: $h(E_{K_{Gi}^1}(p_i^1 \parallel S_i^1))' = h(E_{K_{Gi}^1}(p_i^1 \parallel S_i^1))$.

Here, $E_{K_{Gi}^1}(p_i^1 \parallel S_i^1)$ is the stored value at the registration time and the encryption method is the same as the above 1.

(5) *Compute*: α_i^2, K_{Gi}^2.

SM applies α_i^s to hash function $(s-2)$ times and then computes K_{Gi}^2;

$$\begin{aligned} K_1^2 &= h\left(mK_g\right)\alpha_i^2, \\ K_2^2 &= h\left(mK_g\right) f_{mK_g}\left(mK_g\right)\left(1-\alpha_i^2\right), \\ K_3^2 &= g^{f_{mK_g}(mK_g)}, \\ K_4^2 &= -\left(h\left(mK_g\right) + \alpha_i^2\right), \\ K_5^2 &= f_{mK_g}\left(mK_g\right)\alpha_i^2. \end{aligned} \tag{9}$$

(6) *Compute and Query with QvSM2*:

$QvSM2 = f_{p_i^1}(K_{Gi}^2, p_i^2), f_{p_i^2}(p_i^1 \parallel S_i^1)$, where p_i^2 is also the stored value.

4.3.3. VrU2_QrU3(QvSM2) Construction. A member i verifies the received information from SM, then stores it for the next session, and renders the stereoscopic image on his page.

(i) *Input*; *QvSM2*.

(ii) *Output*; $R_{S_i^j}$ (rendering of S_i^j).

(1) *Decrypt*: i decrypts $D(f_{p_i^1}(K_{Gi}^2, p_i^2)) = K_{Gi}^{2\prime}, p_i^{2\prime}$ with the value p_i^1.

(2) *Compute and Verify*: $h(E_{K_{Gi}^2}(p_i^2))' = h(E_{K_{Gi}^2}(p_i^2))$, $h(p_i^{2\prime}) = p_i^1$.

With the decrypted values $K_{Gi}^{2\prime}, p_i^{2\prime}$, the group member i computes $h(E_{K_{Gi}^2}(p_i^2))'$ and verifies if this is the same as $h(E_{K_{Gi}^2}(p_i^2))$. Then, i hashes the value $p_i^{2\prime}$ and verifies $h(p_i^{2\prime}) = p_i^1$. If the verifications are successful, $K_{Gi}^{2\prime}$ and $p_i^{2\prime}$ become K_{Gi}^2 and p_i^2.

(3) *Decrypt* with p_i^2: $D(f_{p_i^2}(p_i^1 \parallel S_i^1)) = p_i^1 \parallel S_i^1$.

(4) *Render and Upload:* $R'_{S_i^1}$ at a page.

4.3.4. VrSM3($R_{S_i^1}$) Construction. SM verifies what the member i has rendered.

(i) *Input*; $R_{S_i^j}$.

(ii) *Output*; 1 or 0.

(1) *Verify*: $R'_{S_i^1} = R_{S_i^1}$*(3D facial expression and gesture)*.

In this process, a legitimate group member authentication is processed by rendering the decrypted S_i^1 with the member's external device. If SM's verification is successful (return message: 1), the member i can begin to act (login allowed). The action means uploading, downloading, and reading (decryption).

4.4. Action. The Action Process consists of two algorithms: $EncUp(M)$, $DnDec(M)$.

4.4.1. EncUp(M) Construction. A member i encrypts and uploads the sharing information.

(i) *Input*; M.

(ii) *Output*; C_i^1.

(1) *Encrypt and Upload M* by a member *i*.

$$C_i^1 = E_{K_{Gi}^1}(M) = K_{i,3}^{1\,K_{i,1}^1} \cdot g^{K_{i,2}^1} \cdot M = g^{h(mK_g)f(mK_g)}M \quad (10)$$

4.4.2. DnDec(M) Construction. Another member u downloads and decrypts the sharing information.

(i) *Input*; C_i^1.

(ii) *Output*; M.

(1) *Download*: C_i^1.

Another member u downloads C_i^1 from SISS bulletin (server).

(2) *Decrypt* C_i^1: $D = C_i^1 \cdot K_{u,3}^{1\,K_{u,4}^1} \cdot g^{K_{u,5}^1} = g^{h(mK_g)f(mK_g)}M \cdot (g^{f(mK_g)})^{-(h(mK_g)+\alpha_u^1)} \cdot g^{f(mK_g)\alpha_u^1} = g^{h(mK_g)f(mK_g)-f(mK_g)h(mK_g)-f(mK_g)\alpha_u^1+f(mK_g)\alpha_u^1} \cdot M = M$.

[The Second Session]. From the second session, most processes are similar to the first session. As the session is changed, the corresponding pseudonym keys and group session keys are also changed. As for the stereoscopic image information S for 3D real model, a member sends the information S^t kept from the first session to SM, and then SM challenges the member with the newly selected information S^{t+1} at (6) of the algorithm $VrSM1\_QrSM2$. Lastly, the member renders 3D real model $R_{S^{t+1}}$ at his page. Action stage is also similar to the first session.

5. Security Analysis

The security requirements related to group key are as follows:

(1) Group Key Secrecy: It should be computationally impossible that a passive adversary discovers any secret group key.

(2) Forward Secrecy: Any passive adversary with a subset of old group keys cannot discover any subsequent (later) group key.

(3) Backward Secrecy: Any passive adversary with a subset of subsequent group keys cannot discover any preceding (earlier) group key.

(4) Key Independence: Any passive adversary with any subset of group keys cannot discover any other group key [4].

In this paper, the term negligible function refers to a function $\eta : N \longrightarrow R$ such that for any $c \in N$, there exists $n_c \in N$, such that $\eta(n) < 1/n_c$ for all $n \geq n_c$ [5].

The model SISS satisfies group information sharing secrecy as follows: (1) forward secrecy, (2) backward accessibility, (3) group key secrecy, and (4) collusion resistance.

Theorem 1 (forward secrecy). *For any group, an adversary A (including a participant $p \in G_t^j$) cannot know valid group key for (j+l)-th authentication when the adversary A knows group key K_{Gi}^j, where $p \notin G_t^{j+l}$ $(1 \leq j \leq s, 0 < l \leq s - j, 1 \leq t \leq T)$.*

Proof. By Property 3 of Definition 1, if the latest released group key is K_{Gi}^j, no one can know a later value K_{Gi}^l such that $h^{(l-j)}(K_{Gi}^l) = K_{Gi}^j$. Therefore, the probability that a participant $p \in G_t^j$ can generate valid group keys for the next *l-th* session is negligible, where $p \notin G_t^{j+l}$ $(j < l \leq s)$. It means that all leaving group members cannot access any of the next documents of the group anymore. □

Theorem 2 (backward accessibility). *For any group G_t, an adversary A (including a participant $p \in G_t^j$) can generate valid group key for (j-l)-th authentication when the adversary A knows group key K_{Gi}^j, where $p \notin G_t^{j-l}$ $(0 < l < j)$. Namely, all joining members to a group can access all of the previous information of the group.*

Proof. By Property 2 of Definition 1, if the latest released group key is K_{Gi}^j, anyone can deduce earlier values K_{Gi}^l $(0 < l < j)$ by applying the later value K_{Gi}^j to one-way hash key chain like this: $h^{(j-l)}(K_{Gi}^j) = K_{Gi}^l$. Therefore, the probability that a participant $p \in G_t^j$ can generate valid group keys for the earlier *l-th* session is $1 - \eta(n)$, where $p \notin G_t^{j-l}$ $(0 < l < j)$. Namely, all members joining a group can access all of the previous information of the group. □

Theorem 3 (group key secrecy). *For any group G_t, when a revelation of group key K_{Gi}^j happens, the probability that an adversary A (including a participant $p \in G_t^j$) can guess correctly the encrypted information message M of group G_t at the $j - th$ session is negligible.*

Corollary 4. *SISS is semantic secure according to the security game ICR -IS, if DDH is hard and the key material is chosen as described in the algorithm construction.*

The cryptographic elements for authentication and whole protocol are PRF (pseudorandom function, e.g., 128 bit-AES), PRG (pseudorandom generator, e.g., middle-square method, Naor-Reingold pseudorandom function, etc.), and hash function (HAS-160), generally known as secure cryptographic function. Through the cooperative processes of these elements, the final encryption is $C = E_{K_{Gi}^s}(M) = g^{h(mK_g)f(mK_g)}M$. Hence, we have only to show the security under the condition of 'DDH is hard'.

Proof (it is proved by contraposition) . A is assumed as an adversary that wins the security game ICR-IS with advantage ε. We construct an adversary Ω, which uses A as a subroutine and breaks the DDH with nonnegligible advantage.

(i) *Setup*. Algorithm Ω creates m message pages $\in \{M_{i,t}^j\} \in \{M_t\}$ $(1 \le i \le n, 1 \le t \le T, 1 \le j \le m)$ and gives this to the adversary A.

A chooses a polynomial number of subsets $\{M_{i,t}^j\} \in \{M_t\}$ from messages set M. This collection of subsets is called M^*. A sends them to Ω again. Ω invokes algorithm $SysPrm, KeyGen, InfGenStr$. After creating all ciphertext pages $\{CP_{i,t}^j\} \in \{CP_t\}$ for M^*, Ω gives them and their associated subsets to A.

Here, let $g^\delta, g^\tau, g^\gamma$ be a Diffie-Hellman triplet; the challenge is to determine $\gamma = \delta\tau$. Ω guesses a value CP^x for the page CP^y that A will choose in the game ICR-IS, by picking M^x uniformly at random in $\{M_{i,t}^j\}$ $(1 \le t \le T)$. Ω simulates the algorithm *EndUp* on $\{CP_{i,t}^j\}$ as follows. Ω maps every ciphertext page $\{CP_{i,t}^j\}$ to a random value $\{x_{i,t}^j\}$. For B= $\{B_t\}$=$\{CP_{i,t}^j\}$, Ω chooses random number γ_t and outputs the following.

$$\begin{aligned}
&\{B_t\}\left(1 \le i \le n, 1 \le t \le T, 1 \le j \le m\right) = \left\{CP_{i,1}^j\right\} \\
&= \left\{g^{h(k_1)f(k_1)}\left(M_{i,1}^j\right)\right\} = \left\{g^{\gamma_1}\left(x_{i,1}^j\right)\right\} \\
&= \left\{g^{\delta_1\tau_1}\left(x_{i,1}^j\right)\right\} = \left\{CP_{i,2}^j\right\} = \left\{g^{h(k_2)f(k_2)}\left(M_{i,2}^j\right)\right\} \\
&= \left\{g^{\gamma_2}\left(x_{i,2}^j\right)\right\} = \left\{g^{\delta_2\tau_2}\left(x_{i,2}^j\right)\right\} \\
&\cdots \\
&\cdots \\
&= \left\{CP_{i,T}^j\right\} = \left\{g^{h(k_T)f(k_T)}\left(M_{i,T}^j\right)\right\} = \left\{g^{\gamma_t}\left(x_{i,T}^j\right)\right\} \\
&= \left\{g^{\delta_T\tau_T}\left(x_{i,T}^j\right)\right\}
\end{aligned} \tag{11}$$

(ii) *Queries*. If A queries for the message page $\{M_{i,t}^j\}$, Ω outputs the ciphertext page $\{CP_{i,t}^j\}$= $\{(g^{\gamma_t}(x_{i,t}^j)\}$.

(iii) *Challenge*. Finally, A selects a challenge page set $B_0 \in M_t$ at random and generates another page set B_1 from M. Next, A gives B_0, B_1 to Ω. Ω chooses $b \underleftarrow{R} \{0, 1\}$ and chooses random number γ_b. Ω returns to A the following ciphertext: In the case of $b = 0$, $CP^y = g^{\gamma_0}(x_{i,0}^j)$. If $b = 1$, Ω returns random value in reply to DDH challenge. If $\gamma = \delta\tau$, this is an encryption of $\{CP^x\}$; otherwise it is not. A is again allowed to ask for pages of the Board set with the restriction that A must not make a query to distinguish $\{CP_{i,0}^j\}$ from $\{CP_{i,1}^j\}$ = rand$(CP_{i,0}^j)$ where $\{CP_{i,0}^j\}$ means a DDH triplet and $\{CP_{i,1}^j\}$ is not a DDH triplet.

(iv) *Response*. A outputs a bit b'. If $b' = b$, Ω guesses that $g^\delta, g^\tau, g^\gamma$ constitute a DDH triplet. If $b' \ne b$, Ω guesses that $g^\delta, g^\tau, g^\gamma$ do not constitute a DDH triplet. Since the encryption will be random for the page $\{CP^x\}$ if and only if the challenge is not a DDH tuple, Ω solves the DDH challenge with the same advantage that A has in winning security game ICR-IS.

It is shown that Ω can solve the DDH problem $(\gamma = \delta\tau)$ with nonnegligible probability. Accordingly, the advantage of Ω in winning this experiment is as follows.

$$\begin{aligned}
Adv_\Omega &= Pr\left[Exp_\Omega^{DDH} = 1\right] = Pr\left[b' = b\right] \\
&= Pr\left[b' = b \mid b = 1\right] \cdot Pr\left[b = 1\right] \\
&+ Pr\left[b' = b \mid b = 0\right] \cdot Pr\left[b = 0\right] \\
&= Pr\left[b' = b \mid b = 1\right] \cdot \frac{1}{2} + Pr\left[b' = b \mid b = 0\right] \cdot \frac{1}{2} \\
&= Pr\left[b' = 1 \mid b = 1\right] \cdot \frac{1}{2} + Pr\left[b' = 0 \mid b = 0\right] \cdot \frac{1}{2} \\
&= Pr\left[b' = 1 \mid b = 1\right] \cdot \frac{1}{2} + \Big(1 \\
&- Pr\left[b' = 1 \mid b = 0\right]\Big) \cdot \frac{1}{2} = \frac{1}{2} \\
&+ \frac{1}{2}\Big(Pr\left[Exp_A^{ICR-IS-1} = 1\right] \\
&- Pr\left[Exp_A^{ICR-IS-0} = 1\right]\Big) = \frac{1}{2} + \frac{1}{2}Adv_A^{ICR-IS} = \frac{1}{2} \\
&+ \frac{1}{2}\varepsilon
\end{aligned} \tag{12}$$

□

Theorem 5 (collusion resistance). *For any leaving member* $U_i^l \in G^l$ $(1 \le i \le n, 1 \le l < l' \le s)$ *including any other adversaries, SISS is ns-collusion-resistant.*

Proof. For anyone U_i^l $(1 \le l < l' \le s)$ colluding with the legitimate member $U_{i'}^j$, the illegal member cannot compute $K_{Gi}^{\ j}$ $(j \ge l')$. Although the compromised (illegal) member U_i^l knows $\{\alpha_i^l, K_{Gi}^{\ l}\}$ and $\{\alpha_{i'}^j, K_{Gi'}^{\ j}\}$, the illegal member cannot receive $\{\alpha_i^{l+1}, K_{Gi}^{\ l+1}\}$ of the next $(l + 1)$th session. Hence, they cannot compute $K_{Gi}^{\ j}$ $(1 \le i \le n, 1 \le l < l' \le j \le s)$. One more important thing is that the illegal member cannot pass the verifiable process to render the real 3D image from the stereoscopic information S_i^j which consists of the member's own gesture and facial expression. Therefore, the illegal member cannot pass the authentication process of login. □

6. Performance Analysis

The main purpose of this paper is to design a prototype scheme for secure patient networks. In addition, we try to apply a new technology like the AR/VR technique of 3D model to the authentication process. However, the performance for whole protocol of SISS largely depends on the network condition. Hence, we experiment the performance of SISS with separate eight parts as follows: (1) the generation

Table 4: The specification of a server.

		Server
Hardware	CPU	Intel Core i7-4770K 3.5 GHz
	Memory	4 GB
	Disk	200 GB
OS	Ubuntu 14.04 LTS / Linux Kernel 3.19.0	
Development Tool	gcc 4.8	

Table 5: The specification of a client.

		Client (Nexus 5X)
Hardware	CPU	Snapdragon 808 1.8 GHz
	Memory	2 GB
	Disk	16 GB
OS	Android 6.0	
Development Tool	eclipse / Android SDK	

Table 6: Cryptographic parameters and libraries.

Pseudorandom function f	AES 128 bits
Hash function h	SHA-1
Group parameter	cyclic group
Modulus P	2048 bits
Order q	256 bits
Generator g	2048 bits
Time measurement function in a Server	clock()
Time measurement function in a Client	System.currentTimeMillis()
Crypto Library of a Server	MIRACLE
Crypto Library of a Client	Android OpenSSL & Java BigInteger

time in a server including storage time, (2) the time for a client including data transfer and storage in DB, (3) $QrU1$ of login, (4) $VrSM1\_QrSM2$ of login, (5) $VrU2\_QrU3$ of login, (6) $VrSM3$ of login, (7) *EncUp*, and (8) *DnDec*

6.1. Implementation and Experimental Environment. The experimental environments of a server and a client are addressed in Tables 4 and 5.

6.2. Cryptographic Parameters and Library. Cryptographic parameters and libraries are described in Table 6.

6.3. The Results of Implementation. Table 7 shows the performance of SISS divided into eight parts. The registration process which needs only once to join the website totally takes over 20 seconds. Considering that users should generate all their information to use it through all their sessions at this registration process in advance, the estimated time is understandable and applicable to a real world in general. Other processes such as login or actions take much less than 1 second (cf. in the implementation of VrU_QrU3, the rendering process can be skipped because there is no commercialized tools for rendering until now; we replace $R_{S_i^j}$ with 20-byte 3-dimensional image).

6.4. Comparison with Other Works. The related works' main goal was only focused on the methods of group key's rekeying, revocation, and redistribution, whereas SISS's main goal is to design the information sharing protocol with safety for application website. Hence, the Storage Overhead and Communication Overhead analyzed in the related works are obviously different from SISS because our proposed group key system is developed to eliminate the processes of rekeying and redistribution that constitute hard and complicated work with heavy overheads. To the best of my knowledge, SISS's group key is the first scheme without rekeying and redistribution; nevertheless it can guarantee the security requirements of group key.

Gou et al.'s paper [50] is the latest work to analyze the performances of current schemes until now; the minimal Storage Overhead is $\log_2 p$ (p: finite field's order) and Communication Overhead is $(n+2)\,j \log_2 q$ (n: maximum revoked users, j: session, q: multiplicative group's order). As for SISS, Storage and Communication Overheads are O(0). Thus, based on Gou et al.'s work, we only compare and analyze the security performances of group key because the proposed scheme SISS has no process itself for group key's rekeying and redistribution.

Table 8 shows that Staddon et al. and Liu et al.'s schemes can guarantee only forward secrecy, and Rams et al. and Guo et al.'s new scheme can meet all properties of forward secrecy, backward secrecy, and collusion resistance. The number of Revocation Limit is the maximum for Guo et al.'s new scheme and the proposed scheme SISS. And, SISS can guarantee all security properties of forward secrecy, backward accessibility, and collusion resistance, but not backward secrecy.

7. Discussion

7.1. The Differences from Other Group Key Structure

(1) The application and aim of our group key are different from the traditional group key. Rather, they are closer to 'keyword search schemes for multiuser setting with group keys', whose security requirements are forward secrecy and backward accessibility; the leaving members should not know the group's documents, and newly joining members should know the previous documents of the group to perform the group's tasks. In the sense of sharing information among group members, we used the term group key.

(2) The formation structure of the group key is completely different. General group key systems make every user share the same group key for the session. However, in SISS, on the basis of the master key, random numbers and other things are combined, where the random number has a different initial value for each user, which is hashed $(s-1)$ times for s sessions (total number of sessions is s). Finally, each user

Table 7: The result.

Operation		Process time (ms)
Registration (1000 Session)	The generation time in a server including storage time	6138 ms (6.1 s)
	The time for a client including data transfer and storage in DB	14499 ms (14.4 s)
Login	$QrU1$	16 ms (0.016 s)
	$VrSM1\_QrSM2$	2 ms (0.002 s)
	$VrU2\_QrU3$	56 ms (0.056 s)
	$DVrSM3$	skip (comparison between simple values)
EncUp		54 ms (0.054 s)
DnDec		32 ms (0.032 s)

Table 8: The comparison of group key security performance.

Scheme	Revocation Limit	Forward Secrecy	Backward Secrecy	Collusion Resistance
Staddon et al.	n	Yes	No	No
Liu et al.	n	Yes	No	No
Rams et al.	n	Yes	Yes	Yes
Guo et al.'s basic scheme	n	Yes	No	No
Guo et al.'s new scheme	ns	Yes	Yes	Yes
SISS	ns	Yes	Backward Accessibility	Yes

n: maximum revoked users, s: maximum session.

has different group keys for each session and does not share any key with any one, except for K_3 only. The important thing is that even members themselves do not know their group's master key because it is masked with random numbers. At registration, s hashed values generated through hash chain are stored in only SM's server. Each user has only $S_{t,i}^{j}$, p_i^1, K_{Gi}^{1}, $\{h(E_{K_{Gi}^{j}}(p_i^j))\}$, $h(E_{K_{Gi}^{1}}(p_i^1 \parallel S))$, $(1 \le i \le n, 1 \le j \le q)$ for the authentic information required at the start of the session as mentioned in the algorithm $InfGenStr$.

(3) The result of encryption/decryption with the group's master key and the result of encryption/decryption with the every member's group key are the same (refer to [Additional Explanation for Encryption (E) and Decryption (D) with Group Keys] in Section 4). This is because the developed equation (algorithm) is designed according to the principle that all random numbers attached before the computation should be removed after the computation. It makes rekeying and redistributing of the group key unnecessary for SISS. In SISS, members can upload only on their web pages, while download can be done on their own web pages and those of other users (valid users). Therefore, members encrypt the information that they want to share with the group key and encrypt the information that they want to be secret with their private keys.

(4) It can be said that the group key renewals for session changes are accomplished in the authentication processes of login for each member. In other words, the group key and pseudonym key for each user's next session are given by the SM at the end of the login process, which serve more as authenticators to pass the login process. If any member does not receive the group key and pseudonym key for the next session from the SM, the value can never be deduced. The reasons are as follows: (1) Group keys have an effect similar to a one-time password because they have completely different values for each member and for each session. (2) The master key and the random number cannot be inferred because of the combined characteristics (safety) between master key and random number such as DDH, DLP, and other cryptographic functions. (3) Due to the hash chain's one-wayness, which is the method of random number generation, we never know the random number of the next session, so we do not know the group key value of the next session

(5) The leave and revocation process of SISS is also different from the general group key because SISS does not have a rekeying and redistributing process. When SM receives the leave request from a member, the SM enters the revocation process, records the member's id in the leave-list, and deletes the user's hash chain and other additional information. Even if a member who has left a session tries to log in with the next session information which is received from the previous session, the member cannot pass the authentication because all information of the user has been removed from SM's server. And the member cannot receive the next session information any longer. In other words, a member can no longer log in to the group if the member leaves the group, so that the member should download all the previous information before requesting leave. The leaving members can never know the next subsequent information, while newly joining members can decrypt all the shareable information encrypted with the group key.

(6) The meaning of a session of SISS is different from other general group key systems that consider the session as the number of membership-changes, as SISS considers

the session as the number of logins for each member. If a member has performed a total of s logins, then the member can reconnect to SM and generate a new hash chain again as he did at the registration time. The total number of sessions, s, can be determined by the policies of the website or by the needs of individual members.

7.2. Legitimate Group Member. In the last step of the login authentication, 3-dimensional image R_S is rendered. R_S plays a role of a LGM (legitimate group member) which is decided with SM at the registration time. The goal is "improving authentication and security against conspiracy and compromise". If 3-dimensional image is inefficient in a real world, 2-dimensional image is recommendable.

In 2016, Google's project 'Tango' has been showcased with indoor mapping and VR/AR platform [52]. 'Tango' technology enables a mobile device to measure the physical world. Tango-enabled devices (smartphones, tablets) are used to capture the dimensions of physical space to create 3D representations of the real world. 'Tango' gives the Android device platform the new ability of spatial perception.

According to JPMorgan analyst Rod Hall [7], Apple expects that iPhone 8 would feature a front-facing 3D laser scanner for facial recognition. It can be also said that the facial recognition will potentially be more secure than Touch ID, and 3D laser scanner could eventually be used for other purposes such as augmented reality. Unfortunately, however, the iPhone 8 released in 2017 did not have the expected function of front-facing 3D laser scanner. Even though the released AR technique of iPhone 8 was different from the 3D laser scanner for facial recognition by Rod Hall [7], we can anticipate the generalized AR technique for the facial recognition in the near future. Therefore, we can say that the proposition of SISS is timely good to apply LGM to the real world keeping abreast of Tango and iPhone's AR/VR technique of mobile devices.

7.3. Privacy Preserving SISS. SISS can meet the privacy requirements as follows:

(1) Anonymity and Pseudonymity: In SISS, each member uses different pseudonymity for each session. Although perfect anonymity cannot be provided, pseudonymity can be provided instead.

(2) Unlinkability: Every session, users log in with different pseudonyms (P) and use different encryption keys (each member's group key). Consequently, SISS can achieve unlinkability and similar level of security to 'One-Time Encryption'.

(3) Unobservability: All information is encrypted by members' group keys, which have different values by being masked with the differently generated random numbers for each user and each session [53].

7.4. Mutual Authentication. An attacker may try to pretend to be a valid member to log in to the SIS system or masquerade as an SM server to extract users' information. This property is about spoofing attack.

The authentication between a member and the SM server is accomplished through the query and verification algorithms: Qr, Vr_Qr, Vr. Specifically, authentication processes for login consist of $QrU1(RH, K, P)$, $VrSM1\_QrSM2(QvU1)$, $VrU2\_QrU3(QvSM2)$, and $VrSM3(R_{S_i^j})$. In $QrU1(RH, K, P)$, a member queries the SM with $QvU1$ which is the computed values using the stored values at the registration time. Then, in $VrSM1\_QrSM2(QvU1)$, the SM server verifies the value $QvU1$ with the stored values, too. After the successful verification, the SM server queries the member with $QvSM2$, which is also computed using the stored values and $QvU1$. To the last processes $VrSM3(R_{S_i^j})$, the member and the SM server authenticate each other using the stored values, respectively.

From a member to the SM server, if the SM server can obtain the corresponding rightly rendered image in the last authentication process, it means that the SM server is the real server to which a member wants to log in and the member is a valid user to be registered in advance.

8. Conclusion

SISS is the proposal for the patients from all over the world who want to get some help and share information through websites such as 'PatientsLikeMe' or 'CureTogether'. The proposed model SISS can guarantee security and privacy for the sensitive health and private information. As for the main method of group key management system, SISS addressed the hard problems of rekeying and redistribution, conspiracy, and backward accessibility with new ideas such as equations for encryption/decryption and LGM. Moreover, SISS is scalable to general group's project applications with safety. Therefore, it is clear that the problem of information sharing and the approaches between collaborative computing and security should be managed as Integrated Security Management (ISM).

Data Availability

No data were used to support this study. The main method is an encryption algorithm, so that the information in this paper is randomly generated.

Conflicts of Interest

The author declares no conflicts of interest.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by Korea Government (Ministry of Education, NRF-2017R1D1A1B03029488). The author appreciates Ph.D. candidate Park, Jin Hyung for implementing performance.

References

[1] https://www.patientslikeme.com.

[2] https://curetogether.com.

[3] J. Eom, D. H. Lee, and K. Lee, "Patient-controlled attribute-based encryption for secure electronic health records system," *Journal of Medical Systems*, vol. 40, article no 253, 2016.

[4] Y. Kim, A. Perrig, and G. Tsudik, "Communication-efficient group key agreement," in *Proceedings of the International Information Security Conference*, vol. 65 of *IFIP Advances in Information and Communication Technology*, pp. 229–244, Springer, Paris, France, 2001.

[5] H.-A. Park, J. H. Park, and D. H. Lee, "PKIS: practical keyword index search on cloud datacenter," *EURASIP Journal on Wireless Communications and Networking*, vol. 84, no. 8, pp. 1364–1372, 2011.

[6] Y. Hu, A. Perrig, and D. B. Johnson, "Efficient security mechanisms for routing protocols," in *Proceedings of the NDSS'03*, pp. 57–73, February 2003.

[7] L. Rehm, 2017, https://www.dpreview.com/news/3278771312/iphone-8-expected-to-replace-touch-id-with-3d-facial-recognition.

[8] H. Park, J. W. Byun, and D. H. Lee, "Secure index search for groups," in *Proceedings of the TrustBus'05*, vol. 3592 of *Lecture Notes in Computer Science*, pp. 128–140, Springer, Berlin, Germany, 2005.

[9] P. Wang, H. Wang, and J. Pieprzyk, "Common secure index for conjunctive keyword-based retrieval over encrypted data," in *Proceedings of the SDM'07*, vol. 4721 of *Lecture Notes in Computer Science*, pp. 108–123, 2007.

[10] P. Wang, H. Wang, and J. Pieprzyk, "Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups," in *Proceedings of the CANS'08*, vol. 5339 of *Lecture Notes in Computer Science*, pp. 178–195, 2008.

[11] J. Li and X. Chen, "Efficient multi-user keyword search over encrypted data in cloud computing," *Computing and Informatics*, vol. 32, no. 4, pp. 723–738, 2013.

[12] D. Wallner, E. Harder, and R. Agee, "Key management for multicast: issues and architecture," IETF RFC 2627, 1999.

[13] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 16–30, 2000.

[14] M. Zheng, J. Zhu, and G. Cui, "A hybrid group key management scheme for two-layered Ad Hoc networks," in *Proceedings of the 9th International Conference on Information Technology (ICIT'06)*, pp. 83-84, Bhubaneswar, India, 2000.

[15] A. Perrig, D. Song, and J. D. Tygar, "ELK, a new protocol for efficient large-group key distribution," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pp. 247–262, USA, May 2001.

[16] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam, "Batch rekeying for secure group communications," in *Proceedings of the 10th International Conference on World Wide Web, (WWW '01)*, pp. 525–534, Hong Kong, May 2001.

[17] Y. Chen, J. D. Tygar, and W. Tzeng, "Secure group key management using uni-directional proxy re-encryption schemes," in *Proceedings of the Conference on Computer Communications (INFOCOM '11)*, pp. 1952–1960, Shanghai, China, April 2011.

[18] J. A. Naranjo, N. Antequera, L. G. Casado, and J. A. Lopez-Ramos, "A suite of algorithms for key distribution and authentication in centralized secure multicast environments," *Journal of Computational and Applied Mathematics*, vol. 236, no. 12, pp. 3042–3051, 2012.

[19] P. Vijayakumar, S. Bose, and A. Kannan, "Centralized key distribution protocol using the greatest common divisor method," *Computers & Mathematics with Applications*, vol. 65, no. 9, pp. 1360–1368, 2013.

[20] L. Harn and C. Lin, "Authenticated group key transfer protocol based on secret sharing," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 59, no. 6, pp. 842–846, 2010.

[21] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, article no 197, pp. 644–654, 1976.

[22] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Proceedings of the Advances in Cryptology, EUROCRYPT 1994*, vol. 950 of *Lecture Notes in Computer Science*, pp. 275–286, Springer, Berlin, Germany, 1994.

[23] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, pp. 769–780, 2000.

[24] A. Joux, "A one round protocol for tripartite Diffie-Hellman," *Journal of Cryptology*, vol. 17, no. 4, pp. 263–276, 2004.

[25] J. Katz and M. Yung, "Scalable protocols for authenticated group key exchange," in *Proceedings of the Advances in Cryptology, (CRYPTO '03)*, vol. 2729 of *Lecture notes in computer science*, pp. 110–125, Springer, Berlin, Germany, 2003.

[26] Q. Wu, Y. Mu, W. Susilo, B. Qin, and J. Domingo-Ferrer, "Asymmetric group key agreement," in *Proceedings of the EUROCRYPT'09*, vol. 5479 of *Lecture Notes in Computer Science*, pp. 153–170, Springer, Berlin, Germany, 2009.

[27] L. Zhang, Q. Wu, B. Qin, and J. Domingo-Ferrer, "Provably secure one-round identity-based authenticated asymmetric group key agreement protocol," *Information Sciences*, vol. 181, no. 19, pp. 4318–4329, 2011.

[28] X. Lv, H. Li, and B. Wang, "Group key agreement for secure group communication in dynamic peer systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 10, pp. 1195–1200, 2012.

[29] S. Mittra, "Iolus: a framework for scalable secure multicasting," in *Proceedings of the ACM SIGCOMM*, vol. 27, pp. 277–288, New York, NY, USA, 1997.

