RScam: Cloud-Based Anti-Malware via Reversible Sketch

Hao Sun[1(✉)], Xiaofeng Wang[1], Jinshu Su[1,2], and Peixin Chen[1]

[1] College of Computer, National University of Defense Technology, Changsha, China
haosunlight@163.com
[2] National Key Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha, China

Abstract. Cybercrime caused by malware becomes a persistent and damaging threat which makes the trusted security solution urgently demanded, especially for resource-constrained ends. The existing industry and academic approaches provide available anti-malware systems based on different perspectives. However, it is hard to achieve high performance detection and data privacy protection simultaneously. This paper proposes a cloud-based anti-malware system, called RScam, which provides fast and trusted security service for the resource-constrained ends. In RScam, we present suspicious bucket filtering, a novel signature-based detection mechanism based on the reversible sketch structure, which provides retrospective and accurate orientations of malicious signature fragments. Then we design a lightweight client which utilizes the digest of signature fragments to sharply reduce detection range. Finally, we design balanced interaction mechanism, which transmits sketch coordinates of suspicious file fragments and transformation of malicious signature fragments between the client and cloud server to protect data privacy and reduce traffic volume. We evaluate the performance of RScam with campus suspicious traffic and normal files. The results demonstrate validity and veracity of the proposed mechanism. Our system can outperform other existing systems with less time and traffic consumption.

Keywords: Reversible sketch · Suspicious bucket filtering · Signature-based · Anti-malware · Cloud-based

1 Introduction

Cybercrime caused by malicious software(malware) is a persistent and damaging threat looms over businesses and consumers. Targeted attacks increase every year and expose more interest in social media and mobile devices as they are continuing to work their ways deeper into our digital lives. In the year of 2014, 496,657 web attacks blocked per day, and of the 6.3 million apps analyzied, one million of these are classified as mobile malware [1]. The McAfee Labs indicate attacks on the Internet of Things devices will increase rapidly due to hypergrowth in the number of connected objects, poor security hygiene and the high value of

B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 157–174, 2015.
DOI: 10.1007/978-3-319-28865-9_9

data on these devices [2]. Hence, it is urgent to provide a trusted and one-stop security solution to take care of data privacy in those resource-constrained ends.

To defend against various malware, signature-based detection approach still plays an important role and takes up a large proportion after decades of development in both industry and academic research. It is based on the theory that the crux of various malware, called signature, is generally unchangeable and can be detected at the early stage of propagation though the amount of malware samples is limited [3]. This approach is implemented by scanning and checking if a file contains the contents which match the known signatures. There are several commonly used and effective signature matching algorithms, such as Aho-Corasick [4] and Wu-Manber [5]. Besides, many heuristic and complex algorithms [21,22] are proposed for detecting unknown signatures. However, most of them consume a great mount of memory and time which is inapplicable for resource-constrained devices.

Two primary kinds of anti-malware systems with signature-based approach have been deployed according to their infrastructures in state-of-the-art technology. The first one is host-based systems which install detection agents in the users' devices and update the signature databases to ensure timely and complete security protection. ClamAV [6] is an open-source anti-virus system most widely used and many reformative works based on it are recently proposed, such as GrAVity [7]. However, these systems have become increasingly bloated with the development of malware attacks [8]. The problems mainly embody in the following two areas: (1) heavy resource consumption caused by the growing number of signatures, such as memory, time and network bandwidth; (2) system vulnerabilities are easy to be aimed due to their complexity.

The other solution is cloud-based security service [2] which places different types of detection agents over the cloud servers and offers security as a service. This newly developed framework is lenitive and cost-saving for resource-constrained ends. However, the existing cloud-based anti-malware technologies cannot address the following problems: (1) security vendors are designed to directly expose or deliver the signature databases to the clients which is unwillingness for the vendors and do not actually lighten the consumption of clients, such as SplitScreen [9]; (2) users have to upload the whole file contents which may result in some important information(e.g., location, password) leakage without realization, such as CloudAV [10]; (3) the optimization of traffic volume between the server and client is often neglected which is significant for the improvement of detection efficiency. Hence, it is hard to achieve high performance of security detection and data privacy protection simultaneously.