[30] L. R. Dondeti, S. Mukherjee, and A. Samal, "Scalable secure one-to-many group communication using dual encryption," *Computer Communications*, vol. 23, no. 17, pp. 1681–1701, 2000.

[31] A. T. Sherman and D. A. McGrew, "Key establishment in large dynamic groups using one-way function trees," *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 444–458, 2003.

[32] D.-W. Kwak and J. W. Kim, "A decentralized group key management scheme for the decentralized P2P environment," *IEEE Communications Letters*, vol. 11, no. 6, pp. 555–557, 2007.

[33] C. Blundo, P. D'Arco, A. De Santis, and M. Listo, "Design of self-healing key distribution schemes," *Designs, Codes and Cryptography. An International Journal*, vol. 32, no. 1-3, pp. 15–44, 2004.

[34] C. Blundo, P. D'Arco, and A. De Santis, "Definitions and bounds for self-healing key distribution schemes," in *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP '04)*, vol. 3142 of *Lecture Notes in Computer Science*, pp. 234–245, Turku, Finland, 2004.

[35] C. Blundo, P. D'Arco, and A. De Santis, "On self-healing key distribution schemes," *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, vol. 52, no. 12, pp. 5455–5467, 2006.

[36] R. Dutta, "Anti-collusive self-healing key distributions for wireless networks," *International Journal of Wireless and Mobile Computing*, vol. 7, no. 4, pp. 362–377, 2014.

[37] H. Chen and L. Xie, "Improved one-way hash chain and revocation polynomial-based self-healing group key distribution schemes in resource-constrained wireless networks," *Sensors*, vol. 14, no. 12, pp. 24358–24380, 2014.

[38] R. Dutta, S. Mukhopadhyay, and M. Collier, "Computationally secure self-healing key distribution with revocation in wireless ad hoc networks," *Ad Hoc Networks*, vol. 8, no. 6, pp. 597–613, 2010.

[39] R. Dutta, S. Mukhopadhyay, A. Das, and S. Emmanuel, "Generalized self-healing key distribution using vector space access structure," in *Proceedings of the 7th International IFIP-TC6 Networking Conference on Ad Hoc and Sensor Networks*, vol. 4982 of *Lecture Notes in Computer Science*, pp. 612–623, Springer, Berlin, Germany, 2008.

[40] G. Saez, "On threshold self-healing key distribution schemes," in *Proceedings of the Cryptography and Coding, LNCS3796*, vol. 3796 of *Lecture Notes in Computer Science*, pp. 340–354, Springer, Berlin, Germany, 2005.

[41] B. Tian, S. Han, T. S. Dillon, and S. Das, "A self-healing key distribution scheme based on vector space secret sharing and one way hash chains," in *Proceedings of the 2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM'08)*, pp. 1–6, Newport Beach, Calif, USA, June 2008.

[42] X. Du, Y. Wang, J. Ge, and Y. Wang, "An ID-based broadcast encryption scheme for key distribution," *IEEE Transactions on Broadcasting*, vol. 51, no. 2, pp. 264–266, 2005.

[43] S. Han, B. Tian, Y. Zhang, and J. Hu, "An effieient self-healing key distribution scheme with constant-size personal keys for wireless sensor networks," in *Proceedings of the IEEE International Conference on Communications (ICC '10)*, pp. 1–5, Cape Town, South Africa, May 2010.

[44] T. Rams and P. Pacyna, "A survey of group key distribution schemes with self-healing property," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 820–842, 2013.

[45] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean, "Self-healing key distribution with revocation," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 241–257, Berkeley, UK, May 2002.

[46] T. Rams and P. Pacyna, "Self-healing group key distribution with extended revocation capability," in *Proceedings of the International Conference on Computing, Networking and Communications, (ICNC '13)*, pp. 347–353, San Diego, Calif, USA, January 2013.

[47] T. Rams and P. Pacyna, "Long-lived self-healing group key distribution scheme with backward secrecy," in *Proceedings of the 1st International Conference on Networked Systems, (NetSys '13)*, pp. 59–65, Stuttgart, Germany, March 2013.

[48] D. Liu, P. Ning, and K. Sun, "Efficient self-healing group key distribution with revocation capability," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pp. 27–31, Washington, D.C., USA, October 2003.

[49] D. Hong and J.-S. Kang, "An efficient key distribution scheme with self-healing property," *IEEE Communications Letters*, vol. 9, no. 8, pp. 759–761, 2005.

[50] H. Guo, Y. Zheng, X. Zhang, and Z. Li, "Exponential arithmetic based self-healing group key distribution scheme with backward secrecy under the resource-constrained wireless networks," *Sensors*, vol. 16, no. 5, article no 609, 2016.

[51] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P '00)*, pp. 44–55, IEEE, Berkeley, Calif, USA, May 2000.

[52] G. Sterling, "Google to showcase project tango indoor mapping and VR/AR platform at Google I/O. The ambitious initiative brings the digital and physical even closer together," 2016, http://searchengineland.com/google-showcase-project-tango-indoor-mapping-vrar-platform-google-io-249629.

[53] H.-A. Park, J. Zhan, and D. H. Lee, "PPSQL: privacy preserving SQL queries," in *Proceedings of the 2nd International Conference on Information Security and Assurance, (ISA'08)*, pp. 549–554, Republic of Korea, April 2008.

Hindawi
Security and Communication Networks
Volume 2019, Article ID 7140480, 11 pages
https://doi.org/10.1155/2019/7140480

Research Article

Mining the Key Nodes from Software Network Based on Fault Accumulation and Propagation

Huang Guoyan,[1,2] Wang Qian ,[1,2] Liu Xinqian ,[1,2] Hao Xiaobing,[1,2] and Yan Huaizhi[3]

[1]*College of Information Science and Engineering, Yanshan University, Qinhuangdao, Hebei 066000, China*
[2]*Computer Virtual Technology and System Integration Laboratory of Hebei Province, 066000, China*
[3]*Beijing Key Laboratory of Software Security Engineering Technique, School of computer Science and Technology, 5 South Zhongguancun Street, Haidian District, Beijing 100081, China*

Correspondence should be addressed to Wang Qian; wangqianysu@163.com

Received 14 October 2018; Accepted 20 January 2019; Published 7 March 2019

Guest Editor: Chunhua Su

The increasement of software complexity directly results in the augment of software fault and costs a lot in the process of software development and maintenance. The complex network model is used to study the accumulation and accumulation of faults in complex software as a whole. Then key nodes with high fault probability and powerful fault propagation capability can be found, and the faults can be discovered as soon as possible and the severity of the damage to the system can be reduced effectively. In this paper, the algorithm MFS_AN (mining fault severity of all nodes) is proposed to mine the key nodes from software network. A weighted software network model is built by using functions as nodes, call relationships as edges, and call times as weight. Exploiting recursive method, a fault probability metric FP of a function, is defined according to the fault accumulation characteristic, and a fault propagation capability metric FPC of a function is proposed according to the fault propagation characteristic. Based on the FP and FPC, the fault severity metric FS is put forward to obtain the function nodes with larger fault severity in software network. Experimental results on two real software networks show that the algorithm MFS_AN can discover the key function nodes correctly and effectively.

1. Introduction

With the development of computer technology and the expansion of software applications [1], the scale and complexity of software systems increase continuously. Software faults directly lead to the rise of system failure ratio, and their reliability is becoming more and more difficult to guarantee. In the test and maintenance process, developers cannot deal with the software problems with a clear purpose [2]. Therefore, if some potentially useful information can be found from software source code or dynamic execution process to help software workers understand the structural characteristics of software quickly, it will be of great significance for improving software development and maintenance efficiency [3–5]. Affected by the achievements in the complex network field, some researchers regard software system as a software network for scientific research. This provides a novel research idea and platform for better understanding and measuring the internal topology structure of complex software system and receives great attention.

The knowledge of complex network has been introduced into software engineering by using network model to represent the structural characteristics of a software system, and researchers have found many novel features of the structure from different points of view [6, 7]. Valerde et al. [8] apply complex network to construct the topology structure of software and propose a method to model the software as an undirected network for the first time. In the method, the node is regarded as the software class and the edge is regarded as the call relationship among the classes. With experiments, they find the "scale-free" and "small-world" properties in software network. Myers et al. [9] use directed network to represent the collaboration relationship among the classes of software. They learn that indegree and outdegree distribution of the network obey the power-law distribution with different exponents. Pan et al. [10] adopt a binary software network to

represent class units or package units and their dependence relationships in a software system, as well as a community detection method to detect the best module partition of the software system. The optimal module partition is compared with the real module partition in the system to guide the optimization design of the module when a software version is updated. Thung et al. [11] propose a new method to simplify the complexity of a software network. Then, they measure the importance of classes from different properties (betweenness, closeness, etc.). Furthermore, they condense the class network which only contains some important classes. By the method, they are able to depict the overall design for software and make the design model easy to understand. Mohammed et al. [12] construct a mapping of research system to identify software security techniques used in the software development process, which enables software developers to understand the existing software security approach better. Thus, software security problems are urgent to be solved.

Measuring the importance of nodes accurately in software network is the premise to improve the security and reliability of software [13]. In software network, a few key nodes have an important effect on the overall stability, reliability, and robustness of the system [14], such as the impact of cascading failure propagation. If there are faults in these nodes, it can result in partial or total system crashes and irreversible results. Identifying these nodes and providing them with key protection help to prevent system crash caused by deliberate attacks. Researchers have defined the importance of nodes in software network from different aspects. Freeman [15] utilizes the betweenness to measure the node importance and points out that a node is more important in software network if its betweenness is bigger. Callaw et al. [16] consider that a node is more important in software network if its degree is bigger, because the node with bigger degree connects with more nodes. However, it does not consider the overall structure of a software network and has some limitations. Kitsak [17] makes it clear that the location of a node in network has a great impact on its importance and exploits the k-shell decomposition method to measure the node importance. The metric result is proved to be better than the betweenness and degree of centrality. Turnu et al. [18] measure the quality of software by analysing the degree distribution of nodes in a software network. They define the structure entropy to describe the degree distribution of nodes and prove that the statistical information of the structure entropy in a software network can be related to the number of software bugs. It further proves that there is a relationship between the structure characteristics of a software network and the quality of the software. Wang et al. [19] define the influential nodes in a network by studying the weighted software network at the function level. They analyse the relationship between the statistical characteristics of software network and the influential nodes through experiments. Bhattacharya et al. [20] predict software evolution based on the static graph topology analysis and propose NodeRank value to measure the importance of a node. The fault of a function is not only caused by itself but also affected by other functions. Huang et al. [21] define the importance of a node based on the dependence relationship and the information propagation among functions. Their algorithm MIN can effectively mine the influential nodes in a software network, but its assignment to the probability of information propagation has certain subjectivity.

In complex networks, random walk model judges the importance of a node by considering its own connectivity degree and the importance of neighbouring nodes around it. Typical methods are PageRank, NodeRank [20], and so on. In software network, the CK metric set proposed by Chidamber and Kemerer [22] indicates that the number of classes that are coupled to a given class named CBO can affect the propensity of the class node to contain defects. If the CBO of a class is larger, it is more sensitive when other parts change. So it is harder for software workers to maintain. Ren et al. [23] believe that the more numbers of modules, classes, or functions that are directly or indirectly dependent on the function nodes, the greater the cost of constructing it and the probability of error. Ren et al. [24] also believe that when a function node is the role of the calling function, it may accumulate the defect of the called function node with a certain probability. When a function node is the role of the called function, it may propagate its defects to its caller with a certain probability. Based on the random walk model and combined with the directed weighted feature of software network, the following FP and FPC are proposed.

In summary, this paper focuses on the call dependence relationship among functions and the fault accumulation and propagation of dynamic execution process. Firstly, according to the dynamic execution information of software, we build a weighted software network model. Then, utilizing recursive method, the fault probability metric FP of a function is defined in accordance with fault accumulation characteristic, and the fault propagation capability metric FPC of a function is proposed on the basis of fault propagation characteristic. Finally, by combining FP and FPC, the fault severity metric FS is put forward and the algorithm MFS_AN (mining fault severity of all nodes) is proposed to calculate the FS and obtain the function nodes with larger fault severity in software network.

The rest of this paper is organized as follows. Section 2 describes the process of mining key nodes in a software network based on the fault accumulation and propagation. The experiment results are given in Section 3. Conclusion and future work are mentioned in Section 4.

2. Mining Key Nodes Based on Fault Accumulation and Propagation

2.1. Weighted Function Execution Network. In software network, the different execution times among functions reflect the tightness degree of nodes' interaction. In order to incorporate this difference, this paper constructs a weighted software network model.

Definition 1 (WFEN (Weighted Function Execution Network)).

$$WFEN = (NSet, ESet, Weight) \tag{1}$$

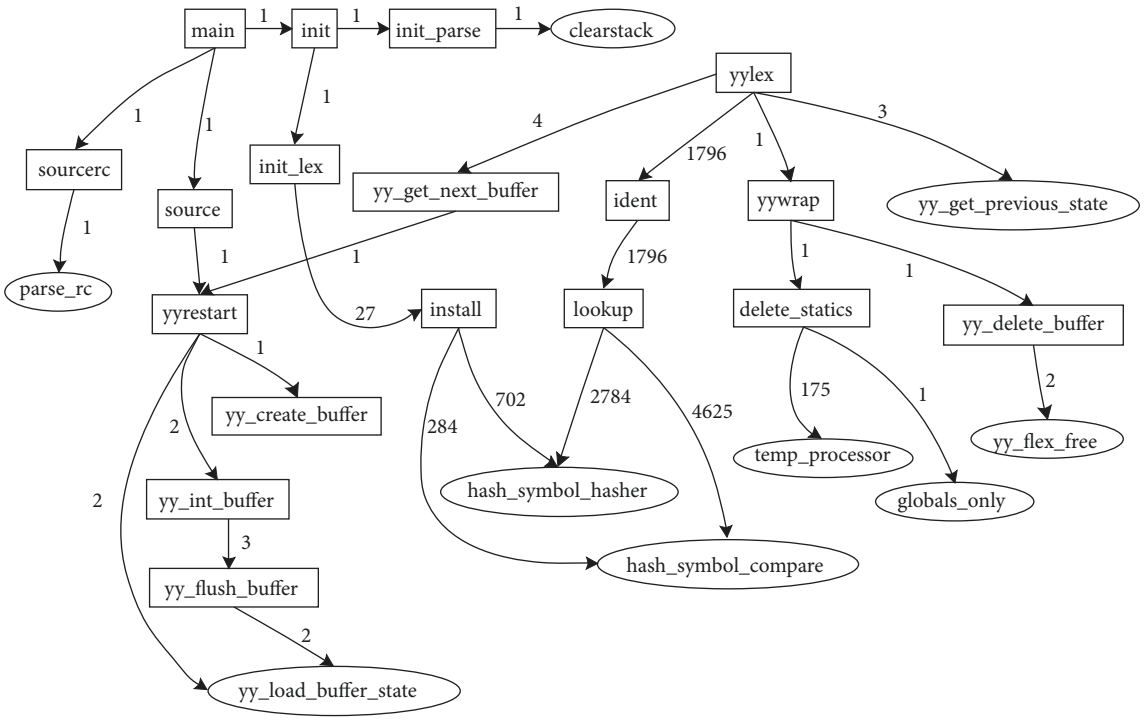


Figure 1: A portion of a weighted function execution network.

where NSet is the function node set of a software network, ESet is the edge set which is the function call relationship during the software execution process, and Weight denotes the execution times that a function calls another one.

Figure 1 represents a portion of a weighted function execution network.

As the software system works, a function is a calling function and also a called function. In the execution process of a function node u, the nodes called directly by u are the direct outdegree neighbor node of u, and the set of these direct out-degree neighbor nodes is called as the Direct Out-degree Neighbor Set (DONS). Similarly, set of the indegree neighbor nodes which call node u directly is named as the Direct In-degree Neighbor Set (DINS).

Definition 2 (DONS (Direct Out-degree Neighbor Set)).

$$DONS(u) = \{v_i \mid u \longrightarrow v_i\}, \quad u, v_i \in NSet \tag{2}$$

Definition 3 (DINS (Direct In-degree Neighbor Set)).

$$DINS(u) = \{v_i \mid v_i \longrightarrow u\}, \quad u, v_i \in NSet \tag{3}$$

2.2. The Fault Probability. Figure 2 shows a more common topology structure of software network. By analyzing these three different topologies, we study the fault accumulation characteristics of functions in software network and obtain the fault probability of a function.

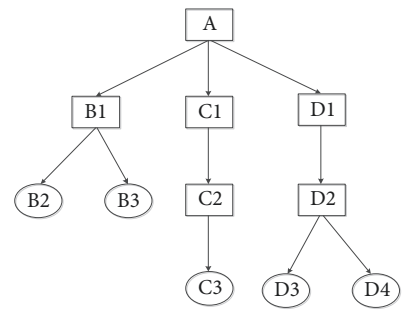


Figure 2: Common topologies in software network.

For nodes B1 and C1, they have the same size of the call function set; that is, the number of call nodes is equal, but the call relationship between these nodes is different. For nodes B1 and D1, they have the same size of execution routes, but the node D1 has a larger call function set and a more complex execution process. Therefore, the influences of the node B1, C1, and D1 on the node A are different.

With the structure shown in Figure 2, we can learn that the function fault is caused not only by itself, but also by its call functions. Moreover, for the objective function, each node in its DONS has different influence on the objective function. For this, we define the fault probability quantitative standard FP of each node which is the accumulation of infection coming from its call nodes. Based on the call relationships among the functions in DONS, the computational formula of the FP is given as follows.

Definition 4 (FP (the fault probability of a node)).

$$FP(u) = \alpha + \sum_{i=1}^{N} P_{u \longrightarrow v_i} * FP(v_i), \quad v_i \in DONS(u) \tag{4}$$

$$P_{u \longrightarrow v_i} = \frac{Weight(u, v_i)}{\sum_{j=1}^{n} Weight(j, v_i)}, \quad j \in DINS(v_i) \tag{5}$$

where α is the fault probability of u caused by itself ($0 \leq \alpha \leq 1$), v_i is a node in $DONS(u)$, N is the size of the $DONS(u)$, $P_{u \longrightarrow v_i}$ is the probability infected by the direct neighbors of u, j is a node in $DINS(v_i)$, and n is the size of the $DINS(v_i)$.

Example 5. Figure 3 is a simple weighted function execution network.

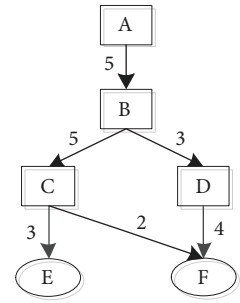


Figure 3: A simple weighted function execution network.

In Figure 3, an example shows how to calculate the FP of a function node. In real world, the size of each function with various definitions is different, and the fault probability of each function is also different. But the setting of specific fault values is more complicated. The main work of this paper is to show the correlation of faults and not to pay attention to the fault calculation method of the node itself. To be universal, we set the fault probability of function node itself to 0.5. That is to say, suppose the probability of fault occurring and not occurring is the same. The function node set NSet = {A, B, C, D, E, F}, and the Direct Out-Degree Neighbor Set of each node is as follows:

$$\begin{aligned} DONS(A) &= \{B\}; \\ DONS(B) &= \{C, D\}; \\ DONS(C) &= \{E, F\}; \\ DONS(D) &= \{F\}; \\ DONS(E) &= DONS(F) = \Phi. \end{aligned} \tag{6}$$

For nodes E and F, which belong to leaf nodes in the software network, then $FP(E) = FP(F) = \alpha = 0.5$; according to the definition of FP, the fault probability of other nodes in the software network is as follows:

$$\begin{aligned} FP(C) &= \alpha + \left(\frac{3}{3} * FP(E) + \frac{2}{2+4} * FP(F)\right) \\ &= 1.1666667; \\ FP(D) &= \alpha + \left(\frac{4}{2+4} * FP(F)\right) = 0.8333334; \\ FP(B) &= \alpha + \left(\frac{5}{5} * FP(C) + \frac{3}{3} * FP(D)\right) = 2.5; \\ FP(A) &= \alpha + \left(\frac{5}{5} * FP(B)\right) = 3.0. \end{aligned} \tag{7}$$

Through the calculation of FP, the fault probability of a function node is identified. According to the above calculation results, the fault probability of each function node in Figure 3 is in the order of: A>B>C>D>E=F.

The fault probability of a node (FP) in Definition 4 is really not probability. It is just a metric of a node, if the FP of a node is higher and the node more likely has faults. So it may take arbitrary large value. Just because FP measures recursive weighted out degree of node u, it happens to embody the process of fault accumulation in a software system. Therefore, in the case of probability cumulative, total more than 1, this is a possibility, not restricted by 1.

Via the above analysis, based on the fault accumulation characteristics of a function and the recursive method, we utilize the formula (4) to calculate the fault probability FP for each function in software network. Then the algorithm MFP_AN (Mining fault probability of all nodes) is proposed to get the FP of all nodes in software network.

In Algorithm 6, we show the process of the method MFP_AN. In line (1), we first initialize a FPList to store the information and fault probability FP of all function nodes. In lines (2-9), a looping procedure calling the procedure MFP to calculate and store the FP for all function nodes is presented. In the procedure MFP, we show the process that the FP of a node is calculated by a recursively process. In line (1), we first define and initialize some related variables. Lines (2-21) describe the process to compute the current node affected by its out-degree neighbor nodes recursively and obtain the FP of the target node.

Algorithm 6 (mining fault probability of all nodes (MFP_AN)).

Input: node set NSet, out-degree adjacency table outDegreeList, in-degree adjacency table inDegreeList

Output: the measuring result FPList of nodes

Process:

```
(01) Initialize FPList;
(02) for (each U ∈ NSet)
(03)     if (tempList.contains(U))
(04)         FPList.add(U,tempList.get(U));
(05)         break;
(06)     end if
(07)     value = MFP (U,outDegreeList,inDegreeList);
(08)     FPList.add(U,value);
(09) end for
(10) return FPList;
```

Procedure (MFP (U: a node; outDegreeList: out-degree adjacency table; inDegreeList: in-degree adjacency table))

```
(01) Initialize α, Fa=0, P=0, FP=0, tempList;
(02) if (outDegreeList[U.index]!=null)
(03)   for (each V ∈ outDegreeList[U.index])
(04)     if (inDegree[V.index]!=null)
(05)       Initialize sum=0;
(06)       for (each I ∈ inDegree[V.index])
(07)         count = Label(I,V);
(08)         sum += count;
(09)       end for
(10)       for (each I ∈ inDegree[V.index])
(11)         if (I.index == U.index)
(12)           count_UV = Label(U,V);
(13)           P = count_UV/sum;
(14)           break;
(15)         end if
(16)       end for
(17)     end if
(18)     Fa += P * MFP
         (V,outDegreeList,inDegreeList);
(19)   end for
(20) end if
(21) FP= α + Fa;
(22) tempList.add(U,FP);
(23) return FP;
```

2.3. The Fault Propagation Capability. Similarly, this section defines the fault propagation capability metric FPC of a function according to the fault propagation characteristics.

Definition 7 (FPC (the fault propagation capability of a node)).

$$FPC(u) = \frac{K_u^{in}}{K_{\max}^{in}} + \sum_{i=1}^{N} P_{v_i \longrightarrow u} * FPC(v_i), \quad v_i \in DINS(u) \tag{8}$$

$$P_{v_i \longrightarrow u} = \frac{Weight(v_i, u)}{\sum_{j=1}^{n} Weight(v_i, j)}, \quad j \in DONS(v_i) \tag{9}$$

where K_u^{in} is the in-degree of a node u, $K_{\max}^{in}$ is the maximum value of in-degree in the network, $K_u^{in}/K_{\max}^{in}$ denotes the fault propagation capability of the objective function itself, v_i is a node in $DINS(u)$, N is the size of the $DINS(u)$, $P_{v_i \longrightarrow u}$ is the probability of u called by functions in the $DINS(u)$, j is a node in $DONS(v_i)$, and n is the size of the $DONS(v_i)$.

Example 8. Based on Figure 3, an example shows how to calculate the FPC of a function node. The function node set NSet={A, B, C, D, E, F}, $K_{\max}^{in}$=2 and the Direct In-Degree Neighbor Set of each node is as follows:

$$\begin{aligned} &DINS(A) = \Phi; \\ &DINS(B) = \{A\}; \\ &DINS(C) = DINS(D) = \{B\}; \\ &DINS(E) = \{C\}; \\ &DINS(F) = \{C, D\}. \end{aligned} \tag{10}$$

According to the definition of FPC, the fault propagation capability of all nodes in the software network is as follows:

$$\begin{aligned} FPC(A) &= \frac{K_A^{in}}{K_{\max}^{in}} = 0; \\ FPC(B) &= \frac{K_B^{in}}{K_{\max}^{in}} + \left(\frac{5}{5} * FPC(A)\right) = 0.5; \\ FPC(C) &= \frac{K_C^{in}}{K_{\max}^{in}} + \left(\frac{5}{5+3} * FPC(B)\right) = 0.8125; \\ FPC(D) &= \frac{K_D^{in}}{K_{\max}^{in}} + \left(\frac{3}{5+3} * FPC(B)\right) = 0.6875; \\ FPC(E) &= \frac{K_E^{in}}{K_{\max}^{in}} + \left(\frac{3}{3+2} * FPC(C)\right) = 0.9125; \\ FPC(F) &= \frac{K_F^{in}}{K_{\max}^{in}} + \left(\frac{2}{3+2} * FPC(C) + \frac{4}{4} * FPC(D)\right) \\ &= 2.0125. \end{aligned} \tag{11}$$

Through the calculation of FPC, the fault propagation capability of a function node is identified. According to the above calculation results, the fault propagation capability of each function node in Figure 3 is in the order of F>E>C>D>B>A.

Algorithm 9 (mining fault propagation capability of all nodes (MFPC_AN)).

Input: node set NSet, out-degree adjacency table outDegreeList, in-degree adjacency table inDegreeList, inDegreeMax

Output: the measuring result FPCList of nodes

Process:

```
(01) Initialize FPCList;
(02) for (each U ∈ NSet)
(03)   if (tempList.contains(U))
```

```
(04)        FPCList.add(U,tempList.get(U));
(05)        break;
(06)     end if
(07)     value = MFPC (U,outDegreeList,
         inDegreeList,inDegreeMax);
(08)     FPCList.add(U,value);
(09) end for
(10) return FPCList;
```

Procedure (MFPC (U: a node; outDegreeList: out-degree adjacency table; inDegreeList: in-degree adjacency table; inDegreeMax: maximum value of in-degree))

```
(01) Initialize Fp=0, P=0, inDegree=0, FPC=0, tempList;
(02) if (inDegreeList[U.index]!=null)
(03)    inDegree = inDegree[U.index].size;
(04)    for (each V ∈ inDegreeList[U.index])
(05)       if (outDegree[V.index]!=null)
(06)          Initialize sum=0;
(07)          for (each I ∈ outDegree[V.index])
(08)             count = Label(V,I);
(09)             sum += count;
(10)          end for
(11)          for (each I ∈ outDegree[V.index])
(12)             if (I.index == U.index)
(13)                count_VU = Label(V,U);
(14)                P = count_VU/sum;
(15)                break;
(16)             end if
(17)          end for
(18)       end if
(19)       Fp += P * MFPC
           (V,outDegreeList,inDegreeList,inDegreeMax);
(20)    end for
(21) end if
(22) FPC= inDegree/inDegreeMax + Fp;
(23) tempList.add(U,FPC);
(24) return FPC;
```

Similarly, via the above analysis, based on the fault propagation characteristics of a function and the recursive method, we use the formula (8) to calculate the fault propagation capability FPC for each function in software network. Then the algorithm MFPC_AN (mining fault propagation capability of all nodes) is proposed to get the FPC of all nodes in software network. Algorithm 9 is similar to Algorithm 6.

2.4. The Fault Severity. Some researchers believe that the type of fault determines the behaviour of transmission [25, 26]. That is, different faults in the same software have different laws of propagation behaviour. The research focuses on the study of fault characteristics. However, other researchers believe that the system architecture determines the behaviour of fault propagation [27]. That is, the same fault in different architectures can be evolved into system failure with different types or different severity levels. This view is based on the analysis of the system structure. It focuses on the regularity of the propagation of faults in the architecture. This paper mainly studies the latter. Therefore, this article firstly calculates the fault probability FP and the fault propagation capability FPC of a function node, respectively. Then the two points are taken into account in this function node. Supposing it fails, the possible fault severity FS of the software system is calculated. Under this premise, we study the function fault characteristics of software system based on architecture.

In Sections 2.2 and 2.3, the fault probability and the fault propagation capability of a function node have been studied, respectively. The former is measured from the Out-Degree Neighbour of a function node, or to say that is the node affected by others. The latter is measured from the in-degree neighbour of a function, or to say that is the effect of the node on others. However, only a comprehensive consideration of these two aspects can fully measure the severity of the damage to a software system.

A node is more likely to have faults if its FP is higher, and it should be paid more attention. However, if a function only has faults but it does not spread its own faults, then the node will not cause very serious consequences to software system, while if a function is not only prone to fail but also has a strong capability to spread its faults to others, then it will cause very serious consequences to software system. Therefore, from the perspective of the fault severity, the fault probability FP and the fault propagation capability FPC of a function are directly proportional. The definition of FS (The fault severity) is given as follows.

Definition 10 (FS (the fault severity)).

$$FS(u) = FP(u) * FPC(u), \quad u \in NSet \tag{12}$$

where $FP(u)$ is the fault probability of a function node u and $FPC(u)$ is the failure propagation capability of u. They jointly determine the fault severity to a software system when the function node u fails. And if a node is with a bigger FS, it will have greater impact on the software system and then it is more critical.

First, we obtain the fault probability set FPList and the failure propagation capability set FPCList of software network through Algorithms 6 and 9, respectively. Then, we use formula (12) to calculate the fault severity FS, and the algorithm MFS_AN (mining fault severity of all nodes) is proposed to discover the top-k key nodes from software network.

In Algorithm 11, the process of the method MFS_AN is presented. Line (1) first initializes an empty FSList set that

stores the FS of all function nodes. Lines (2-7) present a looping process that calculates the FS. In line (8), the FS in the FSList are sorted. In Line (11), the first k function nodes in the FSList are selected as the key nodes of software network.

Algorithm 11 (mining fault severity of all nodes (MFS_AN)).

Input: node set NSet, FPList, FPCList

Output: the top-k key nodes list Knodes

Process:

(01) **Initialize** FSList;

(02) **for** (each U ∈ NSet)

(03) FP(U)=FPList.get(U);

(04) FPC(U)=FPCList.get(U);

(05) FS(U)=FP(U) ∗ FPC(U);

(06) FSList.add(U,FS(U));

(07) **end for**

(08) FSList.sort();

(09) Knodes = FSList.get(K);

(10) **return** Knodes

3. Experiment and Analysis

In this section, we verify the method MFS_AN by testing two kinds of classic tool software Tar and Cflow obtained from the open source community. Tar is a file compression and decompression tool. Cflow is a C program analysis tool for tracking the calling process of functions in the C program. In the Linux environment, we can extract the functions and the dependence relationships of open-source software with the help of tool pvtrace. The results are output to files as text (such as graph.dot). The nodes and the dependence relationships then can be graphically displayed by means of the visualization tool Graphviz. As the main function must be very important to software, so it is excluded in the following experimental verification. In addition, before the experiment, we pretreat the experimental data and delete the loop in the software network, so that recursion can be finished successfully.

3.1. The Distribution of FS. By tracking the execution process of Tar and Cflow, the dynamic execution information of the two types of software is obtained, and the weighted function execution network WFEN is constructed as the basis of experimental data. The fault probability FP and the fault propagation capability FPC of all functions are obtained by mapping the node set and the call relationships of software network to Algorithm 6 (MFP_AN) and Algorithm 9 (MFPC_AN). The return values of Algorithms 6 and 9 are mapped to Algorithm 11 (MFS_AN). The fault severity FS of all nodes and the key nodes in software network are obtained. Figures 4 and 5 show the fault severity scores and the distribution of key nodes in the different versions of Tar and Cflow.