To overcome above shortcomings, we propose a cloud-based anti-malware system, called RScam, which provides fast and trusted security service for the resource-constrained ends. Specifically, we make the following contributions:

- We propose a novel signature-based detection mechanism, called suspicious bucket filtering, based on the structure of reversible sketch for cloud server. It can provide retrospective and accurate orientations of malicious signature fragments. As a result, the time and computation consumption in signature-

based malware scanning are cut down. To the best of our knowledge, no previous work has implemented similar endeavor.
- We implement a lightweight client which utilizes the digest of signature fragments to rapidly classify the file contents into suspicious and clean parts. It can dramatically reduce the scanning range with slight adjustable false positive and further avoid the accurate matching of the whole file contents.
- To protect the data privacy and reduce the traffic volume, we design the balanced interaction mechanism. The client transmits the sketch coordinates of suspicious file segments, instead of the whole file content, to the cloud after fast matching. As for the cloud server, transformations of signature fragments are sent back to the client, rather than the signature database.

We analyze the accuracy of the proposed mechanism theoretically to prove its validity and veracity with appropriate parameters. Our implementation of RScam consists of roughly 2.5K lines of C/C++ code for client and 4.5K for server which makes it easily applied to the resource-constrained devices. In addition, we evaluate the system by normal files and suspicious traffic captured from campus network with the number of signatures ranges from 460000 to 3700000. Statistical results show that RScam can outperform ClamAV and SplitScreen with lower time consumption and smoother increment when scanning increasing number of samples. Moreover, the traffic volume in RScam is averagely 10 times smaller than that in SplitScreen.

The rest of this paper is organized as follows: Section 2 introduces related work about signature-based malware detection. Section 3 gives a detail description about the system architecture and signature-based detection mechanism, followed by discussion of the system in Section 4. Section 5 presents the experimental results and analysis. Finally, we conclude the paper in Section 6.

2 Related Work

Signature-based malware detection remains important and technically reliable after decades of development in anti-malware industry.

ClamAV [6] is the most widespread and representative open-source anti-malware system. The latest database(main v.55 and daily v.19688) approximately contains 3700000 signatures consist of MD5 and regular expression signatures. Input file contents are sequentially matched with the signature database when scanning. If a known signature is successfully matched, the file is claimed to be infected by malware. The matching algorithms adopted are primarily Aho-Corasick [3] and Wu-Manber [5].

Recently, several efforts to improve the detection performance based on host have been proposed. Hash-AV [11] proposes a malware scanning technique which aims to take advantage of improvements in CPU performance. It utilizes hashing functions that fit in L2 caches to speed up the exact pattern matching algorithms in ClamAV. GrAVity [7] is a massively parallel anti-malware engine which utilize the good performance of GPUs to accelerate the process of scanning. Hardware implementations provide better performance, but it is always impracticable for

the resource-constrained devices, such as mobile phones and wearable devices. Deepak et al. [12] design a signature matching algorithm which is well suited in mobile device scanning, but its testing signatures are limited by fixed byte and the performance declines with the growth of signatures volume.

Cloud servers provide high-performance computation support to reduce the match consumption in malware scanning which is the main limitation of signature-based mechanism. Now it is attracting lots of security vendors to start to deploy their cloud solutions, like Trend Micro, Panda Security and Kaspersky Lab.

CloudAV [10] first puts forward the notion of cloud-based malware scanning in academic research and the authors apply their strategy to a mobile environment [13]. It runs a local cloud service consists of heterogeneous anti-virus engines running in parallel virtual machines and uses an end-user agent to transfer suspicious files to the cloud to be checked by all anti-virus engines. CloudAV achieves high detection rate, yet obviously, exposes the sensitive data which compromise users privacy. CloudSEC [14] achieves similar research which moves the analysis and correlation of network alerts into network cloud which also consists of plenty autonomous anti-malware agents, Jakobsson et al. [15] proposed a strategy for malware scanning which allows trusted cloud servers to look through the activity logs of clients in order to give timely monitoring and protection.