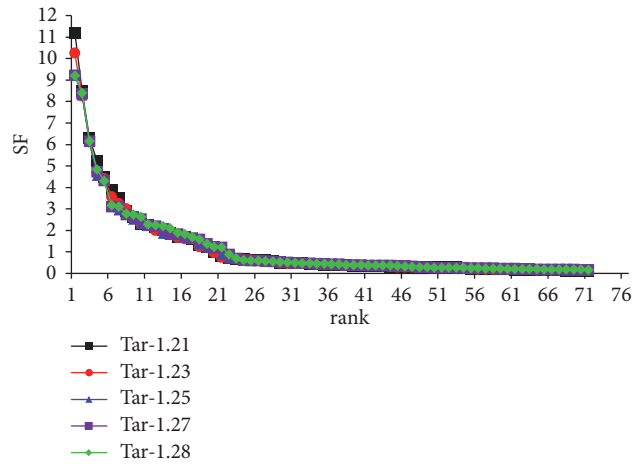


Figure 4: SF value distribution of Tar.

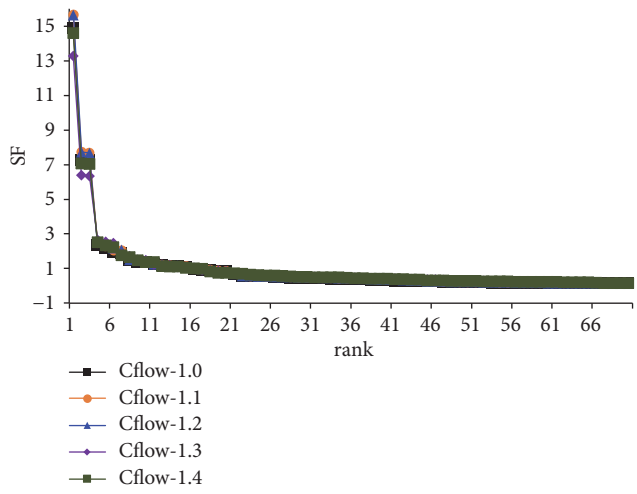


Figure 5: SF value distribution of CFlow.

From the results distribution shown in Figures 4 and 5 (the first 70 nodes are chosen because of the number of nodes in different versions is different), we can summarize the following rules:

(1) We can find that every result distribution obeys the power-law distribution. With the distribution, we verify that software network shows the scale-free properties of complex network.

(2) There are a few nodes with big FS and most of nodes with small FS. But their criticality and impact on the overall software architecture can be reflected in the higher scores.

(3) Their curves are basically at the same trend in different versions of the Tar and Cflow. In other words, in different versions, if the function nodes have the same criticality, the fault severity to software system is no big difference.

By analysing the node criticality in the two types of software from Figures 4 and 5, the key nodes in the software network are defined according to the FS, as the hierarchical structure of FS distribution is obvious, according to the turning point of curves in the graph, to select Top-10 as the key nodes of Tar and Cflow software network, respectively. In

Table 1: The rank of function nodes by SF for Tar versions.

	Rank/value				
Node	Tar-1.21	Tar-1.23	Tar-1.25	Tar-1.27	Tar-1.28
flush_archive	1/11.178	1/10.253	1/9.238	1/9.221	1/9.208
dump_file	2/8.509	2/8.256	2/8.306	2/8.357	2/8.408
dump_file0	3/6.287	3/6.120	3/6.150	3/6.150	3/6.150
find_next_block	4/5.220	4/4.676	5/4.323	5/4.313	5/4.304
update_archive	5/4.460	5/4.412	4/4.556	4/4.729	4/4.830
gnu_flush_write	6/3.849	6/3.567	6/3.112	6/3.107	7/3.103
_gnu_flush_write	7/3.475	7/3.246	8/2.744	8/2.740	9/2.737
create_archive	8/2.873	8/3.004	7/2.944	7/3.104	6/3.170
dump_regular_file	9/2.621	9/2.642	9/2.509	9/2.629	10/2.629
open_archive	10/2.311	10/2.292	10/2.299	10/2.534	8/2.768

Table 2: The rank of function nodes by SF for Cflow versions.

	Rank/value				
Node	Cflow-1.0	Cflow-1.1	Cflow-1.2	Cflow-1.3	Cflow-1.4
nexttoken	1/14.878	1/15.642	1/15.642	1/13.266	1/14.610
yylex	2/7.282	2/7.728	2/7.728	3/6.341	3/7.037
get_token	3/7.255	3/7.670	3/7.670	2/6.382	2/7.063
yyparse	4/2.370	4/2.492	4/2.530	4/2.656	4/2.514
parse_declaration	5/2.175	5/2.308	5/2.355	5/2.547	5/2.349
parse_dcl	6/1.949	7/2.046	6/2.188	6/2.455	6/2.241
expression	7/1.890	6/2.069	7/2.069	7/1.831	7/1.775
yyrestart	8/1.479	9/1.479	9/1.479	10/1.479	8/1.663
func_body	9/1.377	8/1.510	8/1.510	9/1.513	11/1.381
append_to_list	10/1.371	10/1.347	10/1.347	--	--

Table 3: In/Out-degree statistics of ranking top-5 and back-5 nodes in Cflow-1.4.

Rank	Node	K_{in}	K_{out}
1	nexttoken	14	2
2	get_token	1	1
3	yylex	1	5
4	yyparse	1	5
5	parse_declaration	1	4
-5	clear_active	1	0
-4	set_active	1	0
-3	compare	1	0
-2	depmap_alloc	1	0
-1	register_output	1	0

Tables 1 and 2, we present the key nodes and their rank for different versions of the Tar and Cflow.

From the data shown in Tables 1 and 2, the following rules can be summarized:

(1) For a given function node, the criticality ranking in different versions is basically stable. Although there is a change about the ranking of a specific function node in different software versions, the change is very small. For example, in Table 1, the ranking range of the function node find_next_block is [4, 5] and the criticality ranking of the function node dump_file has been stable at 2 in different versions.

(2) Due to the ranking stability of the key nodes in software evolution, we have sufficient reason to predict the position of a function node in a new software version. For instance, in Table 2, the function node yylex is always more critical in each version, and then we can predict that it is likely to still be more critical in the next up-to-date version.

3.2. Correctness Verification

3.2.1. Degree Distribution of FS. In order to illustrate the correctness of the key nodes, taking Cflow-1.4 as an example, we use indegree K_{in} and outdegree K_{out} these two indicators, respectively, as the criticality characterization of a function in software network.

According to the data in Table 3, it can be explained that although the criticality of a function node is not directly related to degree, they have a certain positive correlation. The outdegree values of top-5 are bigger; then their fault probability is greater, and the in-degree values are also bigger; then their fault propagation capability is greater as well, so the overall fault severity is greater. While the back-5 are all leaf

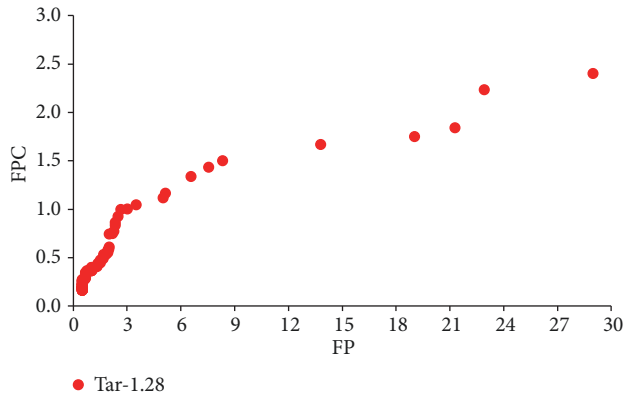


Figure 6: Joint distribution of FP and FPC in Tar-1.28.

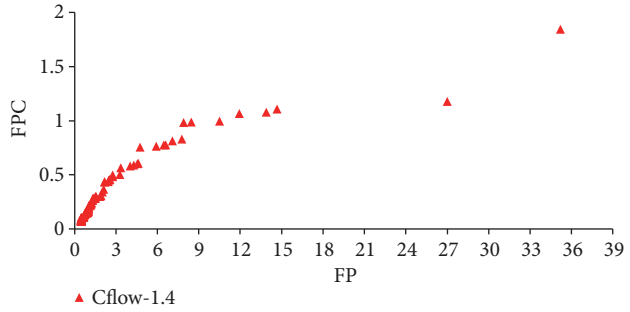


Figure 7: Joint distribution of FP and FPC in Cflow-1.4.

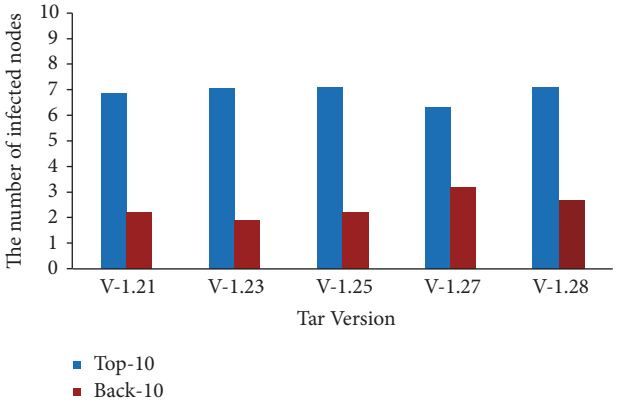


Figure 8: IC Model simulation results for different Tar versions.

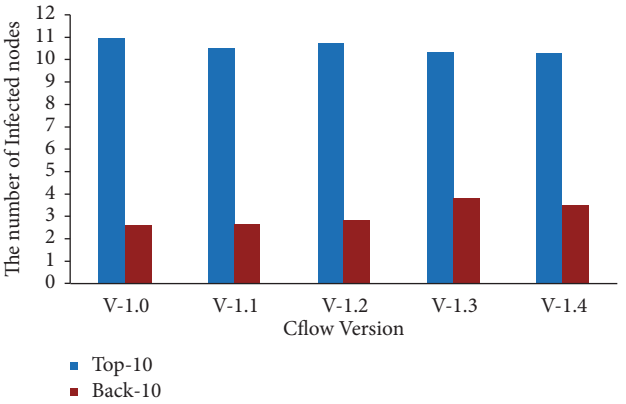


Figure 9: IC Model simulation results for different Cflow versions.

nodes with one in-degree, even if they fail, the range of fault propagation is limited. The different versions of Tar and other versions of Cflow are similar to Table 3 and they will not be described in detail here.

3.2.2. Joint Distribution of FP and FPC. Figures 6 and 7 show the joint distribution of the FP and FPC in Tar-1.28 and Cflow-1.4 software networks. As shown, most functions are located at the lower left corner of the graph; it means that the FP and FPC of these functions are relatively small; a small number of functions are located in the middle of the graph; it means that the FP and FPC of them are relatively big; only a very small number of functions are at the upper right corner of the graph; it signifies that the FP and FPC of these functions are big. Such functions are not only prone to fail but also have a strong fault propagation capability. If they fail, the fault severity to system disruption will be greater. In order to ensure the stability of software system, we should pay more attention to such functions and guarantee their correctness and robustness.

3.2.3. IC Model. In social network, the Independent Cascade Model (IC Model) is a propagation model of researching influence maximization problem. It is a probabilistic model. When a node u is activated, it tries to activate its inactivated neighbor node v with probability P_{uv}. This attempt is only done once, and these attempts are independent of each other. That is to say, the activation of u to v is not affected by other nodes.

In software network, when a failed node u is called, it propagates faults to the neighbor node that calls it with a probability P_{uv}. If the node u can affect a number of nodes, the severity of its failure is significant. This is very similar to the maximization of influence in social network. Therefore, we use IC model to verify that the proposed method MFS_AN does help to measure the fault severity of a node.

According to the mining results of the two kinds of software Tar and Cflow, the top-10 nodes and back-10 nodes are selected as source nodes, respectively. Through the IC model to simulate the number of nodes they can affect after being called, which in turn shows the severity of their failure. Due to the randomness of the IC model, we repeated the simulation 10 times for each version of each kind of software and then averaged the results, as shown in Figures 8 and 9.

As can be seen from Figures 1 and 2, if the functions fail, the number of nodes that can be affected by the top-10 function nodes is about 4 to 5 times that of the back-10 function nodes. This shows that if the top-10 function nodes fail, the fault severity of the software system will be 4 to 5 times that of the back-10 function nodes. Therefore, the ranking of node importance by the MFS_AN method does help to measure the fault severity of a node.

Table 4: The comparison of node rankings in Cflow-1.4.

Cflow-1.4)	Rank/value	
Node	MFS_AN	Degree
nexttoken	1/14.610	1/16
get_token	2/7.063	10/2
yylex	3/7.037	6/6
yyparse	4/2.514	6/6
parse_declaration	5/2.349	7/5
parse_dcl	6/2.241	5/7
expression	7/1.775	4/8
yyrestart	8/1.663	6/6
tree_output	9/1.489	2/14
linked_list_iterate	10/1.383	5/7

Table 5: The comparison of node rankings in Tar-1.28.

Tar-1.28	Rank/value	
Node	MFS_AN	Degree
flush_archive	1/9.208	9/4
dump_file	2/8.408	8/5
dump_file0	3/6.150	3/13
update_archive	4/4.830	1/14
find_next_block	5/4.304	7/6
create_archive	6/3.170	4/9
gnu_flush_write	7/3.103	11/2
open_archive	8/2.768	9/4
_gnu_flush_write	9/2.737	11/2
dump_regular_file	10/2.629	4/9

3.3. Comparison with Degree Method. The algorithm MFS_AN measures the node criticality from two aspects of the outdegree and indegree in the whole network structure. In directed graph, the degree centrality algorithm is a classical algorithm to measure the node criticality from outdegree and indegree. Thus, this paper compares the algorithm MFS_AN with degree centrality algorithm (denoted as Degree). Tables 4 and 5 show the comparative results of Cflow-1.4 and Tar-1.28.

The node ranking lists presented in Tables 4 and 5 are different, and the Degree method has the phenomenon that the same metric value of multiple nodes results in the same ranking. The reason is that the MFS_AN method starts from the fault accumulation and propagation characteristics of a function and focuses on out-degree neighbor nodes and In-Degree Neighbor nodes that have direct or indirect relationship with the current function node. It considers the global influence of the node. While the Degree method only pays attention to the direct out-degree and in-degree neighbor node of the current function node, it ignores the indirect influence of other nodes. However, in software network, the nodes are not isolated and they realize the complicated software function by calling each other. Therefore, compared with Degree method, MFS_AN method can identify the structure of a software more clearly and mine the key nodes of software network more accurately.

In summary, the algorithm MFS_AN proposed in this paper is correct and effective for the node criticality evaluation in software network. By using the algorithm MFS_AN to identify the key nodes in software network, it is helpful to reduce the software fault severity and improve the robustness and stability of software.

4. Conclusions and Future Work

In this paper, a novel algorithm MFS_AN is proposed to evaluate the criticality of nodes in software network by combining the two characteristics of fault probability and fault propagation capability together. And function nodes with larger fault probability and stronger fault propagation capability are regarded as the key nodes. With experiment, we analyse the FS distribution of the nodes in different software versions, realize the evolution law of software, and prove the algorithm MFS_AN can discover the key function nodes correctly and effectively in software network. On the other hand, the criticality of a function node is not directly related to degree, but it has a certain positive correlation. Furthermore, we could understand the software structure more easily and reduce the workload of testing and maintenance process to a maximum extent. In the future research, we will focus on how to divide the software module based on the key nodes.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Key R&D Program of China under Grant No. 2016YFB0800700, the National Natural Science Foundation of China under Grant Nos. 61472341, 61772449, 61572420, 61807028, and 61802332, the Natural Science Foundation of Hebei Province China under Grant No. F2016203330, and the Advanced Program of Postdoctoral Scientific Research under Grant No. B2017003005.

References

[1] F. T. Imam and T. R. Dean, "Modelling functional behavior of eventbased systems: a practical knowledge-based approach," *Procedia Computer Science*, vol. 96, pp. 617–626, 2016.

[2] G. McGraw, "Four software security findings," *The Computer Journal*, vol. 49, no. 1, pp. 84–87, 2016.

[3] S. K. Punia, A. Kumar, and A. Sharma, "Evaluation the quality of software design by call graph based metrics," *Global Journal of Computer Science and Technology*, vol. 14, no. 2, pp. 59–64, 2014.

[4] G. Huang, B. Zhang, R. Ren et al., "An algorithm to find critical execution paths of software based on complex network," *International Journal of Modern Physics C*, vol. 26, no. 09, Article ID 1550101, pp. 1550101-1–1550101-16, 2015.

[5] S. Lamzabi, S. Lazfi, H. Ez-Zahraouy et al., "Pair-dependent rejection rate and its impact on traffic flow in a scale-free network," *International Journal of Modern Physics C*, vol. 25, no. 07, pp. 1450019-1–1450019-10, 2014.

[6] Y.-T. Ma, K.-Q. He, and B. Li, "Empirical study on the characteristics of complex networks in networked software," *Journal of Software*, vol. 22, no. 3, pp. 381–407, 2011.

[7] C. Y. Chong and S. P. Lee, "Analyzing maintainability and reliability of object-oriented software using weighted complex network," *Journal of Systems & Software*, vol. 110, no. C, pp. 28–53, 2015.

[8] S. Valverde and R. V. Sole, "Hierarehieal small worlds in software architecture," *Dynamics of Continuous Discrete & Impulsive Systems*, vol. 14, p. 1, 2003.

[9] C. R. Myers, "Software systems as complex networks: structure, function, and evolvability of software collaboration graphs," *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, vol. 68, no. 4, pp. 046116-1–046116-15, 2003.

[10] W. Pan, B. Li, B. Jiang et al., "Recode: software package refactoring via community detection in bipartite software networks," *Advances in Complex Systems. A Multidisciplinary Journal*, vol. 17, no. 7n08, Article ID 1450006, 2014.

[11] F. Thung, D. Lo, M. H. Osman et al., "Condensing class diagrams by analyzing design and network metrics using optimistic classification," in *Proceedings of the 22nd International Conference on Program Comprehension, ICPC '14*, pp. 110–121, ACM, Hyderabad, India, June 2014.

[12] N. M. Mohammed, M. Niazi, M. Alshayeb et al., "Exploring software security approaches in software development lifecycle: a systematic mapping study," *Computer Standards & Interfaces*, vol. 50, pp. 107–115, 2016.

[13] W.-F. Pan, B. Li, Y.-T. Ma, Y.-Y. Qin, and X.-Y. Zhou, "Measuring structural quality of object-oriented softwares via bug propagation analysis on weighted software networks," *Journal of Computer Science and Technology*, vol. 25, no. 6, pp. 1202–1213, 2010.

[14] J. Liu, "The structure and the volatility of the software," in *Blue*, H. KeQing, Ed., pp. 156–164, Software Network (Beijing:Science Press), 2008.

[15] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1978.

[16] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts, "Network robustness and fragility: percolation on random graphs," *Physical Review Letters*, vol. 85, no. 25, pp. 5468–5471, 2000.

[17] M. Kitsak, L. K. Gallos, S. Havlin et al., "Identification of influential spreaders in complex networks," *Nature Physics*, vol. 6, no. 11, pp. 888–893, 2010.

[18] I. Turnu, M. Marchesi, and R. Tonelli, "Entropy of the degree distribution and object-oriented software quality," in *Proceedings of the 2012 3rd International Workshop on Emerging Trends in Software Metrics, WETSoM 2012*, pp. 77–82, Switzerland, June 2012.

[19] B.-Y. Wang and J.-H. Lü, "Software networks nodes impact analysis of complex software systems," *Journal of Software*, vol. 24, no. 12, pp. 2814–2829, 2013.

[20] P. Bhattacharya, M. Iliofotou, I. Neamtiu et al., "Graph-based analysis and prediction for software evolution," in *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 419–429, IEEE, Zürich, Switzerland, June 2012.

[21] G. Huang, P. Zhang, Y. Li, and J. Ren, "Mining the important nodes of software based on complex networks," *ICIC Express Letters*, vol. 9, no. 12, pp. 3263–3268, 2015.

[22] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[23] J. Ren, H. Wu, T. Yin, L. Bai, and B. Zhang, "A novel approach for mining important nodes in directed-weighted complex software network," *Journal of Computational Information Systems*, vol. 11, no. 8, pp. 3059–3071, 2015.

[24] J. Ren, H. Wu, R. Gao et al., "Identifying important nodes in complex software network based onripple effects," *Icic Express Letters Part B Applications An International Journal of Research and Surveys*, p. 7, 2016.

[25] J. Zhang, "The calculating formulae, and experimental methods in error propagation analysis," *IEEE Transactions on Reliability*, vol. 55, no. 2, pp. 169–181, 2006.

[26] M. Shafique, S. Rehman, P. V. Aceituno et al., "Exploiting program-level masking and error propagation for constrained reliability optimization," in *Proceedings of the 50th Annual Design Automation Conference (DAC '13)*, pp. 1–9, Austin, Tex, USA, June 2013.

[27] M. Hiller, A. Jhumka, and N. Suri, "PROPANE: an environment for examining the propagation of errors in software," *Acm Sigsoft Software Engineering Notes*, vol. 27, no. 4, pp. 81–85, 2003.

Hindawi
Security and Communication Networks
Volume 2019, Article ID 8391425, 13 pages
https://doi.org/10.1155/2019/8391425

Research Article

A Buffer Overflow Prediction Approach Based on Software Metrics and Machine Learning

Jiadong Ren,[1,2] Zhangqi Zheng,[1,2] Qian Liu,[1,2] Zhiyao Wei,[1,2] and Huaizhi Yan[3]

[1]*School of Information Science and Engineering, Yanshan University, Qinhuangdao, Hebei, China*
[2]*The Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao City 066004, China*
[3]*Beijing Key Laboratory of Software Security Engineering Technique, Beijing Institute of Technology, South Zhongguancun Street, Haidian District, Beijing 100081, China*

Correspondence should be addressed to Zhangqi Zheng; 451499304@qq.com

Received 11 October 2018; Revised 26 December 2018; Accepted 10 February 2019; Published 3 March 2019

Guest Editor: Jiageng Chen

Buffer overflow vulnerability is the most common and serious type of vulnerability in software today, as network security issues have become increasingly critical. To alleviate the security threat, many vulnerability mining methods based on static and dynamic analysis have been developed. However, the current analysis methods have problems regarding high computational time, low test efficiency, low accuracy, and low versatility. This paper proposed a software buffer overflow vulnerability prediction method by using software metrics and a *decision tree* algorithm. First, the software metrics were extracted from the software source code, and data from the dynamic data stream at the functional level was extracted by a data mining method. Second, a model based on a *decision tree* algorithm was constructed to measure multiple types of buffer overflow vulnerabilities at the functional level. Finally, the experimental results showed that our method ran in less time than *SVM, Bayes, adaboost, and random forest* algorithms and achieved 82.53% and 87.51% accuracy in two different data sets. The method presented in this paper achieved the effect of accurately predicting software buffer overflow vulnerabilities in C/C++ and Java programs.

1. Introduction

With the popularity and rapid development of computer network technology, software security is facing a growing number of threats. Since the first buffer overflow attack occurred in 1988, the buffer overflow vulnerability [1] has been the most common and serious software vulnerability, posing a great danger to the security of software systems. According to the distribution of vulnerability types in the June 2017 Security Vulnerability Report of the National Information Security Vulnerability Database (*CNNVD*), the buffer error category has the largest proportion of vulnerabilities, at approximately 14.08%. Due to the various forms of buffer overflow vulnerabilities, it is challenging to accurately predict buffer overflows. Due to the large amount of software source code and the complex logic level of invocation, these kinds of vulnerabilities are not easily discovered in a timely manner.

Software vulnerability analysis usually requires source code analysis and binary analysis. Although the source code vulnerability analysis can comprehensively consider the execution path according to the rich and complete semantic information in the program source code, it cannot fully reflect the dynamics of the software [2]. Although binary code vulnerability analysis has been studied comprehensively through static or dynamic and manual or automated methods, there are problems such as low coverage and path explosion. Recently, machine learning algorithms have been widely used in software vulnerability prediction. In vulnerability prediction, machine learning techniques such as *logistic regression*, *naive Bayes methods*, *C4.5 decision tree algorithms*, *random forest classifiers,* and *SVMs* [3, 4] are commonly used for prediction. Although it is common to use a machine learning algorithm to predict software vulnerabilities, the research based on software metrics is still immature. For instance, in the area of predicting software vulnerability models based on software metrics, most approaches consist of a static analyses based on software source code. It does not consider the characteristics of the dynamic data flow during software

operation. It is not possible to more accurately reflect the type of vulnerability, its severity, and the distribution of the module.

This paper proposes a method based on software metrics for buffer overflow vulnerability prediction, called the software vulnerability location (*BOVP*) method. Compared with traditional prediction methods, such as *support vector machines* (*SVMs*), *Bayesian methods, adaboost,* and *random forest* (*RF*) algorithms, this new research method is applicable to software on different platforms (C/C++, JAVA) and can also accurately predict software buffer overflow vulnerabilities. The main contents of this paper are as follows. (1) Software metrics were extracted according to the static analysis of software source code, and a data mining method was used to extract data from the dynamic data stream at the functional level. (2) A *decision tree* algorithm was established to measure multiple types of buffer overflow vulnerability models based on the functional level. The algorithm not only affects the accuracy of the experiment but also reduces the measurement dimension and reduces the experimental overhead. (3) The analyses led to the research idea of considering multiple vulnerability types and different functional classifications for the software buffer overflow vulnerability analysis.

The remainder of this paper is organized as follows: Section 2 introduces the related work of the paper. Section 3 introduces the description of buffer overflow vulnerability overview and source code function classification. Section 4 introduces the overview of the *BOVP* method. Section 5 presents data sources, cross-validation, experimental procedures and results. Section 6 describes the advantages of the *BOVP* approach. Finally, the conclusion and future research direction are presented in Section 7.

2. Related Work

At present, the academic community proposes dual solutions to the phenomenon of software buffer overflow vulnerability based on both detection and protection. For example, to avoid buffer overflow, people use the polymorphic canary to detect an overflow of the stack buffer based on computer hardware research [5, 6]. J Duraes et al. [7] proposed an automatic identification method for buffer overflow vulnerability of executable software without source code. S Rawat et al. [8] proposed an evolutionary calculation method for searching for buffer overflow vulnerability; this method combined genetic algorithms and evolutionary strategies to generate string-based input. It is a lightweight intelligent fuzzy tool used to detect buffer overflow vulnerabilities in C code. IA Dudina et al. [9] used static symbolic execution to detect buffer overflow vulnerabilities, described a prototype of an in-process heap buffer overflow detector, and provided an example of the vulnerability discovered by the detector. In terms of protection from buffer overflow vulnerabilities, S Nashimoto et al. [10] proposed injecting buffer overflow attacks and verified the countermeasures for multiple fault injections. The results showed that their attacks could cover the return address stored in the stack and call any malicious function. J Rao et al. [11] proposed a protection technology called a BF Window for performance and resource-sensitive embedded systems. It verified each memory write by speculatively checking the consistency of the data attributes within the extended buffer window. Through experiments, the proposed mechanism could effectively prevent sequential memory writes across the buffer boundary. Although these methods have effectively detected buffer overflow vulnerabilities to a certain extent, the traditional research methods of software buffer overflow vulnerabilities have the problems of large time overhead, path explosion and lack of dynamic software analysis. Therefore, research on using machine learning methods to predict software buffer overflow vulnerabilities has become increasingly widespread.

At present, research on using machine learning algorithms to predict various types of software vulnerabilities [12] is becoming widespread, and the machine learning methods can improve vulnerability coverage and reduce running time in software vulnerability analysis and discovery processes. However, there are problems in that they that cannot more accurately reflect the type of vulnerability, the severity of the vulnerability or the distribution of the module. For these problems, a method based on the "software metrics" to predict various types of software vulnerabilities was proposed [13]. Early software metrics focused on structured program design. By measuring the complexity of the software architecture, researchers described some basic performance metrics of software systems such as functionality, reliability, portability, and maintainability. Software metrics can be divided into three typical representative categories based on information flow, attributes and behaviour. In recent years, software metrics have been studied extensively and have become a vibrant research topic in software engineering.

James Walden et al. [14] compared models based on two different types of features: software metrics and term frequencies (text mining features). They experimented with Drupal, PHPMyAdmin, and Moodle software. The experiments evaluated the performance of the model through hierarchical and cross-validation. The results showed that both of the models for text mining and software metrics had high recall rates. Based on this research, in 2017, Jeffrey Stuckman et al. [15] explored the role of dimension reduction through a series of cross-validation and project prediction experiments. In the case of software metrics, the dimension reduction technique based on a confirmatory factor analysis produced the best F-measure value when performing project prediction. Choudhary G R et al. [16] studied the change metrics in combination with code metrics in software fault prediction to improve the performance of the fault prediction model. Different versions of the Eclipse project and extracted change metrics from GitHub were used for experimental research. In addition to the existing change metrics, several new change metrics were defined and collected from the Eclipse project repository. The experimental results showed that the variation metrics had a positive impact on the performance of the fault prediction model. Thus, a high-performance model could be established through multiple variation metrics. Sanaz Rahimi et al. [17] proposed a vulnerability discovery model (VDMS), which uses the compiler's code base for static analysis, extracts code characteristics such as complexity (circle complexity) and uses code attributes

Table 1: C/C + + language experimental data set specific data.

	GOOD				BAD			Total
	Good	Good Sink	Good Source	Good Auxiliary	Bad	Bad Sink	Bad Source	
Stack-based buffer overflow	5810	2748	312	7149	4716	2052	288	23075
Heap-based buffer overflow	7122	3076	968	9064	6733	2448	628	30039
Buffer overflow	2380	1180	208	3192	2472	1024	144	10600
Integer overflow	4608	4284	576	10872	4716	2052	288	27396
Integer underflow	3348	3234	408	8100	3414	1548	204	20256
Total	23268	14522	2472	38377	22051	9124	1552	111366

as its parameters to predict vulnerabilities. By performing a static analysis on the source code of real software, it was verified that the code attributes have a high level of influence on the accuracy of vulnerability discovery.

In recent years, many studies have predicted software buffer overflow vulnerability by using a machine learning algorithm. With the maturity of software measurement technology, most scholars have adopted software attribute measurement based on machine learning algorithms to predict vulnerabilities. Most of the software metrics for predicting software vulnerability are based on the static analysis of the software source code, but there are dynamic behaviours such as function calls, memory read and write, and memory allocation during the execution process. Therefore, if the software metrics are to accurately predict software vulnerabilities that are persistent and concealed, the dynamic behaviour of the software needs to be considered.

3. Problem Description

Buffer overflow is the most dangerous vulnerability of software. If the buffer overflow vulnerability can be effectively eliminated, the security threat to the software will be reduced. This paper analysed the data flow based on the source code function level and combined the machine learning algorithm to study the software metrics for real software. Metrics need to be selected in software metrics, such as the number of lines containing source code, the average base complexity, all nested functions or methods, and the average number of lines of source code that contain all nested functions or methods (as shown in Table 1).

3.1. Buffer Overflow Vulnerability Overview. A buffer is a contiguous area of memory within a computer that can hold multiple instances of the same data type. The reason for the buffer overflow is that the computer exceeds the capacity of the buffer itself when it fills the buffer with data. The overflow data are overlaid on the legal data (data, pointer of the next instruction, return address of the function, etc.). Buffer overflows can be divided into stack-based buffer overflows, heap-based buffer overflows, and integer overflows. Integer overflows are classified into three categories based on underflow and overflow of unsigned integers, symbolic problems, and truncation problems. We use the Intel processor (register EBP) as an example to introduce stack-based buffer overflows, heap-based buffer overflows, and integer overflows.

Stack-based buffer overflows: in computer science, a stack is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the most recently added element that was not yet removed. A stack is a container for objects that are inserted and removed according to the LIFO principle. The stack frame is the collection of all data in the stack associated with one subprogram call. The stack frame generally includes the return address, the argument variables passed on the stack, local variables, and the saved copies of any registers modified by the subprogram that need to be restored. The EBP is the base pointer for the current stack frame. The ESP is the current stack pointer. Each time the function is called, a new stack frame is generated and stored in a certain space of the stack. The EBP always points to the top of the stack frame (high address), while the ESP points to the next available byte in the stack. The EBP does not change during function execution, and the ESP varies with function execution. A stack-based buffer overflow occurs when a program writes data with a data length that exceeds the space allocated by the stack to the buffer to the memory address in the stack.

Heap-based buffer overflows: The heap is memory that is dynamically allocated while the program is running. The user generally requests memory through malloc, new, etc. and operates the allocated memory through the returned starting address pointer. After using it, it should be released by free, delete, etc. as part of the memory; otherwise it will cause a memory leak. The heap blocks in the same heap are usually contiguous in memory. If data is written to an allocated heap block, the data overflow exceeds the size of the heap block, causing the data overflow to cover the neighbours behind the heap block.

Integer overflows: Similar to other types of overflows, integer overflows cause data to be placed in a storage space smaller than itself. The underflow of unsigned integers is caused by the fact that unsigned integers cannot recognize negative numbers. The symbol problem is caused by the comparison between signed integers, the operation of signed integers, and the comparison of unsigned integers and signed integers. The truncation problem mainly occurs when a high-order integer (for example, 32 bits) is copied to a low-order integer (for example 16 bits).

Buffer overflow attacks consist of three parts: code embedding, overflow attack, and system hijacking. There are four types of buffer overflow attacks: destroy activity records, destroy heap data, change function pointers, and overflow fixed buffers. Buffer overflow is a very common and very dangerous vulnerability that is widespread in various operating systems and applications. The use of buffer overflow vulnerabilities can enable hackers to gain control of a machine or even the highest privileges. The use of buffer overflow attacks can lead to program failure, system shutdown, restarting, and so on. The buffer overflows we studied in this article include stack-based buffer overflows, heap-based buffer overflows, buffer overflows, integer overflows, and integer underflows.

3.2. Source Code Function Classification. Most software metrics that were previously used to predict software vulnerabilities use metrics to predict whether a vulnerability is included. During the execution of the software, there is a lack of analysis of the calling relationships between functions. Among the characteristics of software are functionality, reliability, usability, efficiency, maintainability and portability. Thus, the software implementation process is dynamic. In the vulnerability prediction of software, the calling relationship of the intrinsic functions in the software execution process should be fully considered, especially the data flow changes during the software call. Therefore, this article describes data flow analysis to distinguish features that contain vulnerabilities.

In general, software source code data flow analysis is based on understanding the logic of the code, tracking the flow of data from the beginning of the analysed code to the end. There are three difficulties in the analysis process. (1) The change in data should accumulate with the increase in the path length. (2) Calculation of the value of the branch condition based on the latest data analysis results since the branch condition should accumulate as the path grows, and each additional branch node on the path must be in the path. (3) The number of paths usually grows at a geometric progression as the code size increases. A more efficient approach is to decompose the global data stream analysis into partial data stream analysis.

This paper analyses the functions called in the test case data and uses the partial data flow analysis method to distinguish the functions into different "source" and "accept" functions. This research performs a functional level analysis on the program source code. According to the behaviour of the data flow of each function in the actual execution of the program, this paper uses a data mining method to distinguish the function categories from the data in the data set. First, there are two types (good and bad) depending on whether or not there are vulnerabilities as a whole. Second, this paper analyses the functions using data flow analysis. Functions that do not contain vulnerabilities are classified into four categories: Good - the only code in a good function is a call to each of the secondary good functions. This function does not contain any nonflawed constructs; Good Sink and Good Source - cases that contain data flows using "source" and "sink" functions that are called from each other from the primary good function. Each source or sink function is specific to exactly one secondary good function; Good Auxiliary - where a Good Source is passing safe data to a potentially Bad Sink and where a Bad Source is passing unsafe or potentially unsafe data to a Good Sink. The categories containing vulnerabilities are divided into three groups: Bad: this is a function that contains a flawed construct; Bad Sink and Bad Source: these are cases that contain data flows using "source" and "sink" functions that are called from each other or from a primary bad function. Each source or sink function is specific to either bad function for the test case.

According to whether or not the vulnerability is included in one of the 8 categories, this paper defines the set of categories as $X = \{x_1, x_2 \ldots, x_8\}$, where x_1, x_2,..., x_8 are the number of functions that do not contain vulnerabilities (Good), sinks that do not contain vulnerabilities (Good Sink), source functions that do not contain vulnerabilities (Good Source), auxiliary functions that do not contain vulnerabilities (Good Auxiliary), a main function containing a bug (Bad), a sink that contains the vulnerability's accept function (Bad Sink), a source function containing a vulnerability (Bad Source), and a function that is not yet explicitly defined (Other). In this study, only the first seven categories were studied.

4. Method

4.1. BOVP Method. This paper first extracts the source code of the executable program and then establishes a dynamic data flow analysis model at the software function level. Finally, a vulnerability prediction method based on software metrics is implemented. The process of establishing the *BOVP* {M, A, S} model in this method is shown in Figure 1.

(1) For the source code of the target software, this paper applied the functional level static analysis, and the software attribute measurement method was used to extract all software metric data (M).

(2) According to the characteristics of the software buffer overflow vulnerability, this paper established a metric "A" of the software level buffer overflow vulnerability based on the function level, A = {a1, a2, ..., an}.

(3) For the characteristics of the data flow where the functions are calling each other, the data mining method was used to extract the data regarding the function class, and items in the "Other" function category were removed.

(4) This paper applied the mutual information value for feature selection and incorporated the metric A to measure the impurity of the data partition D for the split criterion. The cost complexity pruning algorithm was applied as the pruning method to determine the buffer overflow vulnerability classification algorithm S (strategy).

4.2. BOVP Prediction

4.2.1. Model Overview. The *CART* (classification and regression tree) model was proposed by Breiman et al. [18] in 1984 and has become a widely used *decision tree* learning method. The *CART* method assumes that the *decision tree* is a binary tree. The input feature space is divided into a finite number of

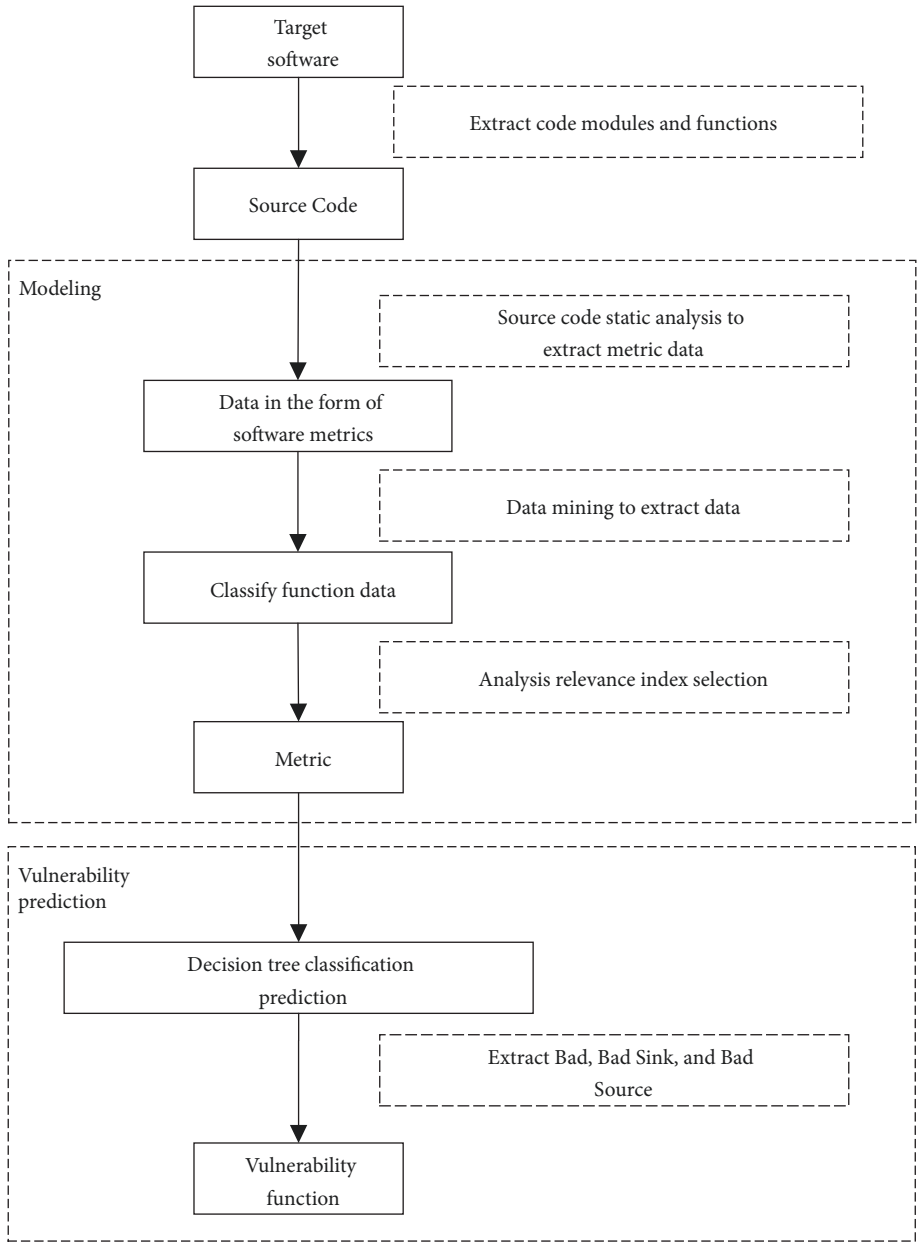


Figure 1: The framework for the *BOVP* method.

cells, and the predicted values are determined by these cells, for instance, for predicting the output values to map to a given condition.

The *CART* model in this article consists of the following three steps:

(1) *CART* generation: generating a *decision tree* based on the training data set.

(2) *CART* pruning: pruning the *decision tree* according to certain constraints, such as the maximum depth of the tree, the minimum number of samples of the leaf nodes, the purity of the nodes, etc.

(3) Optimization of the *decision tree*: generating different CARTs for multiple parameter combinations (maximum depth of the tree (max_depth), minimum sample number of the leaf node (min_samples_leaf), node impurity (min_impurity_split)) and choosing the model with the best generalization ability.

We first chose to use a supervised learning algorithm. The next consideration was the data problem, for example, whether the eigenvalue is a discrete variable or a continuous variable, whether there is a missing value in the eigenvalue vector, and whether there is an abnormal value in the data. Therefore, after analysis, the *CART* tree classification algorithm was selected. *CART* has the following advantages over other classical classification and regression algorithms: (1) less data preparation: no need to standardize data; (2) the ability to process continuous and discrete data; (3) the ability to handle multiclassification problems; (4) the prediction

process of the *CART* model which can be easily explained by Boolean logic and compared to other models such as *Naive Bayes* and *SVMs*.

4.2.2. Training. In the classification prediction part of the *BOVP* method, first, the feature selection operation was performed on the attribute set A. Second, a specific classification algorithm was selected according to the data characteristics. This research adopts the *decision tree* algorithm [19] to establish the classification model based on the characteristics of the integrated software and the characteristics of the classification algorithm.

First, the classification *decision tree* model is a tree structure that classifies instances based on features. The *decision tree* consists of nodes and directed edges. There are two types of nodes: internal nodes and leaf nodes. The internal nodes represent n attributes, including the number of lines of source code, the code path that can be executed, and the circle complexity in A. The leaf nodes represent the type of $X = \{x_1., x_2, ..., x_7\}$. Second, when the *decision tree* is classified, one of the features of an instance is tested from the root node, and the instance is assigned to its child node according to the test result. Each child node corresponds to a value of the feature, so it recursively moves down until the leaf node is reached. Finally, the instance is assigned to the class of the leaf node. The *decision tree* can be converted to a set of if-then rules, or it can be considered a conditional probability distribution defined in the feature and the class space. Compared with the naive Bayes classification method, the *decision tree* has the advantage that the construction process does not require any domain specific knowledge or parameter settings. Therefore, the *decision tree* is more suitable for exploratory knowledge discovery in practical applications. The construction of the *decision tree* algorithm [19] is divided into three parts: feature selection, *decision tree* generation, and *decision tree* pruning, as follows:

(1) Feature selection: the features in this model are a_n ($n <= 22$), an attribute that includes the number of lines in the source code, the code path that can be executed, and the circle complexity. The feature selection is applied to select the feature that has the ability to classify the training data in a way that can improve the efficiency of *decision tree* learning. The feature selection process uses information gain and the information gain ratio or *Gini* indexes.

(2) Generation of *decision tree*: the *Gini* index is used as the criterion of feature selection to measure the impurity of data partition D such that the local optimal feature is continuously selected and the training set is divided into subsets that can be feasible.

(3) *Decision tree* pruning: the cost complexity pruning algorithm is used to prune the tree; that is, the tree complexity is regarded as a function of the number of leaf nodes and the error rate of the tree, specifically from the bottom of the tree. For each internal node N, the cost complexity of the subtree of N and the cost complexity of the subtree of N after the pruning of the subtree are calculated, and the subtree is clipped; otherwise, the subtree is retained.

Combining the attribute set $A = \{a_1, a_2, ..., a_{22}\}$, we can set the training data partition to D, and construct the *decision tree* model through three steps of feature selection, generation *decision tree* and pruning. The basic strategy of the algorithm is to call the algorithm with three parameters D, *attribute_list* and *Attribute_selection_method*. D is a data partition which is a collection of training tuples and corresponding categories of labels. *attribute_list* is a list describing the tuple attributes. *Attribute_selection_method* specifies the heuristic process that is used to select the "best" attributes by class in tuple.

In Algorithm 1, Lines (1) to (6) specify the *Gini* minimization criterion used to obtain the feature attributes and the partition points of the split selection. When the feature selection is started, a tree is constructed step by step. Lines (7) to (8) describe the pruning process of the tree according to three indicators (γ, α, β). Once the stopping condition is satisfied, data set D is split into two subtrees, D1 and D2, which are expressed in Lines (9) to (10). Lines (11) to (12) state that if the sample number of the nodes is less than β, the node will be dropped. Lines (13) to (17) show that if the node does not satisfy all of the conditions, it will be converted into a leaf node whose output class is the highest of the classes on the node. Finally, the process of prediction is conducted by using the TreeModel in Line (19). This method is highly complex, especially when searching for the segmentation point, as it is necessary to traverse all the possible values of all the current features. If there are F features each having N values, and the generated *decision tree* has F or S internal nodes, then the time complexity of the algorithm is O(F∗N∗S).

The *decision tree* in the algorithm uses the *Gini* coefficient as the splitting point of the selection feature. That is, the training data set D is divided into two parts D_1 and D_2 according to whether feature A takes a certain value a. Then, under the condition of feature A, the *Gini* index of set D is defined as

$$Gini(D, A) = \frac{|D_1|}{D} Gini(D_1) + \frac{|D_2|}{D} Gini(D_2) \quad (1)$$

The *Gini* index *Gini*(D) represents the uncertainty of set D and the *Gini* index *Gini*(D, A) represents the uncertainty of set D after A=a partitioning. The larger the *Gini* index is, the greater the uncertainty of the sample is. Compared to other classical classification and regression algorithms, *decision trees* have the advantage of generating easy-to-understand rules, requiring a relatively small number of computations. *Decision trees* are capable of processing continuous categorical fields and have the ability to clearly display more important fields. Therefore, using the existing data set to train the *decision tree* model, the model can be trained to predict whether the program has any of the 7 buffer vulnerabilities by using the metrics, such as the number of rows containing the source code, the number of times called, the code path that can be executed, and the loop complexity.

The specific method of generating the subtree sequences T0, T1,..., Tn in pruning is by pruning the branch in Ti that has the smallest increase in error in the training data set from T0 to obtain Ti+1. For example, when a tree T is pruned at node t, the increase in error is R(t)−R(Tt), where R(t) is the error of node t after the subtree of node t is pruned. R(Tt) is the error of the subtree Tt when the subtree of node t is not pruned. However, the number of leaves in T is

input: D={(X_1, Y_1),(X_2,y_2),...,(X_n, y_n)}, X = {$X_1, X_2, X_3, \ldots, X_n$} represents a feature property set, Y = {$y_1, y_2, y_3, \ldots, y_n$} represents the predicted attribute set.
output: *CART* model sets
(1) for x_i in X
(2) for T_j in x_i
(3) search min(Gini);
(4) end for
(5) end for
(6) Build Trees TreeModel from Tj and xi;
(7) if H(D) ≤ γ && the current depth < α && |D| ≥ β // |D| sample number of D
(8) Divide D to D1 and D2;
(9) DecisionTreeClassifier (D1, α, γ, β);
(10) DecisionTreeClassifier (D2, α, γ, β);
(11) else if |D| < β
(12) drop D
(13) else
(14) D ⟶ leaf_Node; // convert to leaf node
(15) predictive class = the most number of classes; // average of samples
(16) break;
(17) end else
(18) return TreeModel
(19) Prediction on TMS.

Algorithm 1: *BOVP* based on *Gini* indexes.

reduced by L(Tt)−1 after pruning, so the complexity of T is reduced. Therefore, considering the complexity factor of the tree, the error increase rate after the tree branches are pruned is determined by

$$\alpha = \frac{R(t) - R(T_t)}{L(T_t) - 1} \tag{2}$$

where R(t) represents the error of node t after the subtree of node t is pruned, R(t)=r(t)∗p(t), r(t) is the error rate of node t, and p(t) is the ratio of the number of samples on node t to the number of samples in the training set. R(Tt) represents the error of the subtree Tt when the subtree of node t is not pruned, that is, the sum of the errors of all the leaf nodes on subtree Tt. Ti+1 is the pruning tree with the smallest alpha value in Ti.

4.2.3. Prediction. The input part of the prediction section is the buffer overflow vulnerability data set D={(x1, y1), (x2, y2)...(xn, yn)}. Vulnerability data include stack buffer overflow, heap buffer overflow, buffer underwriting ("buffer overflow"), integer overflow, and integer underflow.

When constructing the *decision tree* in the prediction process, the feature-selected metrics A = {$a_1, a_2, \ldots, a_{16}$} were used as internal nodes to represent an attribute test, and the *Gini* index was used to select the partition attribute. Each branch represents an output of the test, and the leaf nodes correspond to the function class X = {$x_1, x_2, \ldots, x_7$}. When constructing a *decision tree*, this paper uses attribute selection metrics to select the attributes that best divide the tuple into different classes.

The prediction result output is based on the class predicted by the *decision tree* algorithm belonging to the X = {$x_1, x_2, \ldots, x_7$} category. Further analysis can be performed on the function containing the vulnerability based on the prediction results. The prediction part used the accuracy rate, recall rate and F_1 evaluation index to measure the effect of the *decision tree* algorithm. Accuracy is used to measure the ratio of the number of correctly classified samples to the total number of samples in the test data set. It reflects the ability of the *decision tree* classifier to examine the entire sample. The recall rate is used to measure the proportion of all that should be retrieved correctly, which reflects the proportion of positive data that is correctly determined by the *decision tree* algorithm to the total positive data. The F1 value is used to measure the accuracy of the recall and the recall average to measure the ideal degree of *decision tree* algorithm classification.

5. Experiment

In this paper, the relationship of function nesting, iteration and looping in the process of software execution, and the dynamic behaviour and operations of the functions were considered comprehensively. The vulnerability classification and prediction were carried out according to the unified pattern of software metrics. This paper carried out the following steps for the program: (1) This article extracted code that contained only one type of defect based on the buffer overflow vulnerability code fragment and that did not contain code of other unrelated defect types. (2) Then, this paper statically analysed the size, complexity, and coupling of the source code for each code segment that could be generated. At the same time, software was used to extract the metrics from the program, and 58 functional level metrics were extracted. (3) Due to the diversity of the types of buffer

overflow vulnerabilities involved in the program, there is data imbalance in the extracted static code indicators. Therefore, it was necessary to perform data cleaning on the initially extracted data to reexamine and verify the data and delete duplicate information. Twenty-two indicators were selected for research. (4) The method of feature selection based on mutual information values was used to select the software metric attribute A. (5) According to the 4:1 ratio between the training set and test set, the *Gini* coefficient was selected as the splitting point for feature selection, and the *CART decision tree* was constructed by the K-times crossover test.

5.1. Data Set Introduction. Using the Juliet data set of the NIST library (https://samate.nist.gov/SRD/view.php), this paper selects 163,737 new types of buffer overflow vulnerabilities in an actual program extraction. The information contained in the data set is made up of the National Security Agency (NSA) Reliable Software (CAS). All the programs selected in this article can be compiled and run on Microsoft Windows using Microsoft Visual Studio. This article extracts software metrics based on the source code of the Juliet data set cross-project test case. A buffer overflow occurs when a write stream flows through the buffer and the function return address is modified during program execution. In this study, source code written in C and Java were used for experiments.

The C/C++ language experimental data set had 111,366 stack-based buffer overflows, heap-based buffer overflows, buffer overflows, integer overflows, and integer underflows. A total of 52,371 buffer overflows were collected from the Java language experimental data set. The summaries of the data sets are shown in Tables 1 and 2.

5.2. Metrics Indexes. This paper extracted a large number of real software attributes through analysis, from software source code. Twenty-two metrics are listed in Table 3. Because some attributes are irrelevant, feature selection based on mutual information is used to eliminate irrelevant metrics. Finally, the accuracy recall and accuracy indicators are used to measure the predicted results.

Software metric indexes provide a quantitative standard to evaluate code quality. It not only visually reflects the quality of software products but also provides a numerical basis for software performance evaluation, productivity model building, and cost performance estimation. According to the development of software from structure, modularization to object-oriented design, metrics can be divided into traditional software metrics and object-oriented software metrics. According to traditional software measurement indexes, the attribute set of the software is first defined as $A = \{a_1, a_2, \ldots, a_n\}$. First, *Understand*, a code analysis tool, was used to extract the software data based on 58 metrics. Second, because the software source code is written in different languages and the coding conventions are different, it is necessary to perform data cleaning on the initially extracted data to reexamine and verify the data, delete duplicate information and correct errors. To ensure data consistency, we need to filter data that does not meet certain requirements. For example, the index of the extracted data contains a large number of zero values and attribute values for the same numerical metrics. Finally, 22 indicators remained for study, so n = 22. The specific indicators are shown in Table 3.

Feature selection based on these 22 metrics can not only improve the efficiency of classifier training and application by reducing the effective vocabulary space but also remove noise features and improve classification accuracy. In this paper, the mutual information method is adopted to select features based on the correlation of attributes.

Mutual information indicates whether two variables X and Y have a relationship [20, 21], defined as follows: let the joint distribution of two random variables (X, Y) be p(x, y), the marginal distributions are p(x) and p(y), the mutual information I(X;Y) is the relative entropy of the joint distribution p(x, y) and the product distribution p(x)p(y), as shown in Equation (3) .

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)\,p(y)} \tag{3}$$

The mutual information of feature items and categories reflects the degree of correlation between feature items and categories. When taking mutual information as the evaluation value for feature extraction, the largest number of features of mutual information will be selected.

5.3. Feature Selection. Due to the limited number of samples, the design of a classifier with a large number of features is computationally expensive, and the classification performance is poor. Therefore, feature selection was adopted for the metrics set A [21]; that is, a sample of the high-dimensional space was converted into a low-dimensional space by means of mapping or transformation and dimensionality reduction was achieved. Then, redundant and irrelevant features were deleted by feature selection to achieve further dimensionality reduction. In this experiment, the mutual information value was used to measure the correlation for feature selection. If there was no correlation, the mutual information value was zero. Conversely, if the mutual information value is larger, the correlation is greater. When the feature was selected, the mutual information value was calculated for both the C/C++ and Java data sets. The results are shown in Table 4.

The mutual information values of the two data sets are synthesized to select a_n, n<=22, and finally, 16 metrics are obtained after mutual information calculation. Define the set of attributes as $A = \{a_1, a_2, \ldots, a_{16}\}$. Among them, $a_1, a_2, \ldots, a_{16}$ is *CountInput, CountLine, CountLineCode, CountLineCode Decl, CountLineCode Exe, CountLine Comment, CountOutput, CountPath, CountSemicolon, CountStmt, CountStmtDecl, CountStmtExe, Cyclomatic Modified, Cyclomatic Strict, Ratio CommentCode*.

Although the global results before and after the feature selection seem similar, we achieve the same effect with fewer features by applying feature selection. The obtained data features and the 16 indicators adopted after the feature selection could achieve high-efficiency cross-project software metrics, thereby reducing the number of indicators, dimensions and amount of overfitting. The process also

Table 2: Java language experimental data set specific data.

	GOOD				BAD			Total
	Good	Good Sink	Good Source	Good Auxiliary	Bad	Bad Sink	Bad Source	
Buffer overflow	8073	9522	828	21114	7866	4554	414	52371

Table 3: Metrics.

	Name	Description
1	CountInput	Number of calls. Calls by the same method are counted only once, and calls by fields are not counted.
2	CountLine	The number of lines of code.
3	CountLineCode	The number of lines containing the code.
4	CountLineCodeDecl	The number of lines of the name class, the method name line is also recorded in this number.
5	CountLineCodeExe	The number of lines of pure executing class code.
6	CountLineComment	Annotation class code line number.
7	CountOutput	The number of calls to other methods, multiple calls to the same method are counted as one call. Return statement counts a call.
8	CountPath	Code paths that can be executed are related to cyclomatic complexity.
9	CountPathLog	log 10, truncated to an integer value, of the metric CountPath.
10	CountSemicolon	The number of semicolons.
11	CountStmt	The number of statements, even if multiple sentences are written on one line, is counted multiple times.
12	CountStmtDecl	Defines the number of class statements, including the method declaration line.
13	CountStmtExe	The number of class statements executed.
14	Cyclomatic	Circle complexity (standard calculation method).
15	CyclomaticModified	Circle complexity (the second calculation method).
16	CyclomaticStrict	Circle complexity (the third calculation method).
17	Essential	Basic complexity (standard calculation method).
18	Knots	Measure overlapping jumps.
19	MaxEssentialKnots	The maximum node after the structured programming structure has been deleted.
20	MaxNesting	Maximum nesting level, relating to cyclomatic complexity.
21	MinEssentialKnots	The minimum node after the structured programming structure has been deleted.
22	RatioCommentToCode	Code comment rate.

enhanced the understanding of the relationship between the indicators' characteristics and the indicator feature values, thus increasing the ability of the model to generalize

5.4. Experimental Results. In the classification algorithm, there is often a phenomenon in which the model performs well for the training data but performs poorly for data outside of the training data set. In these cases, cross-validation can be used to evaluate the predictive performance of the model, especially the performance with new data, which can reduce overfitting to some extent. There are three general cross-validation methods: Hold-Out Method, K-fold Cross-Validation, and Leave-One-Out Cross-Validation. In this experiment, the K-fold cross-validation method was selected. K-fold cross-validation randomly divides the training set into k, k-1 for training, and the remaining data for testing, repeating the process k-times, and thus obtaining k models to evaluate the performance of the model. In this experiment, we validated the model into two parts: 80% of data set D was used as training data, and 20% was used as test data.

First, we group the data D into a training set to train the model and a test set to test the predictive performance of the model. This process is repeated until all data are used. Next, the K value is chosen to be 10, which is so that the data set is divided into 10 subsamples, 9 of which are used as training data and one is used as test data. Finally, the sensitivity of the model performance to data partitioning is reduced by averaging the results of 10 groups of results [22].

5.4.1. Results in C/C++ Programs. In this experiment, the program written in C/C++ language was used to conduct experiments, and the *decision tree* algorithm and other machine learning algorithms were compared with respect to accuracy and running time. The results are shown in Table 5.

Experiments showed that the *decision tree* algorithm achieved the best results regardless of the accuracy or running time, and the accuracy in the experiments reached 82.55%. Second, the data feature selection was verified by comparing the 22 metrics without feature selection and the 16 metrics after feature selection, as shown in Table 6.

Table 4: Mutual information value calculation results.

Metrics	C/C++ dataset		Java dataset	
	Mutual information value	Sort	Mutual information value	Sort
CountInput	0.1892	11	0.6375	3
CountLine	0.5584	1	0.7996	1
CountLineCode	0.4902	2	0.6446	2
CountLineCodeDecl	0.4327	5	0.4373	11
CountLineCodeExe	0.3893	9	0.5	7
CountLineComment	0.3980	8	0.499	8
CountOutput	0.1625	12	0.3477	13
CountPath	Nan	13	0.3959	12
CountPathLog	Nan	13	0.2638	17
CountSemicolon	0.4307	7	0.5484	5
CountStmt	0.4465	4	0.5879	4
CountStmtDecl	0.4313	6	0.439	10
CountStmtExe	0.3434	10	0.5123	6
Cyclomatic	Nan	13	0.3122	14
CyclomaticModified	Nan	13	0.3122	15
CyclomaticStrict	Nan	13	0.2968	16
Essential	Nan	13	0.0099	20
Knots	Nan	13	0.2508	18
MaxEssentialKnots	Nan	13	0.0099	21
MaxNesting	Nan	13	0.2236	19
MinEssentialKnots	Nan	13	0.0099	22
RatioCommentToCode	0.4603	3	0.4959	9

Table 5: C/C++ language dataset machine algorithm results.

	SVM	Bayes	Adaboost	Random forest	Decision tree
Accuracy (%)	82.06	37.69	35.47	82.54	82.55
Recall rate (%)	67.8	30.23	34.25	68.65	68.68
Precision rate (%)	71.06	31.46	33.75	73.59	73.62
Running time (s)	576.32	19.98	19.02	17.94	17.03

It was proved by experiments that the running time of the prediction buffer overflow vulnerability algorithm when using the $A = \{a_1, a_2, \ldots, a_{16}\}$ metrics after feature selection is reduced from the previous 17.03 s to 15.94 s, but the accuracy has hardly changed, so feature selection not only helps to reduce the running time but also yields higher accuracy and improves the efficiency of the vulnerability prediction. In the final experimental results, the functions predicted by Bad, Bad Sink and Bad Source are the probabilities of a high level of buffer overflow vulnerability. The three types of data can be extracted according to the function name for further analysis.

5.4.2. Results in Java Programs. The experimental results of the data set extracted by the program written in the Java language are shown in Tables 7 and 8.

In this experiment, we divided the buffer overflow vulnerability into eight categories. When there is vulnerability in real software, it does not necessarily contain all types of buffer vulnerabilities. There may be only one or several of them, and when we collect data in real software data, the data that may be vulnerable is much less than the data without vulnerability and may even reach a ratio of 1:10000. In the Java data set, because the experiment is a data set extracted from real software, the data volume of Good Source and Bad Sink is very small, which results in extremely unbalanced data. It also leads to the accuracy and recall rate of both types of data being 0. The accuracy and recall rate of other basic balanced vulnerability data have better results. To maintain the distribution of the original data, we need to improve the accuracy and recall rate of other basic balanced vulnerability data. To improve the accuracy of the results, data sampling is not used to balance the data, which then demonstrates the validity of the SVL model used to predict most types of buffer overflow vulnerabilities in real software. At the same time, the experiment proves that using the $A = \{a1, a2, \ldots, a16\}$ metrics to predict the buffer overflow vulnerability after feature selection reduces the running time and ensures high accuracy.

Table 6: *Decision tree* algorithm-specific operation results.

	Before feature selection			After feature selection		
	Precision	Recall	F1	Precision	Recall	F1
Good	88.09	83.56	85.76	88.03	83.47	85.69
Good Sink	61.27	52.91	56.78	61.27	52.91	56.78
Good Source	53.13	25.37	34.34	53.13	25.37	34.34
Good Auxiliary	77.16	96.34	85.69	77.16	96.34	85.69
Bad	77.08	49.23	60.09	77.08	49.23	60.09
Bad Sink	67.94	77.47	72.4	67.94	77.47	72.39
Bad Source	90.71	96	93.23	90.66	95.87	93.12
Average	73.63	68.7	69.76	73.61	68.67	71.05
Accuracy		82.55			82.53	
Time		17.03			15.94	

Table 7: Accuracy and time of Java dataset.

	SVM	Bayes	Adaboost	Random forest	Decision tree
Accuracy (%)	87.16	57.95	49.40	87.41	87.44
Recall rate (%)	59.56	48.25	34.98	59.64	59.64
Precision rate (%)	58.74	44.28	33.11	58.91	58.93
Running time (s)	104.32	9.98	8.99	12.94	11.99

Table 8: *Decision tree* algorithm-specific operation results.