SplitScreen [9] designs a distributed anti-malware system based on ClamAV to speed up the malware scanning. SplitScreen designs its first scanning mechanism based on Bloom filter [16] to perform slight comparisons with file data and reduce the size to be accurately matched. However, bloom filter is not reversible which is similar to sketch data structure due to the multiple-to-one nature of hashing functions, so it does not store any information about the fragments. Actually the first scanning is so coarse-grained that the client still spends plenty of time and computation in exact pattern matching. Our study results show SplitScreen averagely spends 74.3 percent of its time in accurate pattern matching about 65 percent of pending files with small caches.

Our work is inspired by SplitScreen, but differs from it on two significant fronts. First, we employ reversible sketch structure with buckets containing suspicious signature fragments for malware detection. It is more efficient than Bloom filter structure because of needless to accurately match the whole contents of suspicious files. Second, we give consideration to the perspectives of both anti-malware vendors and end-users. Given the rapid incremental trend of signature volume and the security vendors unwillingness of directly exposing malware signature databases which are their core profit and competitiveness, the system opts to transmit the sketch coordinates of file fragments and transformation of malicious signature fragments between the client and cloud server which cut down the traffic volume simultaneously.

3 Design

In this section, we present a lightweight cloud-based anti-malware system called RScam, which can provide fast and trusted security protection for the

resource-constrained ends. We first show the system architecture of RScam and then give a detail description about the signature-based detection mechanism via reversible sketch structure in the proposed system.

3.1 System Architecture

To break out of high time consumption, which is primarily caused by a vast sum of signatures, RScam adopts the reversible sketch structure for effective representation and orientations of signatures, while designing balanced interactive mechanism to protect the data privacy and reduce the traffic volume.

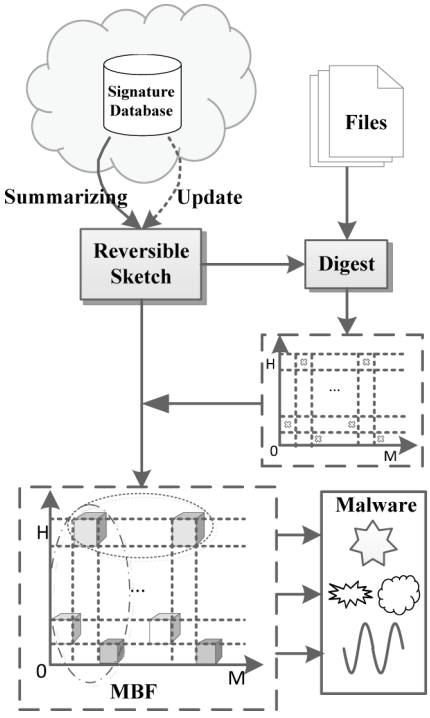

Fig. 1. The system architecture of RScam

We illustrate the system architecture of RScam in Fig. 1. The cloud server maintains the signature database, summarizes the signatures into the reversible sketch. Meanwhile, the cloud generates a digest of the sketch which represents the existence of signatures. The digest is stored in the client when RScam is firstly installed. The cloud updates the signature database and sketch periodically and sends the locations in the sketch where the changes take place to the client. The detail operations will be described in Section 3.3. As for file scanning, the client first initializes the file contents into the segments by the similarity method with

the signatures(described in section 3.2), then sifts out the unmatched segments with the digest. The matched ones are suspicious and their sketch coordinates in the digest are sent to the cloud, rather than the whole file contents. We design the suspicious bucket filtering(SBF) mechanism for the cloud to locate the malicious signature fragments according to the sketch coordinates from the client. The results which consist of transformation of malicious signature fragments and short signatures are sent back to the client as a confirmed report according to which the client takes corresponding security measures.