	Before feature selection (%)			After feature selection (%)		
	Precision	Recall	F1	Precision	Recall	F1
Good	95.85	93.04	94.42	96.11	93.04	94.55
Good Sink	47.15	53.1	49.95	47.15	53.1	49.95
Good Source	0	0	0	0	0	0
Good Auxiliary	98.56	98.13	98.35	98.56	98.13	98.34
Bad	77.27	73.99	75.6	77.27	73.99	75.6
Bad Sink	0	0	0	0	0	0
Bad Source	95.74	99.51	96.54	93.75	99.6	96.59
Average	59.22	59.69	59.27	58.98	59.69	59.33
Accuracy		87.44			87.51	
Time		11.99			7.59	

6. Discussion

The software vulnerability analysis method proposed in this paper has good capability and has been applied and tested using real code and reliable software. Furthermore, the experiments verified the effectiveness of the *BOVP* model and provided guidance for its effective use. The model minimized the probability of misclassification while finding more vulnerabilities. This method has the following advantages: (1) Given the relative advantages of static and dynamic analysis of software vulnerabilities, the static metrics for source code are extracted by static analysis, and the dynamic metrics at the functional level are analysed according to the data flow of the function. The model can ensure the accuracy of vulnerability prediction and take into account the change in data flow between functions in running programs. This is a new vulnerability prediction method based on data flow analysis in a running program. (2) In this paper, the data set contained two different languages, namely, C/C++ and Java, and thus fully validated the validity of the *BOVP* model and proved that the model does not depend on a specific language type. (3) The model can be applied to vulnerabilities caused by multiple types of buffer overflows. The most common and most difficult software security vulnerability is buffer overflow vulnerability. The data set contains multiple types of buffer overflow vulnerabilities, providing valuable data for analysing different types of buffer overflows and offering a basis for future research. (4) To some extent, this research saves investment costs, time and human resources for software development. Applying feature selection without affecting the prediction results reduces the dimension and the experimental overhead, and it greatly improves the practicality of the software vulnerability prediction study. This paper provides a new way for software metrics to study

data collection in software vulnerability prediction. However, this method is only suitable for source code, not for nonopen source software.

7. Conclusion

The *BOVP* model proposed in this paper preprocesses the program source code and then employs the method of dynamic data stream analysis on the software functional level. By studying the characteristics of practical software code and different types of buffer overflow vulnerability features, the *decision tree* algorithm was developed through feature selection and the *Gini* index. Finally, a vulnerability prediction method based on software metrics was constructed. The capability of the *BOVP* model is verified by experiments using program source code written in different languages, and the prediction results are more accurate than other methods. The time taken for the data set collected using C/C++ was less, and the accuracy rate was 82.53%. The accuracy rate of the data set using Java was also high, reaching 87.51%, which fully demonstrated that this model is robust in predicting buffer overflow vulnerability in different types of languages.

The current buffer overflow vulnerability is very complicated and very common. Although it is divided into 8 categories, it does not contain all of the buffer overflow vulnerabilities. Therefore, the classification of buffer overflow vulnerabilities should be more comprehensive in the future. There are still some shortcomings in the method. For example, there is no corresponding improvement in the imbalance between Good Source and Bad Sink in the Java data set, and the methods of this research are mainly for open source software. It is believed that there is no detailed analysis of nonopen source software. In the future, we plan to solve the problem of unbalanced data, make up for the shortcomings of this model, and study the binary buffer overflow prediction model combined with dynamic symbolic execution technology to find increasingly complex types of buffer overflow vulnerabilities and also forecast vulnerabilities for nonopen source software.

Data Availability

The original data (Juliet Test Suite for C/C++ and Juliet Test Suite for Java) used to support the findings of this study can be download publicly from the website (https://samate.nist.gov/SRD/testsuite.php). The thirteenth page of the articles has footnotes giving the link address.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Key R&D Program of China under Grant no. 2016YFB0800700, the National Natural Science Foundation of China under Grants nos. 61802332, 61807028, 61772449, 61772451, 61572420, and 61472341, and the Natural Science Foundation of Hebei Province China under Grant no. F2016203330.

References

[1] C. Cowan, P. Wagle, C. Pu et al., "Buffer overflows: attacks and defenses for the vulnerability of the decade," IEEE, 2003.

[2] S. Wu, *Software Vulnerability Analysis Technology*, Science Press, 2014.

[3] Y. Shin and L. Williams, "Is complexity really the enemy of software security?" in *Proceedings of the ACM Workshop on Quality of Protection*, pp. 47–50, ACM, 2008.

[4] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[5] M. Frieb, A. Stegmeier, J. Mische et al., "Lightweight hardware synchronization for avoiding buffer overflows in network-on-chips," in *Proceedings of the International Conference on Architecture of Computing Systems*, pp. 112–126, Springer, Cham, Switzerland, 2018.

[6] Z. Wang, X. Ding, C. Pang, J. Guo, J. Zhu, and B. Mao, "To detect stack buffer overflow with polymorphic canaries," in *Proceedings of the 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 243–254, IEEE, June 2018.

[7] J. Durães and H. Madeira, "A methodology for the automated identification of buffer overflow vulnerabilities in executable software without source-code," in *Dependable Computing*, pp. 20–34, Springer, Berlin, Germany, 2005.

[8] S. Rawat and L. Mounier, "An evolutionary computing approach for hunting buffer overflow vulnerabilities: a case of aiming in dim light," in *Proceedings of the 6th European Conference on Computer Network Defense, EC2ND '10*, pp. 37–45, IEEE, October 2010.

[9] I. A. Dudina and A. A. Belevantsev, "Using static symbolic execution to detect buffer overflows," *Programming and Computer Software*, vol. 43, no. 5, pp. 277–288, 2017.

[10] S. Nashimoto, N. Homma, Y. I. Hayashi et al., "Buffer overflow attack with multiple fault injection and a proven countermeasure," *Journal of Cryptographic Engineering*, vol. 7, no. 1, pp. 1–12, 2016.

[11] J. Rao, Z. He, S. Xu, K. Dai, and X. Zou, "BFWindow: speculatively checking data property consistency against buffer overflow attacks," *IEICE Transaction on Information and Systems*, vol. E99D, no. 8, pp. 2002–2009, 2016.

[12] M. Bozorgi, *A machine learning framework for classifying vulnerabilities and predicting exploitability [Ph.D. thesis]*, Dissertations and Theses - Gradworks, 2009.

[13] X. Xu, R. Zhao, and L. Yan, "Research and progress in software metrics," *Journal of Information Engineering University*, vol. 15, no. 5, pp. 622–627, 2014.

[14] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: software metrics vs text mining," in *Proceedings of the International Symposium on Software Reliability Engineering*, pp. 23–33, IEEE, 2014.

[15] J. Stuckman, J. Walden, and R. Scandariato, "The effect of dimensionality reduction on software vulnerability prediction models," *IEEE Transactions on Reliability*, vol. 99, no. 1, pp. 1–21, 2017.

[16] G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, and C. Catal, "Empirical analysis of change metrics for software fault

prediction," *Computers and Electrical Engineering*, vol. 67, pp. 15–24, 2018.

[17] S. Rahimi and M. Zargham, "Vulnerability scrying method for software vulnerability discovery prediction without a vulnerability database," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 395–407, 2013.

[18] L. Breiman, J. H. Friedman, R. Olshen et al., "Classification and regression trees," *Biometrics*, vol. 40, no. 3, article 356, 1984.

[19] J. Han and M. Kamber, *Data Mining: Concepts and Techniques, Morgan Kaufmann*, Morgan Kaufmann, 2001.

[20] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[21] P. Viola and W. M. Wells III, "Alignment by maximization of mutual information," IEEE Computer Society, 1995.

[22] P. K. Shamal, K. Rahamathulla, and A. Akbar, "A study on software vulnerability prediction model," in *Proceedings of the 2nd IEEE International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET '17*, pp. 703–706, 2017.

Hindawi
Security and Communication Networks
Volume 2019, Article ID 3898951, 12 pages
https://doi.org/10.1155/2019/3898951

Research Article

An Insider Threat Detection Approach Based on Mouse Dynamics and Deep Learning

Teng Hu,[1,2] **Weina Niu**,[3] **Xiaosong Zhang**,[1] **Xiaolei Liu**,[4] **Jiazhong Lu**,[1] **and Yuan Liu**[2]

[1]*Center for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan, 611731, China*
[2]*Institute of Computer Application, China Academy of Engineering Physics, Mianyang, Sichuan, 621900, China*
[3]*College of Cybersecurity, Sichuan University, Chengdu, Sichuan, China*
[4]*School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China*

Correspondence should be addressed to Weina Niu; niuweina1@126.com

Received 25 September 2018; Revised 21 November 2018; Accepted 16 January 2019; Published 17 February 2019

Guest Editor: Jiageng Chen

In the current intranet environment, information is becoming more readily accessed and replicated across a wide range of interconnected systems. Anyone using the intranet computer may access content that he does not have permission to access. For an insider attacker, it is relatively easy to steal a colleague's password or use an unattended computer to launch an attack. A common one-time user authentication method may not work in this situation. In this paper, we propose a user authentication method based on mouse biobehavioral characteristics and deep learning, which can accurately and efficiently perform continuous identity authentication on current computer users, thus to address insider threats. We used an open-source dataset with ten users to carry out experiments, and the experimental results demonstrated the effectiveness of the approach. This approach can complete a user authentication task approximately every 7 seconds, with a false acceptance rate of 2.94% and a false rejection rate of 2.28%.

1. Introduction

Insider threats have always been one of the most severe challenges for intranets with security requirements [1], because they can cause system destruction, information exfiltration, etc. In recent years, with the frequent occurrence of insider threat events, intranet security has aroused increasing attention [2, 3].

Internal personnel have access to use internal proprietary systems, and they know internal security policies and protection techniques and review regulations from safety facilities [4], e.g., firewalls and IDS. Hence, internal personnel can bypass existing security facilities. Even worse, a malicious insider may also be the one who configures security measures [5]. Moreover, a cyberattack from insiders is prevalent within the organization [6]; according to the survey, 27% of all cybercrime incidents were suspected to be committed by insiders [2]. There are two reasons for these insider threats; on the one hand, employees with malicious intentions modify or steal organization's confidential information, trade secrets, or customer data for personal interests [7]. For example, insiders make use of sensitive information to obtain commercial interests or sell them to foreign organizations. On the other hand, internal employees with inadvertent behavior expose the organization's key assets and sensitive information to external opponents [8]. Furthermore, in some confidential organizations, the insider threat attacks may even be spy activities at the national level [8, 9]. Thus, the effective detection methods are worth studying.

The organizations in physical isolation high-security networks environment, such as confidential research institutes and military enterprises, are less likely to suffer external attacks. Thus, the internal attacks are the main reason for the leakage of sensitive information [10]. Fraudulent use of privileged users' terminals is the primary attack method for such internal threats [11].

There are two main reasons for this attack:

(1) Due to the negligence of internal employees, they left the workstation without logging out of the terminal equipment. As a result, malicious people use their terminals to unauthorized access and copy sensitive information;

(2) Others forge a privileged user's identity, such as password leakage or USB-key loss, and thus access privileged user's terminal for espionage activities.

Traditional authentication methods, such as passwords, USB keys, and fingerprints, determine the user identity when logging in. Thus, they cannot effectively discover end users with identity theft [12]. However, some approaches conduct continued authentication using human-computer interaction (HCI) behaviors [13] to solve the problem. HCI behaviors are unique biometric features from input devices like keyboards and mouses. For example, some researchers used interventional scenarios to capture HCI behaviors, like recording the user's responses during equipment failures. However, the interventional scheme has directly affected the convenience of using the devices, which may be identified by malicious users.

This paper proposed a continuous identity authentication method based on mouse dynamic behavior and deep learning to solve the insider threat attack detection problem. We verified the effectiveness of our proposed method on an open-source mouse dynamic dataset which contains the mouse dynamic data from ten users. The experimental results showed that our proposed method could identify the user's identities in seconds and has a lower false accept rate (FAR) and false reject rate (FRR). Specifically, the contributions of the work are as follows:

(i) We propose a novel continuous identity authentication method using mouse dynamic behavior and deep learning. It achieves better accuracy and lower verification time than existing methods.

(ii) Instead of manually extracting features from raw operations to characterize a user's unique mouse behavior characteristics, such as movement speed curves, we map the mouse dynamic behavior into pictures. Hence, the whole details of the mouse behavior can be preserved.

(iii) We construct a 7-layer CNN network to train the mouse behavior pictures datasets. The network converges with a small amount of data (about 18000 pictures). Moreover, the network can be used to train other mouse behavior datasets and implement identity authentication easily.

The remainder of this paper is organized as follows: Section 2 reviews the background and related work in the area of insider threat and mouse dynamics. Section 3 describes the method of preprocessing the dataset and the CNN network architecture and specific parameters. Section 4 presents our experimental design and results, and Section 5 concludes.

2. Background and Related Work

As early as 2006, the American Institute of Computer Security (CSI) issued a report that the insider threat caused by the malicious abuse of authority has exceeded the traditional Trojan attacks and has become the main threat to organizations [14]. The 2012 Annual Global Fraud Survey revealed that 60% of fraud cases were initiated by insiders [15]. The Edward Snowden incident in June 2013, known as "PRISM," caused widespread concern about insider threats all over the world. The 2014 US State of Cybercrime Survey released by CERT showed that 28% of insider attacks resulted in a loss of 46% [16]. In the same year, insider threats caused incredible damage to many well-known companies: for example, the Korean Credit Bureau has had 27 million accounts of credit card information stolen because of abusing access rights by its computer contractor [17]; the dismissed employee retaliated against US oil and gas company EnerVest, and all the web servers are restored to factory settings, thus leading to 30 days of disruption in overall communications and business operations and recovery costs of hundreds of thousands of dollars [18]. The 2016 US State Cybercrime Survey found that insiders caused 27% of e-criminals. The investigation report also revealed that 30% of the respondents believe that the insider attacks caused more serious losses than outsider attacks [19]. According to the 2017 CSO Cybercrime Survey [20], about 50 percent of organizations experienced at least one malicious insider incident in the previous year [21]. The above examples fully proved that insider threats had become the main cyberspace security threat faced by organizations.

The current research on insider threats is becoming more systematic and specific. Since 2001, the United States Secret Service (USSS) and Carnegie Mellon University have jointly established the CERT Insider Threats Center. The center collected more than 700 insider threat cases of fraud, theft, and destruction and thus solved the problem of insufficient data in insider threat research [22]. In 2011, the United States DARPA proposed building up a military insider threat detection system named ADAMS (Anomaly Detection at Multiple Scales) [23]. In the third year of the ADAMS project implementation, the SAIC company and four universities in the United States, including Carnegie Mellon University, jointly developed a realistic version of the ADAMS called PRODIGAL system and run tests on the actual enterprise data achieving good results [24].

The most common attack on insider threat is identity fraud. Since the user's identity cannot be continuously verified during the process of using the terminal, the malicious user has the opportunity to masquerade as a legitimate user. At present, research on the use of human-computer interaction data for insider threat attack detection has achieved good results in practical applications. The human-computer interaction data detection mainly studies the behavior pattern of the user using the computer input device, in which the mouse and the keyboard are mainly used as the data source.

In [12], the user's mouse operation when using IE browser was collected and based on these operations; the features such as moving coordinates, moving distance, angle, and moving time were constructed. The literature [25] focuses on

the three basic types of mouse movements, clicks, and drags and uses them to establish characteristics such as coordinates and speed. The deficiency of the above work is that the experimental data is too small, and malicious data adopts simulation data, which cannot truly reflect the actual attack conditions.

Research using keyboard data includes static text analysis and dynamic text analysis. For example, in [26], user input passwords are used to analyze the changes in user input methods. This study of user input of the same text is a static text analysis method. Another dynamic analysis is to study the user's arbitrary input of text. For example, the paper [27] analyzes the user's mail information and mainly records the keystroke behavior and time stamp. The insufficiency of this kind of research is that the habit of using the keyboard even by the same user may constantly be changing. It needs a large amount of input data to truly reflect a user's actual behavior characteristics, which is unrealistic in practical application.

Among different input devices, the mouse is a suitable choice. In some existing methods based on mouse dynamics, papers [12, 28, 29] have achieved low false reject rate (FRR) and false accept rate (FAR). For example, in the paper [28] FRR and FAR are approximately 2%. However, these methods have a common limitation. They require an average of several minutes (even more than 10 minutes) to collect enough mouse behavior for verification [30]. In the actual situation, attacks from insiders may only take tens of seconds or even seconds to complete the attack, such as copying files with sensitive information, sending emails with viruses, or just implanting one trojan in the machine. The methods commonly used in existing research are rough as follows: they collect data from users' valid mouse movements, extract selected features from them, and use these features to train models for classification. For example, the literature [28] uses the backpropagation neural network, and the literature [12] uses the C5.0 decision tree. There are two obvious disadvantages to these methods. One is the behavioral features of the mouse are artificial selected, which will inevitably lead to the loss of some details behavior; the other is that these methods need to collect a large number of valid movements when verifying the user's identity, so the verification takes longer.

The paper [13] used an interventional scenario method. In these scenarios, the user's mouse was disabled for a short period and collects the behavioral data generated by the user due to the loss of the mouse cursor during this period. The data for this period was used to verify the user's identity, achieving 5 seconds of verification and having 2.86% of FRR and 4% of FAR. However, this method also has obvious shortcomings. Regularly adopting an interventional scenario to verify user identity will inevitably seriously affect the convenience of using the terminal, and the attacker will easily perceive it. If the interval between two verifications is too long, the attacker will probably be able to complete the attack within the interval.

There are also some other approaches to detect insider threats. Reference [31] presents an ontological framework and a methodology for improving physical security and insider threat detection (called PS0). PS0 can facilitate forensic data analysis and proactively mitigate insider threats by leveraging rule-based anomaly detection. Reference [32] uses machine learning algorithms and extracts the frequently used words from the topic modeling technique and then verifies the analysis results by matching them to the information security compliance elements, to find the possible malicious insider from social media. Reference [33] implements a fuzzy classifier along with genetic algorithm (GA) to enhance the efficiency of a fuzzy classifier and the functionality of all other modules, to achieve better results in terms of false alarms. Reference [34] applies Hidden Markov method on a CERT dataset and analyses a number of distance vector methods in order to detect changes of behavior, which are shown to have success in determining different insider threats.

3. Overview of Our Approach

From previous research on insider threat detection based on mouse behavior, researchers usually extract some mouse features based on the basic mouse movements (which we call raw data, including clicks, moves, and drags), such as direction, velocity, Angle of Curvature, Curvature Distance, and Pause-and-Click [35], and then go through these features to determine human behavior characteristics, to conduct user authentication [36].

This kind of method has achieved outstanding results, but all have a common shortcoming. Researchers are all extracting features based on their own experience and understanding. This method has certain limitations; during the process of extracting features, some researchers use only mouse click actions; some researchers use the moving distance combined with clicks to generate features, and others consider other basic mouse actions. But there is still much raw data that can reflect the individual's unique being ignored and therefore affects the accuracy of recognition.

We propose a method that can completely retain all basic mouse operations and use deep learning for user authentication. First of all, we map all actions generated by the mouse to images through a particular method. Then we train the image datasets through the CNN network to create classification models. In the process of authentication, the user's mouse operation is mapped according to the same method, and then classified by the trained model, so as to achieve the purpose of user identification. Our approach makes full use of the advantages of deep learning. First, in the process of mapping mouse behaviors into images, we retain all basic mouse operations. Neither need to manual extract features to train these image datasets when using the CNN network, nor need the feature extraction algorithm in traditional machine learning. The convolutional neural network can automatically complete feature extraction and abstraction in the training. Secondly, there are many successful and efficient solutions to the problem of how to use CNN to classify images. In order to take this advantage, we map the mouse actions to the behavior trajectory on the picture, which based on mouse behavior. This turns the problem of the user authentication based on mouse behavior into a classic image classification problem.

3.1. Data Preprocessing

3.1.1. Dataset. Many of the previous studies collect the raw mouse data by themselves and use for analysis and

experimentation. Most of the datasets have not been disclosed. Even if some researchers open source their own datasets, they are currently unavailable for download. To prove the effectiveness of our method, we choose to use the open-source mouse dynamic dataset, which published on GitHub [37], for our experiments.

The dataset stores the mouse dynamic data from ten users. The dataset consists of two parts, one part is stored in the "training_files" folder and contains data session files from ten users who normally operate the mouse, and store them separately in their respective named folders. There are 5-7 long session data files in each user folder; the other part stores them in the "test_files" folder, which also contains ten folders named after ten users. Each user folder stores many short session data files. Some of these session files are actually not generated by current users. The dataset also gives a "public_labels.csv" file that marks whether the session data in the folder is legal or not.

Each session is stored in the following format:

$$[\text{record timestamp}, \text{client timestamp}, \text{button}, \text{state}, x, y] \quad (1)$$

The specific meanings are as follows [37]:

record timestamp: elapsed time (in a sec) since the start of the session as recorded by the network monitoring device;

client timestamp: elapsed time (in a sec) since the start of the session as recorded by the RDP client;

button: the current condition of the mouse buttons;

state: additional information about the current state of the mouse;

x: the x coordinate of the cursor on the screen;

y: the y coordinate of the cursor on the screen;

The data in "training_files" is quite complete and large enough compared to the data in "test_files". We divided "training_files" into two parts, one to train our model and the other to verify whether our model is effective or not. In addition, we use "test_files" to further verify that our model is still accurate enough for insider threat detection.

3.1.2. Mouse Dynamic Mapping Method. All the basic operations that the mouse can produce are move, Click, Drag, Scroll, and Stay. In previous studies, the researchers usually extract features from these five basic actions based on their experience. Such as moving distance, moving speed, click frequency, etc., which can reflect the behavior of individuals using the mouse to a certain extent. However, there are also two obvious shortcomings. First, the use of a single feature (or a combination of multiple feature vectors) does not fully or accurately reflect the individual's unique behavioral characteristics; the second is that more time is required in the process of acquiring features, because the feature vectors (some researchers call them effective operations) are extracted from the basic operations. Undoubtedly, relatively more basic actions are needed to generate enough effective operations.

In order to completely preserve the features generated by people using the mouse, we map all the basic actions generated by users to the image. Because the data generated by the mouse is a one-dimensional dataset, we map these one-dimensional datasets to two-dimensional tensors according to the following mapping method. The specific mapping method is as follows:

Figure 1: The position of the mouse is represented by a two-dimensional coordinate system (x, y), and the distance between two mouse positions is expressed as a movement feature of the mouse. The movement features can reflect the personality characteristics of users such as moving speed, moving angle, moving range, and average moving distance.

Require 1. m, the number of mouse basic actions;

Require 2. xMax, the maximum x coordinate in a session; yMax, the maximum y coordinate in a session;

Step 1. Take m mouse operations according to the data sequence of the session;

Step 2. Construct a coordinate system D based on xMax and yMax;

Step 3. Extract all the "Move" operations in this m, record all the coordinates of "Move" $(x, y) \in [\text{xMove}, \text{yMove}]$, and map a red color line on D according to the coordinate distance of two actions operations, as shown in Figure 1;

Step 4. Extract all the "Pressed" operations in m, the pressed operation $(x, y) \in [\text{xLeft\_P}, \text{yLeft\_P}]$ for the left button, and $(x, y) \in [\text{xRight\_P}, \text{Right\_P}]$ for the right button. Similarly, all "Released" operations' coordinates of the left and right button are recorded in [xLeft_R, yLeft_R] and [xRight_R, Right_R], respectively. To distinguish between the left and the right buttons and the difference between "Pressed" and "Released" in the image, the left button "Pressed" is represented by a blue "circle" on D, the left button "Released" is represented by a green "x", the right button "Pressed" is represented by a black ".", and the right button "Released" is represented by a black "x", as shown in Figure 2;

Step 5. Extract all the "Drag" operations and $(x, y) \in [\text{xDrag}, \text{yDrag}]$ is represented by a thick yellow line on D, as shown in Figure 3;

Step 6. In the same way, all "Scrolls" are extracted. Scroll's "Up" operations are recorded in [scroll_up_x, scroll_up_y] and "Down" operations are recorded in [scroll_down_x, scroll_down_y]. The "Scroll Up" is represented by a cyan upward triangle"△" and the "Scroll Down" is represented by "▽", as shown in Figure 4;

Step 7. In this m, if there are two consecutive actions on the same coordinate and stay for a while, we believe that this

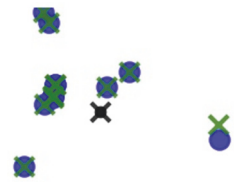


Figure 2: The click action is actually made up of two actions: Pressed and Released. So a "click" action is actually done by a Pressed and a Released. Two consecutive times on the same coordinate (x, y) becomes a "double click" action. In addition, the mapping method also needs to distinguish the left and right buttons of the mouse, that is, "click" and "double-click" of the left mouse button and "click" and "double-click" of the right mouse button. It can reflect the features of the number of clicks, frequencies, and other features.

Figure 3: The mouse did not release immediately after pressing "Pressed" but "drag" a distance and then "Released," which is a drag operation. Drag operations are also common in actual mouse use and are often not noticed as features of the user.

period can also reflect the personal operating habits and call it a "Stay" operation. (x, y, s) ∈ [stay_x, stay_y, stay_s] represents all "Stay" operations in m, where x and y are the coordinates of the "Stay" and s represents the relative time of operation. It is represented by light red translucent squares on D, and the square size is linearly related to s, as shown in Figure 5;

Step 8. Save the mapped D as a picture in JPG format, so we get a track diagram of the mouse's behavior in units of m basic operations.

Step 9. Repeat Steps 2-8 until the number of remaining operations in a session is less than m, and get n pictures of a user in a session, as shown in Figure 6.

According to the mapping method, all sessions of all users in "training_files" store them in the folders named by each user, and we get a dataset that can be trained by the CNN network. Each username is the label of each sub-dataset. We generated image sets for training CNN models in units of m=25, 50, 100, 500, and 1000, respectively, as shown in Figure 7.

3.1.3. Data Augmentation. Although the open-source dataset was used to generate 10 sets of tagged datasets, as shown in table x, the amount of generated images is insufficient. Therefore, we use three methods for data augmentation. One is to flip the image, including horizontal flip and vertical flip; The other is to rotate the image by 90 degrees, 180 degrees, and 270 degrees, respectively; The third is to randomly rotate the image by 25 degrees. Each picture will be judged according to the probability of 50% on whether to perform the above operation. As long as the dataset reaches our preset target, the dataset stops to augment. In this paper, our default augmentation goal is 18,000, which is to augment each user's dataset to 18,000 images.

Figure 4: Mouse "Scroll" operation is also often ignored by researchers, but it can also reflect people's habitual operating characteristics. The "Scroll" operation is divided into two Up and Down scroll actions.

Figure 5: Usually, researchers think of mouse movements and clicks as mouse operations. They do not pay attention to the interval between two mouse operations. In fact, this interval is also an "operation" of the mouse, and we call it the "Stay" operation. When there is no other operation on the mouse in one coordinate or the interval between two operations of the mouse, it can be defined as the "operation," which is represented by a semitransparent square on the two-dimensional image. The size of the square is used to indicate the length of stay.

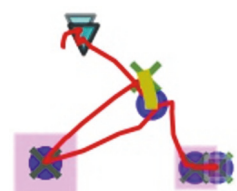
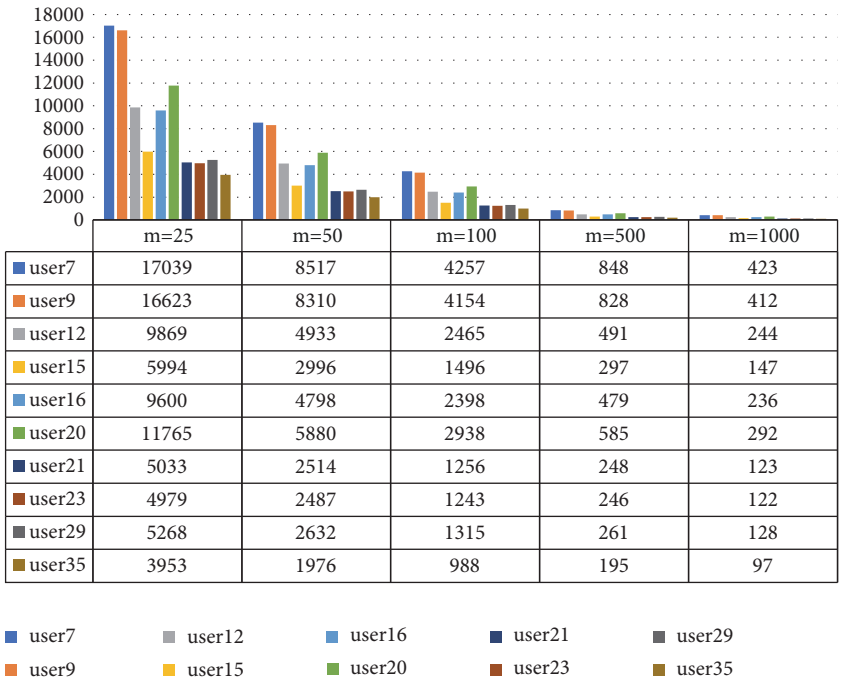


Figure 6: A picture of a user's mouse behavior (when m=100).

3.2. Overall Architecture of CNN. We refer to the networks of Alexnet [38], VGG [39], and GoogleNet [40]. We do not have such a large training set, so we do not choose to use such deep networks as VGG and GoogleNet. In fact, an 8-layer network constructed entirely in accordance with Alexnet parameters is not very applicable in our experiments. Therefore, we have constructed a 7-layer CNN network as shown in Figure 8.

The first four layers are convolution layers, and the remaining three are fully connected layers. A max-pooling layer follows each convolutional layer. The output of the first two full-connected layers is processed by Dropout. The output of the last layer is sent to Softmax and obtained the probability distribution of the classified labels. Apply the ReLU nonlinearity to the output of all convolutional and all fully connected layer except the last one.

Details of Parameters. The first convolutional layer filters the 100∗100∗3 input image with 32 kernels of size 3∗3∗3 with a stride of 1 pixel. The second convolutional layer takes as input the output of the first convolutional layer and filters it with 64 kernels of size 3∗3∗32. The third convolutional layer has 128 kernels of size 3∗3∗64 connected to the outputs of the second convolutional layer. The fourth convolutional layer has 128 kernels of size 3∗3∗128. The first fully connected layers have 1024 neurons and the second have 512, and both have passed the Dropout layer processing with a probability of 0.5.



	m=25	m=50	m=100	m=500	m=1000
user7	17039	8517	4257	848	423
user9	16623	8310	4154	828	412
user12	9869	4933	2465	491	244
user15	5994	2996	1496	297	147
user16	9600	4798	2398	479	236
user20	11765	5880	2938	585	292
user21	5033	2514	1256	248	123
user23	4979	2487	1243	246	122
user29	5268	2632	1315	261	128
user35	3953	1976	988	195	97

Figure 7: Image dataset generated from "training_files" in a unit of m. As the value of m increases, the dataset of user pictures becomes smaller.

(a) Activation Function. The CNN with Rectified Linear Units is much faster than the CNN with tanh [41], so we use the nonlinear activation function ReLU. The function form of ReLU is $f(x) = x^+ = \max(0, x)$. This nonlinear function means that when x is less than 0, the output is always 0, and when x is greater than 0, the output is the input value.

(b) LRN Layer. Although the concept of LRN (Local Response Normalisation) was mentioned for the first time in Alexnet and it also shows that LRN can reduce the error rate of its model, in [39] it is mentioned that LRN has little impact on the network and increased time-consuming. In our actual training, we also found that adding the LRN layer will increase the training time, but the improvement of training results is not very obvious. Therefore, the LRN layer is not added after convolution.

(c) Pooling Function. We used the max-pooling function [42], which is a commonly used pooling function in current CNN networks. The max-pooling can be expressed as taking the maximum value of this area within a certain rectangular area (z∗z) instead of the output of the network at this location and performing pooling every S pixels.

(d) Dropout. To prevent overfitting of the model, we added the Dropout layer to the first two layers fully connected layers. In machine learning, the output of multiple models is usually integrated to reduce the generalization error, such as Bagging [43], but when each model is a large neural network, the training and evaluation of such networks requires a lot of time and memory, as [40] integrates six neural networks, beyond which it will become difficult to handle. While Dropout provides an approximate Bagging integration method [44, 45], Dropout randomly drops some of the nodes' outputs with a certain probability, so that the dropped output does not participate in the propagation. Each round of training randomly drops some nodes, which is equivalent to a part of the network forming a subnetwork or submodel. This can effectively reduce the complex coadaptation between neurons; because of random inactivation, the neuron cannot be overly dependent on a previous neuron with an output. In training our model, we set the Dropout probability to 0.5.

(e) Gradient Optimization Algorithm. We used the Adam [45] optimization algorithm provided by Tensorflow, which is an optimization algorithm with learning rate adaptation and introduces quadratic gradient correction. The specific implementation algorithm in Tensorflow is as in Algorithm 1.

Where learning_rate is stepsize, which is the learning rate, Beta1 and beta2 are the exponential decay rates of first-order moment estimation and second-order moment estimation, respectively. Momentum in other optimization algorithms is directly incorporated into first-order moment estimation in Adam. The range of moment estimation is [0, 1); Epsilon is a small constant used for numerical stability. The default value of this constant is 1e-08, but it is necessary to test the best choice based on experiments. The variable is updated according to the gradient g. In our model training, we set learning_rate=0.01; beta1=0.9; beta2=0.999; epsilon=1e-08.

4. Experiments and Results

In this section, we will conduct three experiments. We completed the entire model training and experiments in

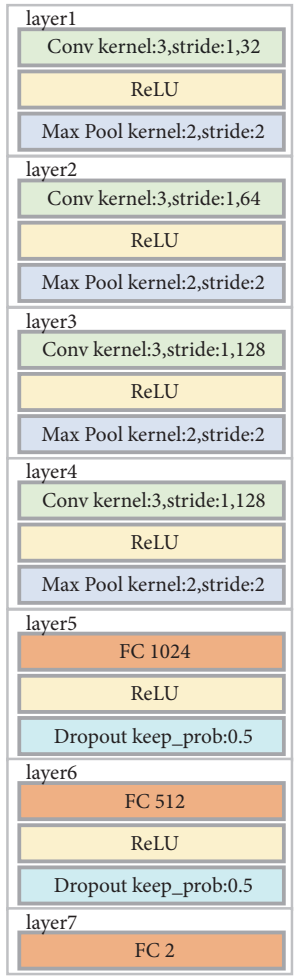


Figure 8: The architecture of our CNN.

the following experimental environment: Python 3.5.2, Tensorflow r1.4, CUDA Version 8.0.61, cudnn-8.0-windows10-x64-v6.0, windows10, and NVIDIA GTX 1060 6GB GPU. Experiment A is to verify the effectiveness of our method. Through the experiment, we can confirm that the use of the CNN network is effective for identity authentication based on mouse features and achieves good FAR and FRR values. Experiment B is to illustrate that our method requires very little time for authentication and can perform continuous user identity authentication after the user logs in to the terminal. Experiment C is to experiment with the "test-files" data provided by the dataset. Our experiment is designed to be a scene that needs to be faced with a real insider threat attack and takes measures to reduce FAR as much as possible. Experiments can show that our method can be applied in practical situations.

4.1. Experiment A: Identity Authentication. We believe that, in the insider threat detection scenario, the problem to be solved by the identity authentication should be judging whether the person currently using the terminal is consistent with the currently logged-in user. Therefore, we designate one user as a legal user and nine other users as illegal users (we total

```
t <- t + 1
lr_t <- learning_rate * sqrt(1 - beta2^t) / (1 - beta1^t)
m_t <- beta1 * m_{t-1} + (1 - beta1) * g
v_t <- beta2 * v_{t-1} + (1 - beta2) * g * g
variable <- variable - lr_t * m_t / (sqrt(v_t) + epsilon)
```

Algorithm 1: Adam optimization algorithm.

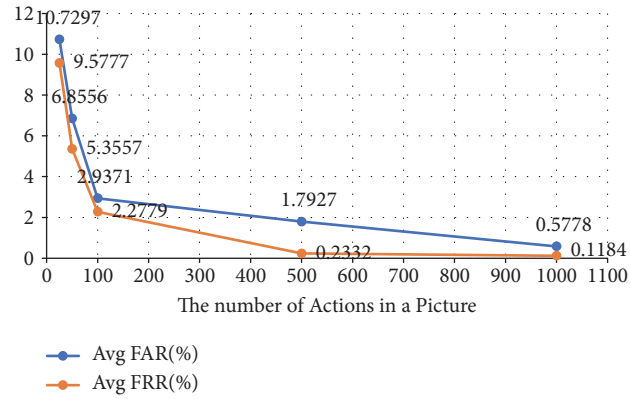


Figure 9: Trend chart of average values of FAR and FRR.

have ten users' mouse dynamic data). This is a typical binary classification problem. To verify the effect of our method, we design the experiment as follows.

Step 1. Use the image dataset that was generated in Section 3.1.2. Appoint a legal user (such as user 12), and extend the subdataset D as described in Section 3.1.3. Then, divide the dataset D into a training set T0 and a test set T0′ according to the ratio of 85% and 15%;

Step 2. According to the T0 and T0′, randomly extract the same amount from the other nine users' subsets, to construct an illegal user training set T1 and an illegal test set T1′. And ensure that there is no intersection between T1 and T1′.

Step 3. Take T0 and T1 as input, and use the CNN network constructed in Section 3.2 to train the model. Then, test T0′ and T1′ with the generated model and calculate FAR and FRR.

Step 4. Appoint one of the ten users as the legal user, and the remaining nine are considered as illegal users. Repeat the above experiment steps and calculate the average FAR and FRR.

In this experiment, we made T0 + T0′ = 18000 and T1 + T1′ = 18000. Hence, the size of training set is T0 + T1 = 30600(85%), and the size of test set is T0′ + T1′ = 5400(15%). The above experiment was conducted for different mouse operation datasets (m=25, 50, 100, 500, or 1000), and the experimental results were shown in Table 1. Figure 9 shows the average values of FAR and FRR vary with m, and they decrease as m increases (the number of features on each image increases).

Table 1: The results of experiment A.

User	m=25		m=50		m=100		m=500		m=1000	
	FAR(%)	FRR(%)	FAR(%)	FRR(%)	FAR(%)	FRR(%)	FAR(%)	FRR(%)	FAR(%)	FRR(%)
7	16.556	10.333	7.37	6.37	4.889	2.444	4.519	0.185	1.778	0.259
9	9.519	10.815	6.222	3.889	2.704	2.556	2.444	0.037	0.519	0.037
12	18.37	12.259	8.148	9.63	4.222	2.963	1.259	0.444	0.852	0.296
15	9.407	7.037	11.593	7.63	4.963	1	0.519	0.148	0	0.259
16	10.667	9.185	7.556	4.148	5.111	1.852	1.037	0.074	0.444	0
20	6.074	8.185	9.63	1.556	2.444	1.112	0.852	0.37	0.519	0.037
21	7.704	8.704	4.407	4	1	2.519	0.778	0.519	0.222	0.037
23	7.259	9.222	3	5.852	0.63	3.481	1.704	0.37	0.259	0
29	12.481	10.333	6.963	5.593	1.704	2.667	1.037	0.148	0.481	0.185
35	9.26	9.704	3.667	4.889	1.704	2.185	3.778	0.037	0.704	0.074
Avg	*10.7297*	*9.5777*	*6.8556*	*5.3557*	*2.9371*	*2.2779*	*1.7927*	*0.2332*	*0.5778*	*0.1184*

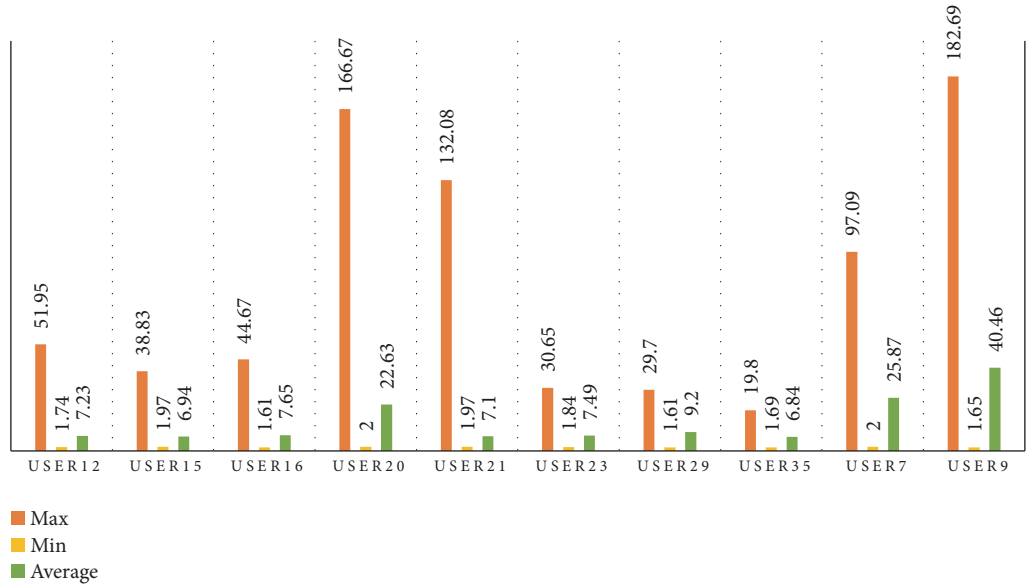


Figure 10: Number of mouse actions per second generated by users.

4.2. Experiment B: The Time Cost of Identity Authentication. In our opinion, the authentication time is composed of the time needed to collect the mouse features and the time required for classification. Compared with the time of collecting mouse features, the time of classifying mouse features using the trained model is almost negligible. Hence, our primary concern is the time required to obtain enough features. It can be seen from the data set analysis that the number of operations per second generated by the user when using the mouse normally has individual differences. The detailed data is shown in Figure 10.

As can be seen from Figure 10, the user (user 9) generated up to about 182 operation actions in one second. That is to say, in the fastest case, it only takes less than 1 second to complete 100 operations collection and authentication (when m=100). In the slowest case, the user (user 16 and user 29) only generates about 1.61 operations per second, which takes about 1 minute to complete the operation collection and authentication (when m=100). On average, ten users can generate 14.141 mouse operations per second, and about 7 seconds can complete acquisition and authentication (when m=100). The mouse operation data we use is the raw data of the mouse (as we mentioned in Section 3). We analyzed the average time and the minimum time, respectively, under different values of m (when m=25, 50,100,500,1000), the result shown in Table 2.

4.3. Experiment C: Insider Threat Detection. In an insider threat attack scenario, the insider is familiar with the internal system; it is possible to copy sensitive information in a very short time. Therefore, we believe that the authentication time should be within 10 seconds, and the time intervals between two authentications should also be controlled within 10 seconds. According to the above experimental results, we select the third model (m = 100) for the next experiment. The model was able to complete authentication in about 7 seconds

Table 2: Average time and the minimum time required to acquire mouse actions.

m	Avg Time(s)	Min Time(s)
25	1.768	0.315
50	3.536	0.63
100	7.072	1.26
500	35.358	6.296
1000	70.716	12.592

Table 3: The test data set generated according to the mapping method (when m=100), in which each user folder contains legal sessions and illegal sessions.

test_files	legal	illegal	total_picutres
user7	36	37	1659
user9	23	43	1585
user12	56	49	1306
user15	45	70	1238
user16	68	38	1173
user20	30	20	1089
user21	37	22	605
user23	38	33	765
user29	43	20	684
user35	35	73	1076
Total	411	405	11180

Note. During the generation process, we found that some sessions do not have label information in "public_labels.cvs". Because we could not confirm these sessions are legitimate data or not, we removed this data information.

on average and reached 2.94% FAR and 2.28% FRR. We think this can basically meet the needs of such insider threat attack detection.

We will use the test set ("test_files") provided by the dataset for the experiment and then determine whether a session is legal data based on the labels ("public_labels.csv") provided by the dataset. The user data in "test_files" is mapped according to the mapping method in Section 3.1.2 to generate a test data set. The results are shown in Table 3.

It would be fair to say that an insider threat detection system with low false reject rates may be tolerated, but an insider threat detection system with low false accept rates cannot be allowed. That is because if there is a false reject event, the results of the false reject event report can be assisted by various measures, such as on-site inspection, video surveillance, IDS, firewall, and SOC, and the false reject event can be verified. But if a malicious behavior of espionage is not be detected, that means the attacker has successfully achieved the goal and will incur an incalculable loss to the organization. That is to say, the system designing is to minimize FAR but with FRR-tolerant in the actual application scenario.

Therefore, we design this experiment according to the purpose of reducing the FAR as much as possible. A session represents the beginning of a mouse session, in which mouse actions are generated in chronological order. In the actual authentication scenario, authentication is performed every time sufficient actions are generated (we generate pictures according to the setting of m=100). We do not consider the current user to be a legitimate user until three consecutive authentications are legal. In other words, each authentication result is compared with the previous two authentication results, and a warning is issued as long as one of the authentication results is illegal. In this way, the actual authentication requires three pictures (m=100∗3), which can effectively reduce the FAR. The experimental results are shown in Table 4.

Table 4: The results of experiment C.

User	FAR(%)	FRR(%)
7	0	3.223
9	0	2.365
12	2.5	7.537
15	0	3.704
16	0	12.776
20	0	12.296
21	0	23.958
23	0	22.52
29	0	11.556
35	0	15.73
Average	0.25	11.5665

4.4. Comparison with Previous Work. In this section, we show a comparison of our experimental results against the results of previous works, which are shown in Table 5. The verification time is highly dependent on the number of mouse actions. As described in Section 3.1.2, the type of mouse actions can be divided into Click, Move, Drag, Scroll, and Stay. We choose to use all these five raw mouse actions to construct the authentication model, but the previous works do not use all the basic actions and try to extract features from the basic actions. Reference [28] uses move, drag, click, and stay for authentication, but they need about 2000 mouse actions and 1033 seconds. Reference [35] needs 20 mouse clicks, but sometimes they need a very long time to collect enough clicks (average of 37.73 minutes). If they also use mouse move actions, the verification time can reduce to 3.03 minutes. Reference [36] chooses to use mouse click and mouse movement to construct the features.

Generally speaking, as the number of mouse actions increases, both FAR and FRR show a downward trend, but the verification time increases accordingly.

5. Conclusion

Many previous studies have shown that mouse biobehavioral features can authenticate users. In this paper, we focus on the challenges of using mouse behavioral features for insider threat detection and propose a method that combines deep learning with mouse biobehavioral features for insider threat detection. This method can complete a user authentication task in a very short time while maintaining high accuracy. In the previous studies, one or several basic actions were selected

Table 5: Compare with previous works.

Source	FAR	FRR	Data required	Authentication time
[28]	2.46%	2.46%	2000 mouse actions (click/move/drag/stay)	1033 seconds
[35]	1.30%	1.30%	20 mouse clicks (click or click/move)	37.73 minutes(click) or 3.03 minutes(click/move)
[36]	8.74%	7.69%	32 mouse operations (click/move)	11.80 seconds
	4.69%	4.46%	160 mouse operations (click/move)	59.49 seconds
	3.33%	2.12%	320 mouse operations (click/move)	118.14 seconds
ours	10.73%	9.58%	25 mouse actions (click/move/drag/stay/scroll)	1.768 seconds
	6.86%	5.36%	50 mouse actions (click/move/drag/stay/scroll)	3.536 seconds
	2.94%	2.28%	100 mouse actions (click/move/drag/stay/scroll)	7.072 seconds
	1.79%	0.23%	500 mouse actions (click/move/drag/stay/scroll)	35.358 seconds
	0.58%	0.12%	1000 mouse actions (click/move/drag/stay/scroll)	70.716 seconds

from mouse five basic actions, and these actions were used to extract features to describe the unique behavioral characteristics of the user and then classified by using methods such as SVM, to realize user authentication. We use all five basic mouse actions to prevent the user's unique behavior characteristics from being omitted. Then, we map the user's mouse actions into pictures and automatically extract and model the user behavior pictures through the CNN network of deep learning. We use an open-source mouse behavior dataset that contains mouse action data from 10 users. The experiments have demonstrated the effectiveness of the proposed approach, with a false acceptance rate of 2.94%, a false rejection rate of 2.28%, and the authentication time of 7.072 seconds (when m = 100). These results show that this approach can be applied to detect insider threat attacks in specific scenarios.

Data Availability

The mouse dynamic data used to support the findings of this study were supplied by Balabit Mouse Dynamics Challenge dataset and available at https://github.com/balabit/Mouse-DynamicsChallenge.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by Defense Industrial Technology Development Program JCKY2018212C020 and JCKY2016212C005 and in part by the National Natural Science Foundation (NSFC) under Grant CNS 61572115.

References

[1] Q. Yaseen and B. Panda, "Insider threat mitigation: preventing unauthorized knowledge acquisition," *International Journal of Information Security*, vol. 11, no. 4, pp. 269–280, 2012.

[2] R. Trzeciak and D. Costa, *Model-Driven Insider Threat Control*, Carnegie Mellon University, 2017, https://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_509187.pdf.

[3] B. Bose, B. Avasarala, S. Tirthapura, Y.-Y. Chung, and D. Steiner, "Detecting insider threats using RADISH: a system for real-time anomaly detection in heterogeneous data streams," *IEEE Systems Journal*, vol. 11, no. 2, pp. 471–482, 2017.

[4] D. Liu, X. Wang, and J. Camp, "Game-theoretic modeling and analysis of insider threats," *International Journal of Critical Infrastructure Protection*, vol. 1, pp. 75–80, 2008.

[5] http://www.cert.org/insider-threat/index.cfm.

[6] I. Homoliak, F. Toffalini, J. Guarnizo, Y. Elovici, and M. Ochoa, "Insight into insiders: A survey of insider threat taxonomies, analysis, modeling, and countermeasures," 2018, https://arxiv.org/abs/1805.01612.

[7] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: a survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1397–1418, 2018.

[8] M. Collins, *Common Sense Guide to Mitigating Insider Threats*, Carnegie Mellon University, Pittsburgh, PA, USA, 2016.

[9] S. L. Pfleeger, J. B. Predd, J. Hunker, and C. Bulford, "Insiders behaving badly: addressing bad actors and their actions," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 1, pp. 169–179, 2010.

[10] M. Watkins and K. Wallace, *Understanding Network Security Principles*, Chapter 1, NetworkWorld, 2009, https://www.networkworld.com/article/2268110/lan-wan/chapter-1–understanding-network-security-principles.html.

[11] S. Eberz, K. Rasmussen, V. Lenders, and I. Martinovic, *Preventing Lunchtime Attacks: Fighting Insider Threats with Eye Movement Biometrics*, 2015.

[12] M. Pusara and C. E. Brodley, "User re-authentication via mouse movements," in *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pp. 1–8, ACM, USA, October 2004.

[13] X. J. Chen, F. Xu, R. Xu, S. M. Yiu, and J. Q. Shi, "A practical real-time authentication system with Identity Tracking based on mouse dynamics," in *Proceedings of the Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference*, pp. 121-122, IEEE, 2014.

[14] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson, *CSI/FBI Computer Crime and Security Survey*, 2006.

[15] Kroll and Economist Intelligence Unit, *Annual Global Fraud Survey, 2011/2012*, 2012.

[16] *2014 US State of Cybercrime Survey*, US CERT, Camegie Mel-lon University, 2014.

[17] *Credit Card Details on 20 Million South Koreans Stolen*, BBC News, 2014, https://www.Bbc.Com/News/Technology-25808189.

[18] *Former Engineer Sentenced to Prison for Destroying Company's Computer System*, WSAZ News, 2014, http://www.wsaz.com/home/headlines/Former-Engineer-Sentenced-to-Prison-for-Destroying-Companys-Computer-System-259992681.html.

[19] *2016 State of Cybercrime Survey*, 2016, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=499782.

[20] *2017 U.S. State of Cybercrime*, CSO, 2017, https://images.idgesg.net/assets/2017/08/idg_presentation_cybercrime_07202017_final_compressed.pdf.

[21] *5 Best Practices for Preventing and Responding to Insider ... (n.d.)*, 2017, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=509914.

[22] *The CERT Guide to Insider Threats*, US CERT, 2012, http://www.cert.org/insider-threat/.

[23] *Anomaly Detection at Multiple Scales (ADAMS) Broad Agency Announcement DARPA-BAA-11-04 (PDF)*, General Services Administration, 2011.

[24] E. Ted, H. G. Goldberg, A. Memory et al., "Detecting insider threats in a real corporate database of computer usage activity," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery And Data Mining*, pp. 1393–1401, ACM, August, 2013.

[25] A. Weiss, A. Ramapanicker, P. Shah, S. Noble, and L. Immohr, "Mouse movements biometric identification: a feasibility study," in *Proceedings of Student/Faculty Research Day*, CSIS, Pace University, White Plains, New York, NY, USA, 2007.

[26] K. Killourhy and R. Maxion, "Why did my detector do that?!," in *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, pp. 256–276, Springer, Berlin, Heidelberg, 2010.

[27] A. Messerman, T. Mustafić, S. A. Camtepe, and S. Albayrak, "Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics," in *Proceedings of the Biometrics (IJCB), 2011 International Joint Conference*, pp. 1–8, IEEE, USA, October 2011.

[28] A. A. E. Ahmed and I. Traore, "A new biometric technology based on mouse dynamics," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 3, pp. 165–179, 2007.

[29] H. Gamboa and A. Fred, "A behavioral biometric system based on human-computer interaction," in *Biometric Technology for Human Identification*, vol. 5404, pp. 381–393, International Society for Optics and Photonics, August, 2004.

[30] Z. Jorgensen and T. Yu, "On mouse dynamics as a behavioral biometric for authentication," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pp. 476–482, ACM, March, 2011.

[31] V. Mavroeidis, K. Vishi, and A. Jøsang, "A framework for data-driven physical security and insider threat detection," in *Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 1108–1115, IEEE, August, 2018.

[32] W. Park, Y. You, and K. Lee, "Detecting potential insider threat: analyzing insiders' sentiment exposed in social media," *Security and Communication Networks*, vol. 2018, Article ID 7243296, 8 pages, 2018.

[33] M. Bin Ahmad, A. Akram, M. Asif, and S. Ur-Rehman, "Using genetic algorithm to minimize false alarms in insider threats detection of information misuse in windows environment," *Mathematical Problems in Engineering*, vol. 2014, Article ID 179109, 12 pages, 2014.

[34] O. Lo, W. J. Buchanan, P. Griffiths, and R. Macfarlane, "Distance measurement methods for improved insider threat detection," *Security and Communication Networks*, vol. 2018, Article ID 5906368, 18 pages, 2018.

[35] N. Zheng, A. Paloski, and H. Wang, "An efficient user verification system via mouse movements," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pp. 139–150, ACM, October 2011.

[36] C. Shen, Z. Cai, X. Guan, Y. Du, and R. A. Maxion, "User authentication through mouse dynamics," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 16–30, 2013.

[37] Á. Fülöp, L. Kovács, T. Kurics, and E. Windhager-Pokol, *Balabit Mouse Dynamics Challenge Data Set*, 2016, https://github.com/balabit/Mouse-Dynamics-Challenge.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[39] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014, https://arxiv.org/abs/1409.1556.

[40] C. Szegedy, W. Liu, Y. Jia et al., *Going Deeper with Convolutions*, 2014, https://arxiv.org/abs/1409.4842.

[41] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann," in *Proceedings of the 27th International Conference on Machine Learning (ICML '10)*, pp. 807–814, Haifa, Israel, June 2010.

[42] Y. T. Zhou, R. Chellappa, A. Vaid, and B. K. Jenkins, "Image restoration using a neural network," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 7, pp. 1141–1151, 1988.

[43] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[45] D. P. Kingma and J. L. Ba, "Adam: a method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, 2014.

Hindawi
Security and Communication Networks
Volume 2019, Article ID 2674684, 8 pages
https://doi.org/10.1155/2019/2674684

Research Article

A Feature Extraction Method of Hybrid Gram for Malicious Behavior Based on Machine Learning

Yuntao Zhao,[1,2] **Bo Bo**,[1] **Yongxin Feng**,[1] **ChunYu Xu**,[1] **and Bo Yu**[1]

[1]*School of Information Science and Engineering, Shenyang Ligong University, Shenyang 110159, China*
[2]*College of Information Science and Engineering, Northeastern University, Shenyang 110819, China*

Correspondence should be addressed to Yongxin Feng; fengyongxin@263.net

Received 12 October 2018; Accepted 20 January 2019; Published 4 February 2019

Guest Editor: Jiageng Chen

With explosive growth of malware, Internet users face enormous threats from Cyberspace, known as "fifth dimensional space." Meanwhile, the continuous sophisticated metamorphism of malware such as polymorphism and obfuscation makes it more difficult to detect malicious behavior. In the paper, based on the dynamic feature analysis of malware, a novel feature extraction method of hybrid gram (H-gram) with cross entropy of continuous overlapping subsequences is proposed, which implements semantic segmentation of a sequence of API calls or instructions. The experimental results show the H-gram method can distinguish malicious behaviors and is more effective than the fixed-length n-gram in all four performance indexes of the classification algorithms such as ID3, Random Forest, AdboostM1, and Bagging.

1. Introduction

With the development of computer and Internet technology, *mal*ware soft*ware* (*malware*) such as Trojans, viruses, and worms also constantly mutates, self-renews, and becomes one of most serious threats to Cyberspace. Malware is a program that performs malicious tasks on a computer system, which implements control by changing or destroying process execution flow. In recent years, there are endless security incidents from malware. For example, in May of 2017, WannaCry [1], one of the devastating ransomware which plagued over 150 countries and traversed all continents, spared no industry niche owing to the indiscriminate nature of the attack [2]. At 7:10 PM Eastern Time in October 21, 2016, hackers manipulated millions of "broilers" to paralyze the server of the DNS supplier of Dyn through the Mirai malware in a hijacking attack, which led to the collapse of well-known American websites, including Twitter, Paypal, Spotify, Netflix, Airbnb, Github, Reddit, and New York Times, and half of the United States fell into a disconnected state. From these events, we can see that malware detection is extremely urgent.

At present, malware detection methods are divided into static detection and dynamic detection [3–5]. Based on the static method, features are extracted from the original codes or files such as PE files, binary code, and disassembled code. Without running, the static features are represented as a series of coding that is the only identity representation and can distinguish from other software. For example, antivirus software products (e.g., Symantec and Kaspersky) mainly use the signature-based method of detection [6, 7], which is unique for known malware so that its samples can be correctly classified with a small error rate [8]. But the static method is usually vulnerable to obfuscation technology. For example, hacker makes tiny changes in malware variants, such as adding null instruction (NOP) in noncritical areas, changing instruction jump mode, which will produce new signature-based codes. If the virus library is not updated and preserves the new codes, the antivirus software will not be able to detect these variants. Different from the static method, the dynamic features are extracted under runtime environment (e.g., sandbox or honeypot), where malicious acts and operations are not hidden or discounted. For example, the dynamic representations of malicious behaviors may be from a sequence of API calls or instructions in a virtual machine environment. Compared with static method, the dynamic one does not need reverse engineering such as decompilation and decryption. Though it consumes a lot of running time and storage space, the

dynamic method is more resilient to obfuscation technology [9, 10].

As a subset of AI (artificial intelligence), machine learning, developed to allow robots and computers to learn autonomously, is used to better serve the corresponding business needs and has achieved great success. Through the approximation of complex function and nonlinear fitting, machine learning, such as SVM, Naive Bayes, and decision trees, has been used for model to detect malicious codes [7, 11]. Ye [12] and others use a sequence of API calls as the behavioral characteristic of malware and develop the Intelligent Malware Detection System (IMDS) scanning malware based machine learning. Li et al. [13] propose an approach for extracting the dynamic features of malicious code semantics. The method extracts the dynamic features of malicious codes in virtual environment so as to achieve the purpose of protecting physical machine. The experimental analysis adopts the machine learning methods such as decision trees, KNN, and SVM. Zhou et al. [14], based on the isomorphism of sensitive API call graph, propose a method which is used to construct malware family features with graph similarity metric.

In this paper, we propose a features extraction method of hybrid gram for malicious behavior with cross entropy based on machine learning. Inspired from semantic segmentation of NLP (Natural Language Processing), the API sequences of malware are essentially kinds of context and are as rigorous as NLP on semantic and structural features. Therefore, a sequence of high correlation API can be used to represent the malware behavior. Moreover, in order to select the malware features, the feature vector is extracted from the sequences of API calls by sliding time window of different gram. Considering that the vectors are very sparse and high dimensional at this time, nonsignificant features should be excluded. Furthermore, through calculating cross entropy of continuous overlapping subsequences in hybrid gram, the new feature vector is chronologically integrated and selected. Finally, the machine learning method is applied to classify and detect malware samples. We adopt the dataset from VXHeavens website [15] to train the model. The experiment results show that H-gram method effectively distinguishes malicious behaviors.

The rest of the paper is organized as follows. In Section 1 the API-related knowledge is introduced, as well as dynamic track and capture in a virtualized environment. Section 2 elaborates the process of extracting and quantifying n-gram semantic features and also an adaptive variable-length n-gram feature selection method. In Section 3, we describe the process of converting the API sequences processed by n-gram into information entropy, namely, the feature quantization. In Section 4, we use the machine learning method to verify the experimental data analysis. Finally, the full paper is summarized.

2. Win32 API Call Mechanism and Feature Selection

In the Windows operating system, user applications rely on interfaces provided by dynamic link libraries such as kernel32.dll, advapi32.dll, user32.dll, and gdi32.dll to access the hardware and system resources. Windows API call mechanism is shown in Figure 1.

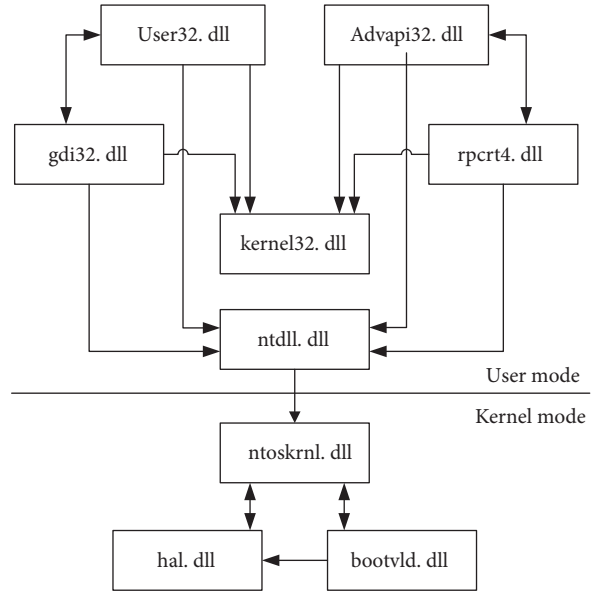


Figure 1: Windows API call mechanism.

This interface is called the Win32 API. For example, when a user program calls the Win32 API function of reading file, the process jumps to the NtReadFile function in the kernel state entry "ntdll.dll" firstly. Then the NtReadFile function calls the service routine in kernel mode, which is also named NtReadFile. Almost every program need directly calls the native API (kernel mode). If you want to monitor the program, the best way is to directly monitor its API calls. API function itself is not divided into the malicious or the benign. In other words, the malware uses the normal API function to achieve its own malicious purpose. The same API can be called by either the malicious or the benign. Only in terms of these sequences of API calls with context information can the diversity, between the malware behaviors and the benign ones, be discriminated effectively. However, due to the large number of API functions, it is not possible to describe the behavioral characteristics of samples in the actual operation through tracking all the APIs at all time. Therefore, in the paper we use the API hook technology to dynamically capture the API call sequence generated by the samples under the virtualized environment. After analysis and summary, six kinds of malicious behavior are obtained. The six kinds of behavior of API call sequences generated by each test sample are captured in chronological order. The API feature selection process is shown in Figure 2.

3. Feature Selection Model

The n-gram model has been successfully applied to the field of text analysis, which has improved the accuracy of similarity measure between texts. The API sequences of programs are essentially kinds of text and are more rigorous than the text

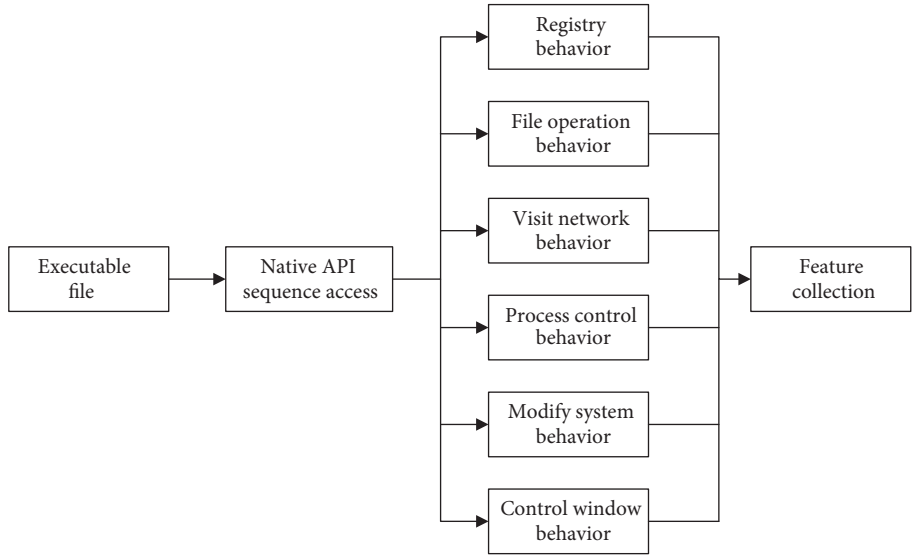


Figure 2: API feature selection process.