3.2 Signature Initialization

Let DB be the signature database managed in the cloud. Considering signatures do not have uniform length generally, we set a sliding window with length w to scan the signatures in DB. For an arbitrary signature S of length l, there will be a set of segements with length w-*byte* after initial scanning, namely, $S \rightarrow \{S_1, S_2, \ldots, S_{l-w+1}\}$. Moreover, we take account of the wildcards in specific signatures to map down multiple versions of a malware that originated from the same source. In a way, the initialization can be effective in handeling polymorphic malware caused by wildcards [11]. However, it is still impractical to deal with all possiblilties. In CloudEyes, the signatures with wildcard are roughly divided into two portions.

(1) Fixed-Size Wildcard: It denotes the wildcards which contains numbered probabilities. For example, $''?''$ matches any byte, $''a|b|c''$ matches $''a''$ or $''b''$ or $''c''$. We adapt modulo(q) in the wildcard signature initialization, which maps each string byte to a class between 0 to $q-1$(q is a random number smaller than 256), to support wildcard matching [17]. Therefore the matching space size is restricted because matching any value between the range of [0,$q-1$], instead of all possible values between 0 to 255, means successful hit. For instance, suppose a signature $''abcd?efgh''$ is initialized with $q = 4$ and $w = 9$. The initialization is processed by constructing four segments:$''abcd0efgh''$,$''abcd1efgh''$,$''abcd2efgh''$ and $''abcd3efgh''$. Similarly, $''abcd(x|y|z)efgh''$ is classified into three substrings: $''abcd0efgh''$,$''abcd1efgh''$ and $''abcd2efgh''$ because character x would be mapped to class 0 as $ASCII(x)\ mod\ q = 0$.

(2) Variable-Size Wildcard: It denotes the wildcards with unfixed size, such as, $''*''$ matches any number of bytes, $''\{n\}''$ matches n bytes. Considering the large amount of probabilities lead to serious performance slowdown, we ignore these wilcards and initialize the rest part of signature. For instance, a signature $''abcdef*ghijkl''$ or $''abcdef\{200\}ghijkl''$ is initialized with $w = 6$, the corresponding substrings are $''abcdef''$ and $''ghijkl''$.

Additionally, if a signature does not contain a fixed fragment at least as long as the window size, the signature cannot be initialized. Small value of w cannot provide enough amount of unique fragments which raises the rate of collision to an unacceptable level during mapping. Alternatively, if the value is too large, there is not enough granularity to answer queries for smaller file fragments in

detection. Study result of ClamAV's signature set for the 16-byte window size shows that the short-signature proportion is about 0.15% after initialization. This infrequence does not significantly reduce performance. For convenience, below we use X to represent a signature fragment after initialization.

3.3 Reversible Sketch Structure

Sketch structure is an aggregation method which maps diverse data streams into uniform vectors based on the Turnstile Model [18]. Let $I = \alpha_1, \alpha_2, \ldots,$ be a sequential input stream during a given time interval. Ecah item $\alpha = (\alpha_i, \mu_i)$ consists of a key $\alpha_i \in \{0, 1, \ldots, n-1\} \Leftrightarrow [n]$, and a value $\mu_i \in R$. The model assigns a time varying signal $T[\alpha_i]$ for each key $\alpha_i \in [n]$, and update $T[\alpha_i]$ with an increment of μ_i if a new item (α_i, μ_i) arrives. Most researches [19,20] based on sketch are applied to analysis of elements in flow, such as source and destination IP/Port, but rarely content. Our design is inspired by this structure whose properties can be applied in identifying malicious data fragments from large amount of suspicious data.