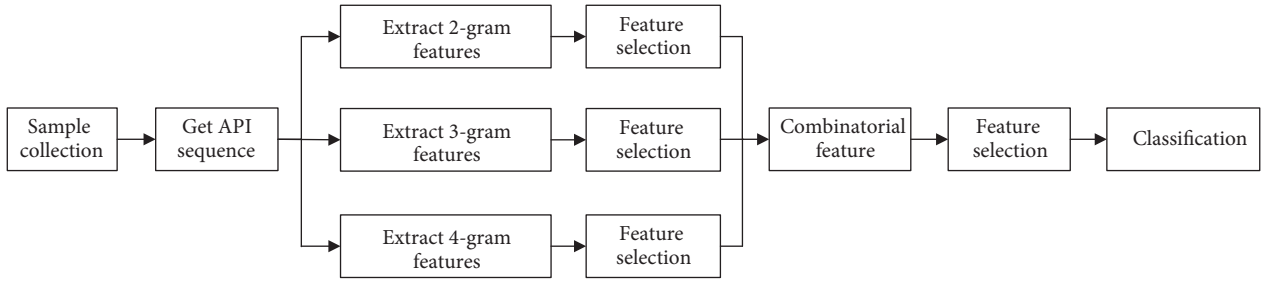


Figure 3: A model of feature selection of hybrid gram.

on semantic features and structural features. So n-gram can also be used for malware features analysis and selection. Without knowing which subsequences have representative semantic information, we extract new features from API call sequences by sliding fixed time windows. API call sequences are represented as $\{\cdots, x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}, \cdots\}$. The n-gram model, which satisfies Markov hypothesis as formula (1), can produce partial overlapped continuous subsequences.

$$P\left(x_t \mid x_{t-1}, x_{t-2}, \ldots, x_{t-N}, x_{t-N-1}, \ldots, x_{t-N-M}\right) = P\left(x_t \mid x_{t-1}, x_{t-2}, \ldots, x_{t-N}\right) \quad (1)$$

For the example of 2-gram, the formula can be expressed as the following.

$$P\left(x_t \mid x_{t-1}, x_{t-2}, \ldots, x_{t-N}\right) = P\left(x_t \mid x_{t-1}\right) \quad (2)$$

For the example of 3-gram, the formula can be expressed as the following.

$$P\left(x_t \mid x_{t-1}, x_{t-2}, \ldots, x_{t-N}\right) = P\left(x_t \mid x_{t-1}, x_{t-2}\right) \quad (3)$$

For the API call sequence, if the semantic segmentation is performed by 3-gram as an example $\{x_1, x_2, x_3, \cdots, x_n\}$, a short sequence of consecutive partially overlapping sequences is generated as the following.

$$\left\{\left(x_1, x_2, x_3\right), \left(x_2, x_3, x_4\right), \ldots, \left(x_{n-2}, x_{n-1}, x_n\right)\right\} \quad (4)$$

From the above the formula (4), a new sequence set is as follows, where $s_1 = (x_1, x_2, x_3), s_2 = (x_2, x_3, x_4), \ldots, s_{n-2} = (x_{n-2}, x_{n-1}, x_n)$.

$$S_{3\text{-gram}} = \left\{s_1, s_2, \ldots, s_i, \ldots, s_{n-2}\right\} \quad (5)$$

The set element of s_i represents a short sequence of 3-gram (x_i, x_{i+1}, x_{i+2}). Among them, it is relatively easy to extract n-gram segmentation items, but the semantics features of n-gram segmentation are not as obvious as those of the real code. So the amount of the length of the sliding time window of n-gram is a very important issue. A small value would ignore the structure and order of the process context, and an oversize value would reduce the similarity between the calls. Therefore, we use the same procedure to experiment with a hybrid n-gram to preserve as much semantic information as possible.

A model of feature selection of hybrid n-gram is proposed and shown in Figure 3. First of all, we collect representative samples of malware and benign software and put each sample

into the analysis environment for a period of time and then record the dynamic API sequence of each. Secondly, we extract the semantic features with a variable N value of n-gram from API call sequence of each sample and generate feature segments such as 2-, 3-, and 4-gram and so on. The weight value of each short characteristic sequence can be represented by information entropy.

As the feature dimension of the API sequence is very huge, the selection method is used to reduce the dimension of features and extract the valid features. Due to dimension reduction, some semantic information is lost, while some semantic information is retained. By combining the features selected by hybrid n-gram, the complementarity between feature types is achieved, and more semantic information is retained as much as possible. After hybrid n-gram, the number of features is still large. As some features are redundant, the feature selection method is used to establish the effective feature subset and then do it again. After the above process, we can get the effective features that can distinguish between the malware and the benign software well. Finally, we use a variety of classification algorithms to detect the malware and verify the validity of the method of feature selection. A model of feature selection of hybrid n-gram is shown in Figure 3.

4. Feature Selection Method Based on Joint Entropy

The most obvious behavior difference between malware and normal software is that the malware needs to accomplish its own illegal goal, such as the realization of hidden, self-deleting, unrecognized payload and so on. These behavior features are not in the normal software. Therefore, different API sequence features have different information entropy. It happens that the entropy will change when the malware is illegally performed on the host computer.

In the paper, a novel feature selection of hybrid n-gram with cross entropy is proposed. Two variables C_i and s_j are set. C_i represents the number of the category that this behavior belongs to. s_j represents the j^{th} short sequence of n-gram formed from formula (4)~(5). Firstly, the behaviors of API calls are converted to the sequence of n-gram such as $\{s_1, s_2, \ldots, s_j\}$. In same category the occurrence number of each short sequence of n-gram is counted, which is expressed as $a_{i,j}^{(k)}$. The sign k stands for the category label. The sign i stands for the number of samples. The sign j stands for the serial number of each short feature sequence of n-gram. The relationship of category and feature sequence of samples is shown as Table 1.

The purpose of feature selection is to obtain a kind of characteristics with the ability to classify data, which can improve the efficiency of classification learning. If there is no obvious difference between the results of classification with one feature and the results of random classification, this feature has no classification ability. Discarding the feature will not affect the classification accuracy. In the process of identifying the malware, the cross entropy of API calls is used to extract important features.

```
1 for n=1:N; %the value of n with n-gram
2    calculate H(D);
3    for j=1:M
4       calculate H(D | s_j)
5       ga(s_j)=calculate g(D, s_j)
6    end
7    for k=1:K
8       s_max(n,k)=max{ga(s_1), ga(s_2),...,ga(s_M)}
9       delete s_max(n,k) from{ ga(s_1), ga(s_2),...,ga(s_M)}
10   end
11 end
12 establish feature selection set { s_max(n, k)}
```

Algorithm 1: The algorithm procedure.

Let D be the train data set. |D| represents sample capacity, namely, the amount of total samples. There are K kinds of API behaviors from C_1 to C_k, and $i = 1, 2, \cdots K$. $|C_i|$ is the amount of samples that belong to the C_i category. So $\sum_i^K |C_i| = |D|$. The feature set of API sequences of n-gram is represented as S_{n-gram}.

$$S_{n-gram} = \{s_1, s_2, \ldots, s_j\} \tag{6}$$

Firstly, the empirical entropy of the data set D is calculated.

$$H(D) = -\sum_{i=1}^{K} \frac{|C_i|}{|D|} \log_2 \frac{|C_i|}{|D|} \tag{7}$$

Secondly, the empirical conditional entropy of the characteristic s_j for data set D is expressed as the following formula.

$$H(D \mid s_j) = -\sum_{i=1}^{K} p(C_i, s_j) \log_2 p(C_i \mid s_j) \tag{8}$$

In the above formula, $p(C_k, s_j)$ is expressed as the following.

$$p(C_k, s_j) = \frac{\sum_i^{|C_k|} a_{i,j}^{(k)}}{\sum_{i,j,k} a_{i,j}^{(k)}} \tag{9}$$

Also $p(C_i \mid s_j)$ is expressed as the following.

$$p(C_k \mid s_j) = \frac{\sum_i^{|C_k|} a_{i,j}^{(k)}}{\sum_{i,k} a_{i,j}^{(k)}} \tag{10}$$

Thirdly, the cross entropy of the characteristic s_j for data set D can be calculated.

$$g(D, s_j) = H(D) - H(D \mid s_j) \tag{11}$$

The top (maximal) M of $g(D, s_j)$ is selected. Then the feature set of S_{n-gram}:$\{s_1, s_2, \cdots, s_M\}$ is extracted, whose elements match the top M of $g(D, s_j)$ one by one. Finally, the total data set of feature selection such as $\{S_{2-gram}, S_{3-gram}, \cdots, S_{n-gram}\}$ is established. The key procedure with the hybrid n-gram algorithm is shown as Algorithm 1.

Table 1: The relationship of category and feature sequence of samples.

sample	category	s_1	s_2	… …	s_j
Sample-c1-1	C_1	$a_{1,1}^{(1)}$	$a_{1,2}^{(1)}$	… …	$a_{1,j}^{(1)}$
Sample-c1-2		$a_{2,1}^{(1)}$	$a_{2,2}^{(1)}$		$a_{2,j}^{(1)}$
… …		…	…		…
Sample-c1-|C_1|		$a_{\|C_1\|,1}^{(1)}$	$a_{\|C_1\|,2}^{(1)}$		$a_{\|C_1\|,j}^{(1)}$
Sample-c2-1	C2	$a_{1,1}^{(2)}$	$a_{1,2}^{(2)}$	… …	$a_{1,j}^{(2)}$
Sample-c2- 2		$a_{2,1}^{(2)}$	$a_{2,2}^{(2)}$		$a_{2,j}^{(2)}$
… …		…	…		…
Sample-c2-|C2|		$a_{\|C_2\|,1}^{(2)}$	$a_{\|C_2\|,2}^{(2)}$		$a_{\|C_2\|,j}^{(2)}$
… …	… …	… …	… …	… …	… …
Sample-ci-1	C_i	$a_{1,1}^{(i)}$	$a_{1,2}^{(i)}$	… …	$a_{1,j}^{(i)}$
Sample-ci-2		$a_{2,1}^{(i)}$	$a_{2,2}^{(i)}$		$a_{2,j}^{(i)}$
… …		…	…		…
Sample-ci-|Ci|		$a_{\|C_i\|,1}^{(i)}$	$a_{\|C_i\|,2}^{(i)}$		$a_{\|C_i\|,j}^{(i)}$

Table 2: Malware categories and quantity.

Malware category	Backdoor	P2P-Warm	Warm	Trojan	Virus	Total
Amount	138	56	72	162	158	586

5. Experimental Analysis

5.1. Evaluation Indicators. In order to evaluate the method of the hybrid n-gram, three main indicators are used to measure the performance of the classifier through different feature extracted, including true positive rate, false positive rate, and the accuracy. The performance of the classifier can also be evaluated by using a ROC (receiver operating characteristic) curve, whose vertical axis represents true positive rate and whose horizontal axis is false positive rate. The area under the ROC curve (AUC) is a more comprehensive index for evaluating the classifier. The value of AUC is usually between 0.5 and 1.0. A larger value of AUC generally indicates that the performance of the classifier is better.

5.2. Experimental Data Acquisition. We collected 932 experimental samples. The benign software samples are collected from the Windows XP system directory and PE format EXE files, including different types of software, such as graphics software, system tools, multimedia software, and office software, whose amount is a total of 346. All benign software samples are detected by 360 anti-virus software. Malware samples were collected from the VXHeavens website [15], a total of 586, including viruses, worms, Trojans, and backdoor programs. The distribution of malware is shown in Table 2.

The Windows XP system is installed in the VMware virtual machine and then run the samples. API Monitor is taken as a hook routine to capture the native API sequences that the samples constantly call during execution. The sample run time is 120 seconds. For the vast majority of malware programs, 120 seconds is enough for them to execute all the processes, including a large number of cycles calling. Each sample intercepts the top 1000 API sequences as extraction feature. API sequences are recorded for each sample and not directly processed with the machine learning algorithm. In the paper, we select 100 feature sequences with the highest cross entropy of the fixed-length (N = 2, 3, 4) n-gram short sequences, respectively.

The Integrated 300 features after the initial selection are still too much, which is high dimension for the classification. The feature selection algorithm needs to select the most relevant subset of the features. In this paper, the features generated by hybrid n-gram (N=2, 3, 4) model are adopted. By features reduction of APIs segmentation by hybrid n-gram, the features of short sequences of different lengths are obtained. The combined number of features is 154 that is still more for the classification learning. Further, by adjusting the threshold value of cross entropy of the feature selection method again, we get 28 features that will eventually be used for categorical learning. After the above process, the features of low dimension with effectively distinguishing between malware and benign software are obtained.

5.3. Experimental Results Analysis. The final features that are used as the input of the classification algorithm and model are obtained. The experiment adopts four kinds of machine learning classification algorithms including ID3, Random Forest, AdboostM1, and Bagging. All four algorithms use an implementation version of the open source machine learning platform WEKA [16].

In the paper, we use 10-fold cross-check experiment method and apply the above four classification algorithms. The comparison results of the novel proposed method and other methods are shown in Table 3. We also use the above four classification algorithms to test the selected features of API call sequence.

Table 3: Malware detection experiments based on feature selection.

Feature representation	Features Quantity	Classification algorithm	TPR /%	FPR /%	Accuracy /%	AUC
2-gram	36	ID3	85.9	14.3	87.5	0.863
		Random Forest	86.3	12.8	86.6	0.850
		AdboostM1	82.9	16.3	80.2	0.808
		Bagging	83.9	15.8	82.3	0.826
3-gram	53	ID3	86.3	15.7	85.0	0.841
		Random Forest	94.1	13.7	92.0	0.971
		AdboostM1	91.2	14.7	93.0	0.956
		Bagging	91.2	12.8	87.5	0.931
4-gram	65	ID3	87.9	14.3	95.8	0.868
		Random Forest	96.8	6.2	93.1	0.98
		AdboostM1	90.9	9.1	87.5	0.974
		Bagging	93.9	7.0	92.0	0.957
Hybrid n-gram with cross entropy	28	ID3	96.8	6.3	92.5	0.963
		Random Forest	97.8	5.1	96.8	0.983
		AdboostM1	97.8	5.1	96.8	0.983
		Bagging	97.6	5.2	96.8	0.897

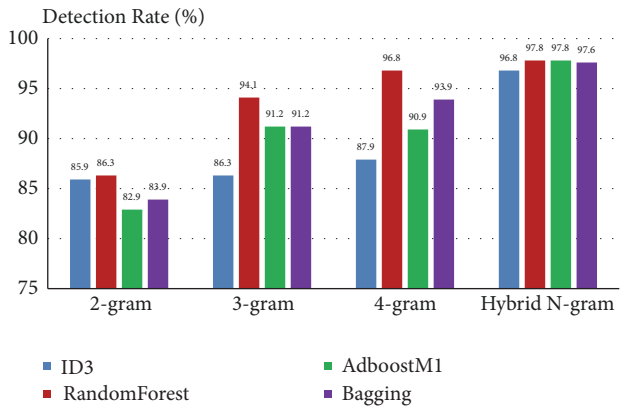


Figure 4: Comparison of detection rate in four methods.

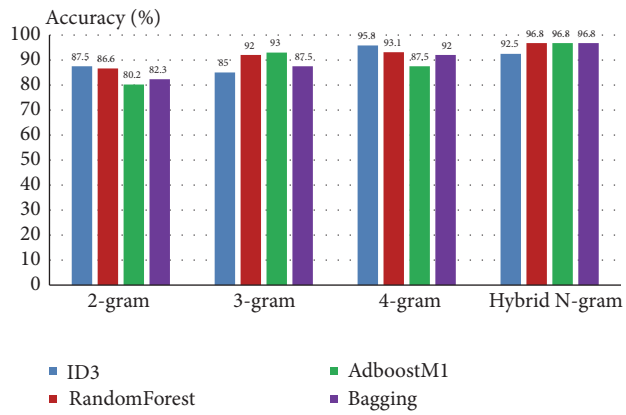


Figure 5: Comparison of accuracy in four methods.

In order to compare the effectiveness and generalization of feature selection method of API call sequence such as 2-gram, 3-gram, 4-garm, and hybrid gram with cross entropy, the experimental chooses four evaluation indexes, including detection rate, accuracy, false positive rate, and AUC value. Comparison of detection rate is shown in Figure 4. Comparison of accuracy is shown in Figure 5. Comparisons of false positive rate and AUC value are shown in Figures 6 and 7.

As can be seen from Figure 4, test results of 2-gram are less than the other three results, which illustrate that the extracted feature sequence is not obvious enough and leads to a low degree of discrimination. Of the four classification algorithm methods, Random Forest detection performs the best. The experimental results showed that the detection rate of the proposed method of H-gram is higher than that of the other three methods.

As can be seen from the comparison of accuracy of Figure 5, the overall accuracy rate is on the rise along the direction of the horizontal axis. The overall gap is small. The accuracy of 4-gram and H-gram is higher than that of 2-gram and 3-gram feature extraction. The H-gram method is slightly higher than the 4-gram method.

As can be seen from the comparison of false positive rates in Figure 6, 2-gram remains the weaker feature, with the highest false positive rate reaching 16.3%. The 4-gram and the proposed method of H-gram have achieved a relatively low false positive rate of 6.2% and 5.1%, respectively, with Random Forest algorithm. The method of H-gram has achieved the lowest false positive rate 5.1%. The false positive rate drop is more obvious with H-gram.

It can be seen from the comparison of AUC values in Figure 7 that the AUC value of the 2-gram is still lower than the other three feature selection methods. The difference of

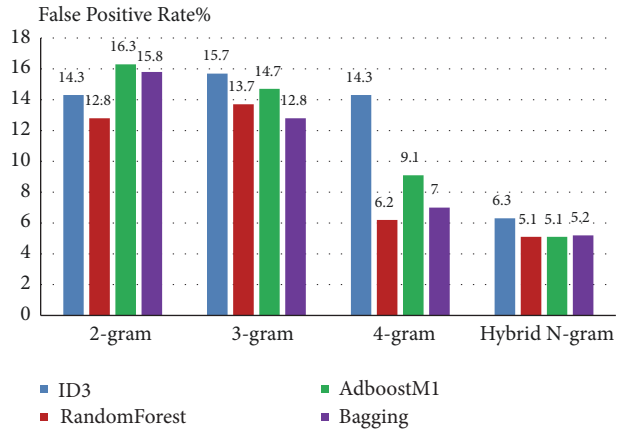


Figure 6: Comparison of false positive rate in four methods.

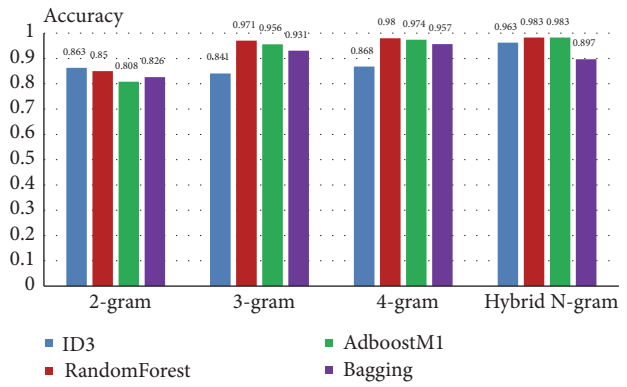


Figure 7: Comparison of AUC value in four methods.

the AUC values for other three methods is similar, and the method of H-gram is slightly better than the other three selection methods. The AUC value is up to 0.983, which is close to the optimal AUC value of 1.

6. Conclusion

As cyberspace becomes the fifth dimension of human life, the network security is getting more and more attention. Dynamic analysis method of malware behavior has become the focus of research. The past analysis methods are mainly by capturing all API functions of malware running, which describe the malware behavior with semantic segmentation of fixed parameters. So the previous methods are not only large amount of information, but also high redundancy. In the paper a novel feature selection method of hybrid H-gram with joint cross entropy is proposed. Based on the dynamic behavior tracking and feature analysis of native API in virtual environment, the proposed method is of adaptive variable length n-gram. At the same time the joint cross entropy is introduced to select the features of API sequence. Compared with the results of the experiments on semantic short sequence of the fixed-length n-gram, the proposed method is more effective in all four performance indexes of four classification algorithms (ID3, Random Forest, AdboostM1, and Bagging).

Data Availability

The dataset is real and available from VXHeaven website as mentioned in reference [15].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Authors' Contributions

Yuntao Zhao is the main contributor of this work, given that he originated the idea, provided the general design, and wrote most of the paper. Bo Bo and others contributed to the implementation and testing of the experiment. All authors read and approved the final manuscript.

Acknowledgments

This work was supported by China Postdoctoral Science Foundation (2016M590234), General Project of Liaoning Provincial Department of Education (LG201611), Postdoctoral Fund of Shenyang Ligong University, Project of Applied Basic Research of Shenyang (18-013-0-32), 2017 Distinguished Professor Project, Liaoning Nature Science Foundation (20180551066), and Program for Liaoning Distinguished Professor.

References

[1] A. McNeil, "How did the WannaCry ransomworm spread?," 2017, https://blog.malwarebytes.com/cybercrime/2017/05/how-did-wannacry-ransomworm-spread/amp/.

[2] T. Webb and S. Dayal, "Building the wall: Addressing cybersecurity risks in medical devices in the U.S.A. and Australia," *Computer Law and Security Review*, vol. 33, no. 4, pp. 559–563, 2017.

[3] D. Tumer, S. Entwisle, M. Fossi et al., *Symantec Internet Security Thread Report*, Symantec Corporation, 2014.

[4] C. Xu, *Research on The Automatic Classification Method Based on The Behaviors of The Malicious Software*, Xiangtan University, 2014.

[5] R. Jing, *Research And Implementation of Malicious Code Detection System*, University of Electronic Science and Technology of China, 2010.

[6] J. Huang and A. L. Swindlehurst, "Secure communications via cooperative jamming in two-hop relay systems," in *Proceedings of IEEE Conference on Communications Society*, pp. 524–528, 2010.

[7] R. Wang, D. G. Feng, Y. Yang et al., "Semantic based behavior feature extraction and detection method for malicious code," *Journal of Software*, vol. 23, no. 2, pp. 378–393, 2012.

[8] D. Kirat, G. Vigna, and C. Kruegel, "Bare cloud: bare-metal analysis-based evasive malware detection," in *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[9] X. Liu, *Research on Malware Analysis and intelligent technology detection based on machine learning*, Xiangtan University, 2014.

[10] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers, "Malware target recognition via static heuristics," *Computers & Security*, vol. 31, no. 1, pp. 137–147, 2012.

[11] D. W. Chen, P. Tang, and S. T. Zhou, "Malware detection model based on sandbox," *Computer Science*, 2012.

[12] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent PE-malware detection system based on association mining," *Journal of Computer Virology and Hacking Techniques*, vol. 4, no. 4, pp. 323–334, 2008.

[13] M. Li, X. Q. Jia, R. Wang et al., "A method of feature selection and modeling for malicious code," *Computer Application And Software*, no. 8, pp. 266–271, 2015.

[14] H. Zhou, W. Zhang, F. Wei, and Y. Chen, "Analysis of android malware family characteristic based on isomorphism of sensitive API call graph," in *Proceedings of the 2nd IEEE International Conference on Data Science in Cyberspace, DSC 2017*, pp. 319–327, Guangdong, China, June 2017.

[15] "VXHeaven," 2018, http://vxheaven.org/.

[16] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann Publishers Inc., San Francisco, Calif, USA, 2011.

Hindawi
Security and Communication Networks
Volume 2019, Article ID 9595081, 11 pages
https://doi.org/10.1155/2019/9595081

Research Article

Identifying Known and Unknown Mobile Application Traffic Using a Multilevel Classifier

Shuang Zhao, Shuhui Chen, Yipin Sun, Zhiping Cai, and Jinshu Su

College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China

Correspondence should be addressed to Shuhui Chen; shchen@nudt.edu.cn

Received 24 September 2018; Accepted 3 December 2018; Published 1 January 2019

Guest Editor: Chunhua Su

Due to the proliferation of mobile applications, mobile traffic identification plays a crucial role in understanding the network traffic. However, the pervasive unconcerned apps and the emerging apps pose great challenges to the mobile traffic identification method based on supervised machine learning, since such method merely identifies and discriminates several apps of interest. In this paper we propose a three-layer classifier using machine learning to identify mobile traffic in open-world settings. The proposed method has the capability of identifying traffic generated by unconcerned apps and zero-day apps; thus it can be applied in the real world. A self-collected dataset that contains 160 apps is used to validate the proposed method. The experimental results show that our classifier achieves over 98% precision and produces a much smaller number of false positives than that of the state of the art.

1. Introduction

Mobile apps are now the most popular way to access the Internet. Smart Insights [1] reports that mobile devices dominate in minutes spent online across countries and more than 80% of mobile minutes are spent on apps. According to Statista [2, 3], as of the first quarter of 2018, Android users were able to choose between 3.8 million apps in Google Play and an average of 6,140 mobile apps were released through the Google Play Store every day. BrightEdge [4] reports that 57% of all online traffic was on mobile and tablet in 2017. Hence the current focus of research is shifting from the traditional workstation traffic identification to the mobile traffic identification, which is the task of associating network traffic with a certain app in this paper.

Mobile traffic identification plays an important role in network management, marketing research, and user characteristic analysis [5, 6]. For example, based on this technology, a network administrator can obtain the popular apps in the network and optimize the resource allocation accordingly to improve the user experience. A company can monitor whether employees use unallowed apps during working hours, such as game and shopping. For advertisers, understanding a certain app is popular with users in which area and time periods can help them create a better advertising strategy. For market researchers, understanding the use of apps of concerned users can help them analyze the interests and needs of users; then further business activities can be carried out. For example, if a person uses a flight booking app frequently, then the user may be a potential customer of travel services.

Although mobile traffic identification task looks similar to the traditional workstation traffic identification task, the particularities of mobile traffic pose great challenges for traditional identification methods. First, mobile traffic is almost carried over HTTP/HTTPS, making the port-based approach identify mobile traffic as Web only. Second, lots of apps use encryption protocols for data transmission in order to protect user privacy. Indeed, some encryption protocols may expose useful information during its negotiation process, such as the TLS SNI (Server Name Indication), so that part of the encrypted traffic can be identified by DPI (Deep Packet Inspection) approach. However, the SNI field sometimes is blank and not every SSL/TLS connection has a negotiation phase, which decrease the effectiveness of this method. For example, we randomly checked 50 HTTPS connections that come from MOMO, which is a social app in China. There are 9 connections that do not have a negotiation process and 11

connections have a negotiation process but the SNI field is blank. Third, mobile apps often access third-party libraries, resulting in the fact that different apps would generate similar traffic. It is difficult to discriminate such traffic via DPI technology or IP address. This problem can be circumvented in a sense if such traffic is considered as an individual category. Fourth, CDN (Content Delivery Network) is used by many apps to improve the user experience. As a result, a server's IP address can be shared by multiple apps. For example, we found an IP address 101.226.220.12 serving at least five apps at the same time. Additionally, there are apps that do not use DNS to obtain the IP address of the server. For example, WeChat, a popular instant messaging app in China, is observed to return a list of hundreds of server IP addresses to a particular request from the client; thus the client no longer needs to perform a DNS query. The above scenarios reduce the traffic volume that can be identified by DNS-based approach. In view of the above reasons, the traditional traffic identification methods are insufficient for handling mobile traffic.

The statistical-based method has recently gained extensive research. It uses raw traffic data or side-channel information leaked from network traffic to train classifiers based on machine learning. Many methods have been proposed and the results are encouraging. However, it is impossible to identify all apps traffic due to the large number of mobile apps; thus a classifier usually merely identifies several apps of interest. Then the massive unknown traffic that comes from unconcerned apps and the emerging apps (also called as zero-day apps in this paper) brings great challenges to the classifier.

In general, there are two ways to enable a classifier to handle unknown instance in machine learning. One is constructing a N+1-class classifier, and the other is achieving multiclass classification by multiple binary classifiers. The N+1-class classifier treats the unknown instances as one category. A major drawback of this method is the training set is always insufficient since it is not possible to collect all unknown instances. The latter way learns each known category's patterns separately by training a binary classifier. Only when the predictions from all binary classifier are negative will the instance be classified as unknown. The first drawback of this method is the same as that of the former method. Another drawback is that the prediction criterion is prone to identifying unknown instances incorrectly.

In this paper, we propose a three-layer classifier to identify mobile traffic under open-world settings. This classifier possesses the capability of excluding unknown apps traffic even if the training set is insufficient. The first layer does a coarse-grained classification to exclude unconcerned apps traffic whose patterns have been learned. Then the second layer does a fine-grained classification to discriminate between target apps traffic. Finally, the third layer learns the patterns of each target app traffic from different perspectives and sets a strict prediction criterion to exclude the false positives caused by unknown traffic. Besides, we only use side-channel traffic information and raw traffic data as traffic features. To the best of our knowledge, this is the first time to identify mobile traffic in open-world settings which contain unknown app traffic.

The main contributions of our work are as follows. Firstly, we propose a novel multilayer classifier that can identify target app traffic and exclude unknown app traffic. This approach can be applied to the identification of mobile traffic in the real world. Secondly, we collect a representative mobile traffic dataset to validate our method. This dataset contains network traffic from 160 apps that are installed on 12 mobile devices. Finally, our method outperforms the state of the art. The results show that the proposed classifier achieves more than 98% precision with the lowest number of false positives.

The rest of the paper is organized as follows: Section 2 surveys related work; Section 3 describes the proposed multilayer classifier architecture; Section 4 evaluates the proposed method; Section 5 gives a brief discussion; Section 6 concludes the paper.

2. Related Work

For the reasons explained above, there are some deficiencies in port-based, DPI-based, and DNS-based approaches when they are applied to mobile traffic identification. Please refer to work [7] for a detailed survey about using DPI-based methods to identify mobile traffic. Here we highlight several DNS-based and machine learning-based traffic identification methods that have recently been proposed.

2.1. DNS-Based Traffic Identification. Bermudze et al. [8] presented a notable work that associates network traffic with domain names. They extracted the 3-tuple (i.e., $\langle ClientIP, ServerIP, Domain\rangle$) by parsing the captured DNS packets and labelled traffic with domain name according to $\langle ClientIP, ServerIP\rangle$. Then offline analysis was performed based on the label of traffic, including the distribution of domain names and server IP addresses and the domain names or service offered by certain CDN vendor. The authors pointed out that about 73% of server IP addresses have a unique domain name and 82% of domain names have a single IP address. Similar mechanisms were proposed in work [9, 10]. In addition, Mori et al. [10] enriched the tuple library by combining the DNS response information of multiple users and ignoring the TTL, thus increasing the identifiable traffic volume. However, the above work only maps the traffic to the corresponding domain name without further identifying its related apps.

Trevisan et al. [11] investigated the effectiveness of DNS-based traffic identification method. They showed that about 65% of server IP addresses have a unique domain name, but less than 15% of the traffic is owing to these addresses. By manually mapping domain names to services, up to 55% of the traffic can be identified. The authors further explored how the associations between domain name and IP addresses evolve over time. The authors discovered that some IP addresses would become invalid over time. Therefore, although the DNS-based traffic identification method is simple and straightforward, only a small portion of mobile traffic can be handled.

2.2. Machine Learning-Based Traffic Identification. Wang et al. [12] proposed a system for identifying mobile apps. They

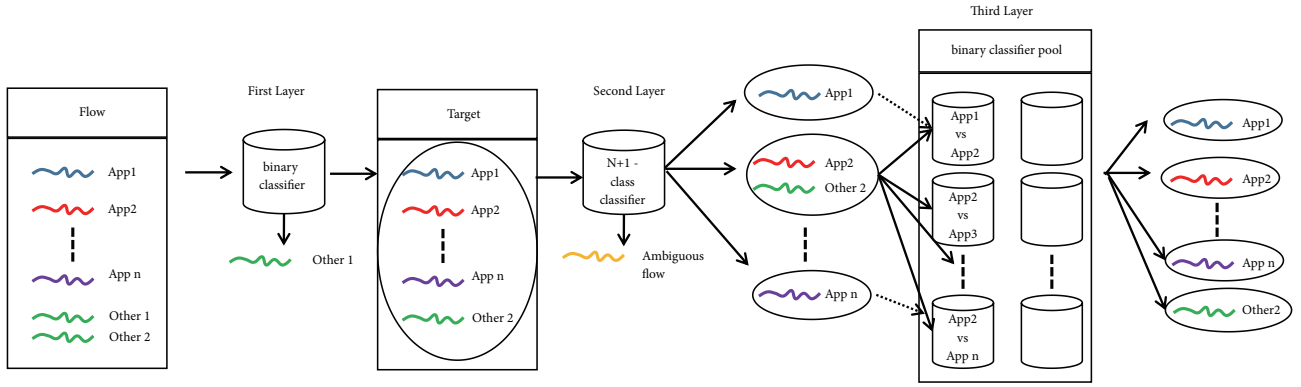


Figure 1: A visualization of the three-layer classifier.

collected traffic from 13 target apps by running apps dynamically for 5 minutes and then a Random Forest classifier was trained. Since the sample size in this work is inadequate, it is difficult to assess whether the results of this work are representative.

Alan et al. [13] identified thousands of apps using the launch-time traffic generated by target apps. The results showed that the classification accuracy reached 88% when training and test sets are collected on the same device; otherwise the accuracy would drop by as much as 26%.

AppScanner [14, 15] proposed a scheme for fingerprinting and identifying apps. They collected network traffic generated by different versions of apps installed on two Android devices. Then Support Vector Classifiers and Random Forest Classifiers were trained to classify 110 apps. Bursts of data are considered in this work to extract statistical features. Additionally, they improved the performance of the classifier by detecting "ambiguous flows". Moreover, AppScanner also used a postvalidation mechanism to reject samples with low prediction credibility. The experimental results reported 96% average accuracy in the best case with recall lower than 40%. This work suffers from the fact that the burst is used to model traffic; thus AppScanner is only feasible in the simple network, such as a network which contains a single mobile device. It cannot be applied to the high-speed backbone network because it is likely that bursts cannot be extracted.

Some approaches based on CNN (Convolutional Neural Networks) also give a notable performance. For example, Chen et al. [16] encoded HTTP plaintext requests and identified 20 apps using 2D-CNN, but this technique only works with unencrypted traffic. Wang et al. [17] proposed a classifier for identifying malicious traffic based on 2D-CNN. The raw traffic data is converted into 2D-vectors as the input of the classifier. Their another work [18] held the view that the traffic is essentially sequential data, so 1D-CNN model is used to identify traffic. The accuracy reached 86.6% when network traffic is identified in fine-grained classification. They showed that the classifier can achieve better performance when using raw traffic data from all protocol layers than using payload only. However, as pointed out by Giuseppe et al. [19], the data provided in this case is always in the form of PCAP files, containing information that could introduce a bias in the classification results. Deep Packet [20] proposed a similar mechanism to identify mobile traffic using a 1D-CNN and a Stacked AutoEncoder (SAE). Giuseppe et al. compared four NN-based traffic identification methods and [18] gives the best performance when identifying the traffic generated by Android apps.

Although the aforementioned studies have proven to be effective, the proposed methods do not take into account the impact of unknown traffic on the classifier, thus impeding their application in the real-world networks.

3. Methodology

To handle the real-world mobile traffic identification task, a classifier needs to meet two requirements. One is identifying the target app traffic correctly and the other is eliminating a large amount of unknown traffic even if the unknown traffic training set is insufficient. Based on these two requirements, we present a three-layer classifier and the architecture is depicted in Figure 1.

We first introduce the terms defined in this paper. Bidirectional flow, which is a set of packets carrying the same 5-tuple (i.e., $\langle SrcIP, DstIP, SrcPort, DstPort, Protocol\rangle$), is used to decompose the captured traffic into discrete units. Flow is used to represent bidirectional flow in the rest of the paper when it does not cause ambiguity. For a TCP connection, SYN and RST/FIN indicate the beginning and end of the flow, respectively. A timeout mechanism (90s) is used to determine the end of a flow when a termination is not observed. Since mobile apps use mostly HTTP/HTTPS, only TCP flows are considered in this paper. But the proposed method can be ported to work with UDP traffic without any changes. Target represents the traffic that comes from the apps of interest (also called target apps in rest of the paper). Appi represents the i-th target app. The unknown traffic that comes from the unconcerned apps and zero-day apps is defined as Other category. Inspired by AppScanner, "ambiguous flows", that is, traffic that is common among more than one app, is also used in our method. A new ambiguity detection method, which will be described in detail later in this paper, is designed to extract ambiguous flows.

3.1. Coarse-Grained Classification. The first layer of the architecture is a coarse-grained binary classifier that identifies flows as Target or Other. The binary classifier is not attempting to discriminate the Target from the Other accurately, which is also unrealistic because the classifier cannot be trained with the universe unknown flows. Although unknown instances are insufficient, the classifier can still learn some patterns of unknown traffic from the existing instances. Thus the primary purpose of this stage is to eliminate as much unknown traffic as possible without misidentifying the Target, thus reducing the incorrect classification of the followed second layer classifier. Hence this binary classifier is expected to have a high recall but may have a low precision of the Target class. This can be carried out by assigning appropriate weights to training instances.

3.2. Fine-Grained Classification. The second layer is responsible for fine-grained classification; i.e., the classifier in this layer aims to distinguish between target apps traffic. If there are N target apps, then an N+1-class classifier is trained in this layer. The N+1 classes consist of N target apps and ambiguous flows class. The flows classified as Target in the first layer will be classified by this fine-grained classifier. The possible classification results of a flow at this stage are as follows:

(1) Classified as ambiguous flow: the classifier cannot identify the flow to a target app and refuses to give an explicit label.

(2) Classified as Appi and the flow belongs to Appi: the classification produces a true positive.

(3) Classified as Appi but the flow belongs to another target app or unknown app: the classifier produces a false positive on Appi.

In close-world settings, the primary purpose of a classifier is to distinguish different target apps traffic effectively, the traffic generated by other apps is not under consideration. By contrast, in open-world settings, the unknown traffic is the main source of false positives of the classifier and it will decrease the performance of the classifier dramatically. Therefore, the third layer is designed to verify the classification results of the second layer.

3.3. False Positive Exclusion. The third layer aims to eliminate the false positives caused in previous layers, i.e., to eliminate the misclassified target traffic in the second layer and unknown traffic that is not excluded by the first layer. The involved classification categories in this stage include N target apps, Other class, and ambiguous flow class. Then $(N+2)*(N+1)/2$ binary classifiers are trained using One vs One.

If a flow is classified as Appi in the second layer, then it will be classified by N+1 binary classifiers in this layer. The N+1 binary classifiers are Appi vs Appj (j not equal to i), Appi vs Other, and Appi vs ambiguous flow. The output of this layer is Appi only when all binary classifiers classify the flow as Appi. Otherwise, this stage refuses to give a prediction. Multiple base classifiers and different traffic features can be utilized to train these binary classifiers so that mobile traffic can be portrayed from different perspectives. In short, the classifiers designed in this stage start with the assumption that if a flow belongs to Appi, it should be identified as Appi regardless of the feature or model used.

By this way, the third stage focuses on portraying each target category from multiple perspectives. Therefore, even if the patterns of unknown traffic are not learned by classifiers, the strict prediction criteria of the third layer will enable the classifier to eliminate nontarget instances effectively. Additionally, although One vs One is used to train $(N+2)*(N+1)/2$ classifiers, each flow that arrives at the third layer only needs to be classified by N+1 classifiers.

In fact, the third stage excludes the unknown traffic at the expense of the number of flows whose prediction is valid. However, we hold the same view as [15]: false positives are usually undesirable for app identification.

3.4. Classifier Implementation

3.4.1. Ambiguous Flows Extraction. An Android app is built to aid us in collecting and labelling mobile traffic. Further information on this tool is available in Section 4. However, there is some "noisy data" in the captured data. First, network traffic coming from different apps using the same third-party libraries has different app labels. Second, we do not impose any restrictions on the user behavior, but some user actions may cause flows to have wrong labels. For example, if a user clicks an Appj's link in Appi and keeps on using Appj in Appi process, the generated traffic will be labelled as Appi rather than Appj. In fact, the pattern of such traffic is in accord with that of Appj. Thus the classifier will be given contradictory training examples. We adopt the concept of "ambiguous flow" given in AppScanner to alleviate these problems.

A heuristic rule is exploited to extract ambiguous flows in this paper in contrast to AppScanner, which trained a Random Forest classifier for extracting such traffic. For network traffic coming from the same third-party library, it may have identical server IP addresses and ports. Similarly, the server IP address and port of the traffic coming from the latter case should also have some associations with the traffic generated by Appj. Based on this assumption, we extract the ambiguous flows as follows. First, the training set is grouped according to the $\langle ServerIP, Port \rangle$ pair. Then for each group, if the traffic in the group has multiple labels and there is no dominant category, that is, there is no category accounting for more than 90% of the total sample size in the group, flows in this group are relabelled as ambiguous flows.

3.4.2. Traffic Features. We designed 37 traffic features, including packet length related features, time interval related features, packet numbers, and ports. Then the correlation-based feature selection and best-first search provided in Weka [21] were used to select an effective feature set. Additionally, Bela et al. [22] showed that a P2P traffic classifier can reach a remarkable accuracy over 95% using as limited data the first 16 bytes of the first packet of each flow. Therefore, as listed in Table 1, our final feature set has 29 features including 12 statistical features, 16 byte values, and destination port. In order to classify traffic in real time, we extract all features

Table 1: Traffic feature set.

Feature	Description	Number
Payload size related	the first three non-null payload size of the packets transmitted from the client to the server; the first and third non-null payload size of the packets transmitted from the server to the client	5
Packet length related	maximum, minimum of the length of the packets transmitted from the client to the server; maximum, minimum, average, variance of the length of the packet transmitted from the server to the client; the minimum packet length of the flow	7
First 16 bytes of payload	a symbol is used to preserve the transmission direction	16
Destination port		1

Table 2: The details of each layer classifier.

	Model	Features	Output	Classifier Number
Layer 1	Random Forest	First 16 bytes of payload	binary-class	1
Layer 2	Random Forest	First 16 bytes of payload	N+1-class	1
Layer 3	Random Forest Xgboost [24]	Port, 12 statistical features	binary-class	$(N+2)*(N+1)/2$ $(N+2)*(N+1)/2$

from the first five packets with non-null payload of each flow given that Bernaille et al. [23] found that the first five packets of a TCP connection are effective for traffic classification.

3.4.3. Base Classifier. Previous work [12, 14, 15] has shown decision tree-based models have an impressive performance in identifying mobile traffic. Therefore, we use decision tree-based models as base classifiers to implement our method. The details of each layer classifier are shown in Table 2.

The training set includes 2 categories in the first layer, i.e., Target and Other, and 16 features are used to train a Random Forest classifier. Next, ambiguous flows are extracted from the training set. The training set for the Random Forest classifier in the second layer consists of the instances of the ambiguous flows class and the remaining instances of N target apps. In the third stage, in order to describe traffic from a different perspective, port and 12 statistical features are used as traffic features and two different decision tree-based classifiers are trained. The training set in this stage includes N+2 categories. The N+2 categories include the ambiguous flows class, N target app classes, and Other class. Other class contains unknown instances that are misclassified by the first layer classifier. This is because the third layer classifier has no need to learn the features of an unknown flow if the flow is excluded in the first layer. Since we use two models at this stage, the third layer contains $(N+2)*(N+1)$ classifiers in total, and a flow entering this stage needs to be classified by $2*(N+1)$ classifiers.

4. Evaluation

4.1. Dataset Collection. Since mobile traffic involves user private data, public mobile traffic dataset is not available currently. Therefore, the existing work uses the self-collected dataset to validate the proposed method. However, the commonly used data collection methods have drawbacks and poor scalability. First, mobile devices generate lots of background traffic, resulting in the fact that running one application at a time cannot get the accurate ground truth. Some measures are still needed to exclude background traffic [15]. Additionally, although there are tools, such as Network Log [25], that can be used to collect and label traffic according to the app process, these tools always require a rooted device, which results in poor scalability.

To overcome the above deficiencies, we built an Android app based on the VPNService framework provided in the Android system. This tool does not require the mobile device to be rooted. When it runs in the background, it can capture traffic and label it according to the app process that generates it. Then the captured traffic will be saved as a pcap file and the file is sent to a server every 5 minutes. In this way, our data collection has the following benefits over the data collection by manually running an application in a limited environment or using UI fuzzing technique for automatically running the apps. First, once the app is launched, the device will upload its traffic whenever the user uses it. Thus there is no need for further human intervention. Second, there are some execution paths in the app that cannot be executed by UI fuzzing, but this problem does not exist in our approach. Third, the captured traffic is generated in various network environments, making our dataset more representative. Fourth, this method is not affected by background traffic and has a good scalability. However, since other mobile operating systems do not provide similar interfaces, this tool is only available for Android devices. We are developing a function-like app for the iOS system to obtain traffic coming from iOS devices for future work.

Based on the aforementioned tool, we collected the mobile traffic generated by mobile devices of 12 users in nearly three months. Although all devices run under Android system, they come from different vendors such as HuaWei, XiaoMi, and Samsung. The final captured dataset contains

Table 3: Overview of the two datasets.

	Time of data collection	TCP flow size	Number of apps
Dataset1	2018.05.24 - 2018.08.08	129817	138
Dataset2	2018.08.09 - 2018.08.17	31567	88

Table 4: Composition of the two datasets.

	App Name	DataSet1 (Flow Size)	DataSet2 (Flow Size)
Apps of Interest	WeChat	22497	3526
	TencentVideo	11764	0
	BILIBILI	8461	0
	Sougou Pinyin	8290	783
	TaoBao	7341	0
	BaiDu Browser	6839	98
	QQ	5915	104
Other	Other	58710	27056
Total		129817	31567

network traffic from 160 apps. Besides, the traffic is generated in various network environments covering 3G, 4G, and WI-FI. The collected data is divided into two datasets as shown in Table 3.

Dataset1 is used to train and test our three-layer classifier. Seven apps with more than 5000 flows in Dataset1 are chosen as target apps in our setting; the remaining 131 apps act as unconcerned apps. Dataset2 is only used to test the classifier. It is worth mentioning that Dataset2 contains 22 apps that have not been seen in Dataset1; thus the 22 apps can be regarded as zero-day apps and they account for 5569 flows. The details of the two datasets are listed in Table 4.

4.2. Evaluation Metrics and Experimental Setup. Five evaluation metrics are used: True Positive (TP), False Positive (FP), False Negative (FN), precision, and recall. For an app A, TP refers to the number of samples correctly classified as A. FP refers to the number of samples incorrectly classified as A. FN refers to the number of samples incorrectly classified as non-A. Then the precision and recall of identifying A are TP/(TP+FP) and TP/(TP+FN), respectively.

Scikit-learn [26] machine learning library is used to implement our classifiers. We compare it with the state of the art [18] and a single Random Forest classifier is trained as the baseline. The baseline is an N+1-class Random Forest classifier which includes 30 trees with a maximum depth of 20, and the features used for the training are 29 features as described in Table 1. To implement the 1D-CNN model proposed in [18], the first 784 bytes of the payload of each flow are converted into a 1D-Vector and an N+1-class 1D-CNN classifier is trained by Keras [27] with TensorFlow [28] as backend. The parameters of the 1D-CNN classifier are consistent with those in [18]. The ambiguous flows detection is not applied to these two classifiers. The parameters of our classifier are as follows. The Random Forest classifiers of the first two layers each include 30 trees with a maximum depth of 20. The Random Forest classifiers of the third layer include 20 trees with a maximum depth of 20, and the XGboost classifiers include 10 trees with a maximum depth of 5.

4.3. Evaluation on Dataset1. Before evaluating our classifier, we first verify how much mobile traffic can be identified using the DNS-based method described by Trevisan [11]. First, we extract the $\langle ServerIP, Domain \rangle$ pairs from the DNS traffic in Dataset1. Then the IP addresses with a single domain are used to identify traffic. Finally, this method can identify up to 30.66% of the flows, which account for 20.8% of the amount of bytes. Hence the traffic that can be identified by the DNS-based method is in the minority.

To evaluate our classifier, Dataset1 is randomly split into a training set (70% of samples) and a testing set (30% of samples). For each classifier the evaluation process is repeated 10 times (with different splits each time) and the average results are presented. The average precision, recall, and FP for target apps of each classifier are listed in Table 5. The detailed results are illustrated in Figures 2–4.

The results show that our method achieves the highest precision of nearly 99% and produces a much smaller average FP number than the other two classifiers. The 1D-CNN classifier has the highest FP number, resulting in its lowest precision. Our method produces a 94% reduction in FP compared to the baseline, which indicates that the third layer of our classifier has a better capability of excluding unconcerned traffic. However, our classifier has the lowest recall. As can be seen in Figure 3, TaoBao, BaiDu, and QQ's low recall decrease the average recall.

The recall of the second and third layer of the proposed classifier is shown in Figure 5. It can be seen that the classifier already has a low recall on the latter three apps in the second stage. In the second stage, 57.29%, 47.43%, and 33.97% of the instances of the latter three apps were classified as ambiguous flows, respectively, which is the main reason for the classifier's low recall. Then we examined the training data carefully. It is interesting to note that the latter three apps have a lot

Table 5: The average precision, recall, and FP for target apps of each classifier.

	Average Precision	Average Recall	Average FP
Random Forest	94.74%	86.74%	119
1D-CNN	88.68%	82.83%	318.14
Three-Layer Classifier	98.9%	52.71%	7.07
Three-Layer Classifier + Smoteenn	92.24%	61.01%	58.65

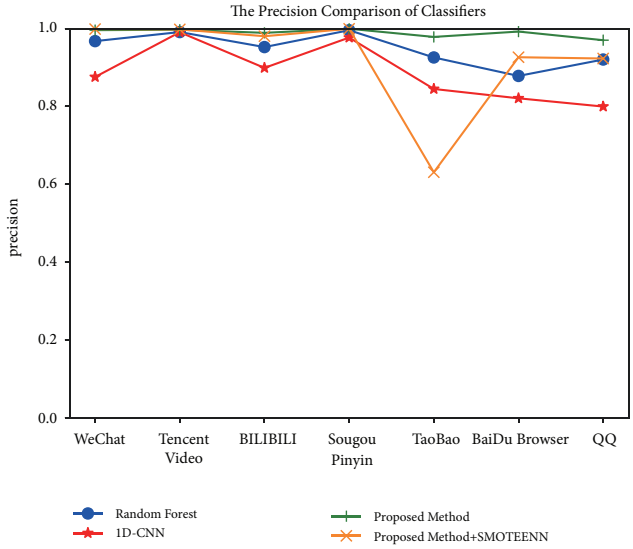


Figure 2: The precision comparison of classifiers.

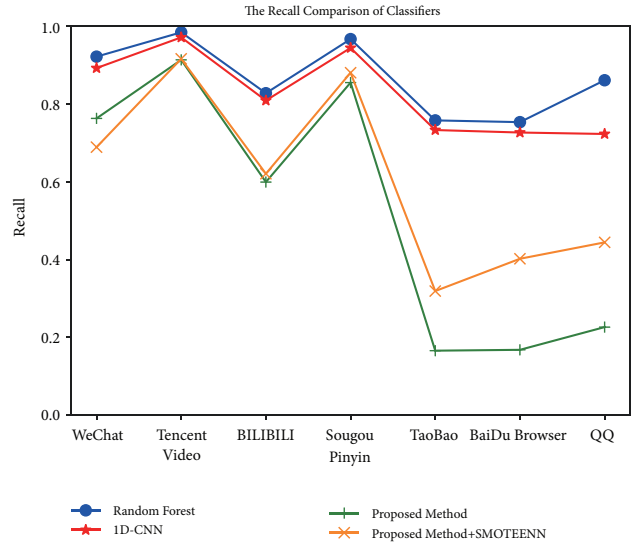


Figure 3: The recall comparison of classifiers.

of associations with the unconcerned apps in Dataset1. For example, QQ is an instant messaging app, but it integrates a lot of functions, such as news, mail management, and music playing. Moreover, these additional functions have independent apps which belong to the same company as QQ. This results in the fact that some traffic of QQ has similar behavior patterns to that of other apps. And the classifier would reject the judgment of these flows aggressively in order to prevent false positives. A similar phenomenon exists in TaoBao and BaiDu.

Additionally, we also investigate whether the low recall is related to the sample size, considering that the sample sizes of the latter three apps are indeed lower than those of the other four apps. Therefore, we oversampled the training set using the SMOTEENN [29] before training the second and third layer classifiers. The results are already shown in Table 4 and Figures 2–4. It can be seen that the sample size is not the main reason affecting the recall of the classifier. Oversampling does increase the recall of apps with smaller sample size but also increases the FP greatly.

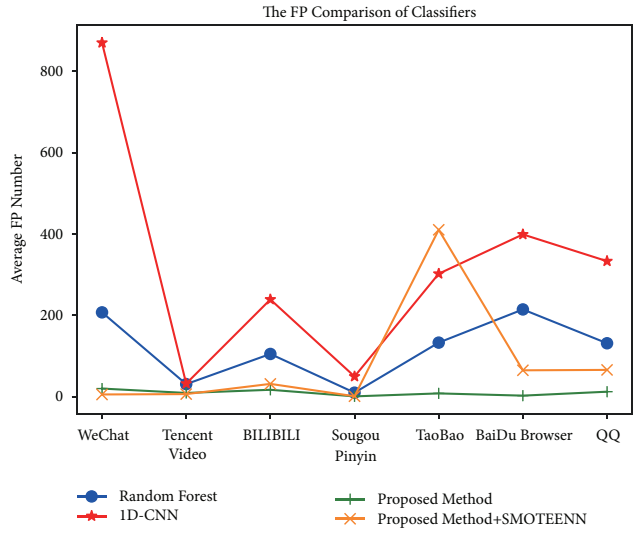


Figure 4: The FP comparison of classifiers.

4.4. Evaluation on Dataset2. We retrain the three classifiers with Dataset1 as training set and evaluate them on Dataset2. The results are listed in Table 6.

The results are similar to the evaluation results for Dataset1, which show that the three-layer classifier has a better capability of excluding unknown traffic compared to the other two classifiers. The proposed classifier produces 152 false positives in total, among which 45 flows come from zero-day apps. By contrast, Random Forest produces 1478 false positives, among which 403 flows come from zero-day apps. 1D-CNN produces 3963 false positives, among which 1348 flows come from zero day-apps. Therefore, our classifier can exclude up to 99.2% zero-day traffic.

Table 6: Evaluation results for Dataset2.

	WeChat		Tencent Video		BILIBILI		Sougou Pinyin		TaoBao		BaiDu		QQ	
	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
Random Forest	3295	212	*	151	*	215	733	20	*	376	60	281	83	232
1D-CNN	3082	807	*	130	*	664	750	213	*	691	56	309	53	1149
Three- Layer Classifier	2662	17	*	31	*	10	697	1	*	22	2	24	13	47

Table 7: Identification results of encrypted and unencrypted traffic.

	WeChat	Tencent Video	BILIBILI	Sougou Pinyin	TaoBao	BaiDu	QQ
Encryption ratio	19%	0.3%	74.39%	5.08%	61.38%	92.03%	24.09%
Unencrypted traffic precision	99.66%	99.94%	97.92%	100%	97.01%	100%	96.82%
Unencrypted traffic recall	86.63%	91%	55%	89.94%	23.90%	13.25%	29.62%
Encrypted traffic precision	99.24%	\	99.83%	\	97.13%	97.24%	96.77%
Encrypted traffic recall	30.84%	0	63.2%	0	14.72%	16.73%	6.88%

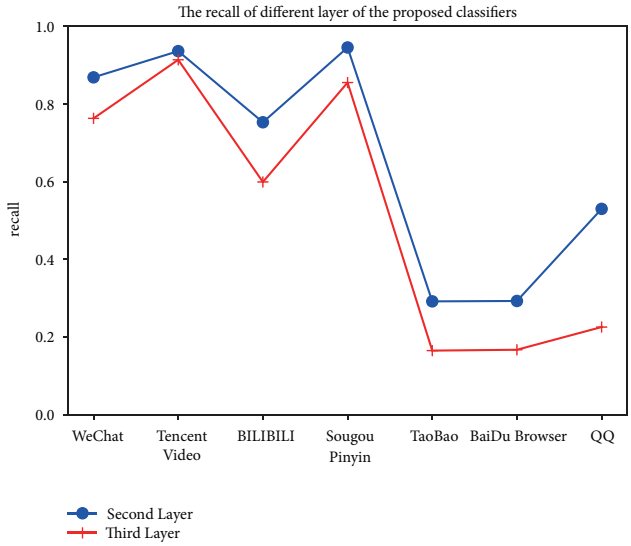


Figure 5: The recall of the proposed classifier.

Additionally, some interesting results are observed by scrutinizing the remaining false positives produced by our classifier. For example, 31 flows are misclassified as Tencent Video, among which 3 flows belong to QQ Music and 28 flows belong to Tencent News. It is noteworthy that Tencent Video, QQ Music, and Tencent News are all developed by Tencent, and Tencent News needs to access a lot of video resources which are also accessed by Tencent Video. A similar situation exists in QQ, where 46 false positives are from several other apps of Tencent, such as Tencent Maps and Tencent Weibo. Besides, we extracted the server IP address of the flow which is misclassified as Sougou Pinyin and then filtered the flows in the Dataset1 with the same server IP address. We note that all labels of those training samples are Sougou Pinyin. Therefore, the misclassified flow is likely to have an inaccurate ground truth.

4.5. Encrypted and Unencrypted Traffic Identification. In order to find how the proposed classifier behaves for the encrypted and unencrypted traffic, the proposed classifier is used to identify the encrypted and unencrypted traffic, respectively. For simplicity, flows over port 443 are considered to be encrypted traffic in this paper. The rest is considered as unencrypted traffic, even if the data of one flow is encrypted before transmission.

In Dataset1, 70% of encrypted flows and unencrypted flows are used to train the three-layer classifier, and the remaining are used as the test set. The encrypted traffic distribution of Dataset1 and identification results are shown in Table 7. The encryption ratio means the ratio between the size of encryption flows and the whole flow size.

It can be seen from Table 7 that the precision of the identification of unencrypted traffic is slightly higher than that of encrypted traffic, but the recall of the identification of unencrypted traffic is much higher than that of encrypted traffic. It shows that the identification of encrypted traffic is indeed more difficult than unencrypted traffic. One of the possible reasons is that the proposed classifier uses the payload byte values as features, and the byte values of encrypted traffic do not have distinct distinguishable features due to encryption. In addition, it can be seen that the encryption ratio of BILIBILI is relatively high, and its identification recall is also higher than other apps. Although TaoBao also has a higher encryption ratio, its recall is lower. In contrast, QQ has a low encryption ratio and a low identification recall. Therefore, the identification performance of encrypted traffic is not directly related to the encryption ratio of an app. Further comparisons can be made between identifiable encrypted traffic and unidentifiable encrypted traffic in the future work.

5. Discussion

Mobile traffic identification in real world requires more than merely identifying and discriminating apps traffic of interest. Another requirement is eliminating massive unknown app traffic. In contrast to other methods proposed in close-world settings, our method takes into account both the requirements. The experimental results obtain better performance than the state of the art. Throughout this work, there are some observations deserving further discussion.

First, the results show that 1D-CNN has poor capability of excluding unknown traffic. 1D-CNN uses payload as input, so the extracted features are limited to sequence features in the payload, which may lead to its poor performance. In contrast, models based on decision trees and side-channel data feature show better robustness. Second, the evaluation results for Dataset2 suggest that it is difficult to completely discriminate an app from other apps, especially when two apps have a close relationship. For example, two apps have the same functionality and are developed by the same company. In this case, both will access the same resources, thus generating similar traffic that cannot be discriminated. It is

worth mentioning that half of the false positives produced by our classifier are caused by this reason. Therefore, the impact of various associations between apps on identification tasks deserves further study. Additionally, the experimental results show that the identification of encrypted traffic is more difficult than the identification of unencrypted traffic. In order to better identify encrypted traffic, different traffic characteristics can be designed for encrypted traffic, and encrypted traffic and unencrypted traffic could be identified separately.

The limitation of our classifier is that it radically excluded many true positives in the second layer, resulting in a low recall for some apps. In future work, we will try to design different ambiguous traffic extraction method to detect ambiguous flows, thus enhancing the performance of our classifier.

6. Conclusion

In this paper, we proposed a three-layer classifier to identify mobile traffic. This classifier can distinguish the traffic between different target applications and eliminate unknown traffic effectively. We collected a representative dataset to validate the classifier. The proposed classifier has a precision of 98.9%, and the produced false positives are far less than the state of the art. Additionally, the experiment results show our classifier has great capability of detecting zero-day apps traffic, which meets the requirements of mobile traffic identification in real world networks.

Data Availability

The mobile network traffic data used to support the findings of this study have not been made freely available because of the need to protect user privacy. Requests for access to these data should be made to Shuang Zhao, zhaoshuang16@nudt.edu.cn.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. 61379148.

References

[1] "Mobile marketing statistics 2018," https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/.

[2] "Number of apps available in leading app stores as of 1st quarter 2018," https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[3] "Average number of new Android app releases per day from 3rd quarter 2016 to 1st quarter 2018," https://www.statista.com/statistics/276703/android-app-releases-worldwide/.

[4] "Mobile-First: 57% of traffic is now mobile," https://www.brightedge.com/resources/resources/research-reports/mobile-first-57-traffic-now-mobile.

[5] Z. Cai, Z. Wang, K. Zheng, and J. Cao, "A Distributed TCAM coprocessor architecture for integrated longest prefix matching, policy filtering, and content filtering," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 417–427, 2013.

[6] Y. Yu, J. Long, and Z. Cai, "Network intrusion detection through stacking dilated convolutional autoencoders," *Security and Communication Networks*, vol. 2017, Article ID 4184196, 10 pages, 2017.

[7] A. Tongaonkar, "A Look at the mobile app identification landscape," *IEEE Internet Computing*, vol. 20, no. 4, pp. 9–15, 2016.

[8] I. N. Bermudez, M. Mellia, M. M. Munafò, R. Keralapura, and A. Nucci, "DNS to the rescue: Discerning content and services in a tangled web," in *Proceedings of the ACM Internet Measurement Conference (IMC '12)*, pp. 413–426, November 2012.

[9] D. Plonka and P. Barford, "Flexible traffic and host profiling via dns rendezvous," *Workshop Satin*, 2011.

[10] T. Mori, T. Inoue, A. Shimoda et al., "Statistical estimation of the names of HTTPS servers with domain name graphs," *Computer Communications*, vol. 94, pp. 104–113, 2016.

[11] M. Trevisan, I. Drago, M. Mellia, and M. M. Munafo, "Towards web service classification using addresses and DNS," in *Proceedings of the 12th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC '16)*, pp. 38–43, September 2016.

[12] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in *Proceedings of the 3rd IEEE International Conference on Communications and Network Security (CNS '15)*, pp. 433–441, September 2015.

[13] H. F. Alan and J. Kaur, "Can android applications be identified using only TCP/IP headers of their launch time traffic?" in *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '16)*, pp. 61–66, July 2016.

[14] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "AppScanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proceedings of the 1st IEEE European Symposium on Security and Privacy (EURO S and P '16)*, pp. 439–454, March 2016.

[15] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2018.

[16] Z. Chen, B. Yu, Y. Zhang, J. Zhang, and J. Xu, "Automatic mobile application traffic identification by convolutional neural networks," in *Proceedings of the IEEE Trustcom/BigDataSE/ISPA*, pp. 301–307, Tianjin, China, August 2016.

[17] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proceedings of the 31st International Conference on Information Networking (ICOIN '17)*, pp. 712–717, January 2017.

[18] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-To-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proceedings of the 15th IEEE International Conference on Intelligence and Security Informatics (ISI '17)*, pp. 43–48, July 2017.

[19] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic," *Journal of Network and Computer Applications*, vol. 103, pp. 131–145, 2018.

[20] M. Lotfollahi, R. Shirali, M. Jafari Siavoshani, and S. Mohammdsadegh, "Deep packet: A novel approach for encrypted traffic classification using deep learning," 2017, https://arxiv.org/abs/1709.02656.

[21] "Weka 3: Data mning software in Java," https://www.cs.waikato.ac.nz/ml/weka/.

[22] B. Hullár, S. Laki, and A. György, "Early identification of peer-to-peer traffic," in *Proceedings of the IEEE International Conference on Communications (ICC '11)*, pp. 1–6, Kyoto, Japan, June 2011.

[23] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 2, pp. 23–26, 2006.

[24] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, pp. 785–794, August 2016.

[25] "Pragmatic software Network Log," https://play.google.com/store/apps/details?id=com.google code.networklog.

[26] "Scikit-learn: Machine learning in Python," http://scikit-learn.org/stable/.

[27] "Keras Documentation," https://keras.io/.

[28] A. Martin, A. Agarwal, P. Barham, E. Brevdo et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016.

[29] "Imbalanced-learn," 2018, https://github.com/scikit-learn-contrib/imbalanced-learn.