Reversible sketch(represented by RS) is based on the k-ary sketch data structure which H is the number of hash tables and m is the size of per hash table, i.e. $m = k$. In our design, each element of hash table consists of a container called $bucket(RB)$ which stores the information of signature and a bit called $digest(D)$ which stands for the bucket is empty or not, with the value 0 or 1 respectively. Let $h_1, h_2, \ldots, h_H$ be H functions randomly chosen from a class of 2-universal hash functions, each hash table adopts one independent function respectively. Assume an arbitrary signature X with length of w-byte, that is $X = \{x_1, x_2, \ldots, x_w\}$. As we adopt modulo($q$) to deal with the signature contain fixed-size wildcards initially, each byte of X(or file content) needs to do the same modulo arithmetic to avoid false negative rate in detection, although it will bring slight false positive rate. Hence the hashing result of X is $h_i(X) = h_i((x_1 \ mod \ q), (x_2 \ mod \ q), \ldots, (x_w \ mod \ q))$. Then we can use $L(X) = \{L_1(X), L_2(X), \ldots, L_H(X)\}$ which consists of $L_i(X) = (i, h_i(X))(1 \leq i \leq H)$ to be the sketch coordinate of X. When summarizing X into RS, $L_i(X)$ can be utilized to locate the corresponding reversible bucket $RB[i][j]$ and digest $D[i][j](j = h_i(X))$.

There are three operations related with RS:

(1) Insert(X,$L(X)$): Initially, RB contains no element and all the digests value is 0. For X which has not been mapped, $L(X)$ decides which buckets it belongs to. Then the sketch is updated as follows.

$$RB[i][j] \leftarrow RB[i][j] \bigcup \{X\}$$
$$D[i][j] \leftarrow 1, 1 \leq i \leq H$$

Fig. 2 illustrates the state of reversible sketch structure after inserting X_1, X_2 and X_3. The buckets labeled by coordinates mean each contains at least one signature and the rest stand for empties.

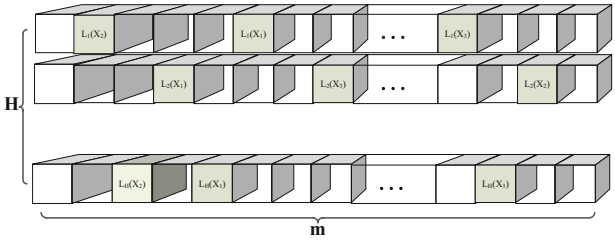

Fig. 2. Reversible Sketch Sturcture

(2) Delete(X,$L(X)$): For the signature X that is proved to be incorrect or reduplicate for malware description, the servers call delete operation to get rid of X from the sketch with following steps:

$$RB[i][j] \leftarrow RB[i][j] - \{X\}$$
$$D[i][j] \leftarrow 0, if\ RB[i][j] = \emptyset, 1 \leq i \leq H$$

(3) Update(Σ_X,Π_L,OP): The cloud needs to periodically update the signature database with the increment of signature quantity. $\Sigma_X = \{X_1, X_2, \ldots, X_n\}$ is the set of signatures need to be updated, $\Pi_L = \{L(X_1), L(X_2), \ldots, L(X_n)\}$ is the set of sketch coordinates to locate the signatures and OP is the set of operations($Insert$ or $Delete$) corresponding to each signature. After the $Update$ operation, the RB and D complete the similar changes with the two operations described above.

After summarizing the signature database into the reversible sketch, fundamental scanning about the database can be approximately answered very quickly according to the previous work [20]. However, more information about signature should be stored in the structure in order to insure the scanning veracity without the accurate scanning process like SplitScreen. Generally speaking, the basic signature database contains plenty of two-tuples $(signature, malwarename)$. Once a signature is matched, the comprehensible malware name is needed to show what kind of attack it is. So the malware name should be stored by certain format into the RS with corresponding signature segment. To balance memory consumption and searching speed in the implementation, we design the infrastructure based on red black tree for fast and dynamic operations. More theoretical analysis about the accuracy of reversible sketch structure is discussed in section 4 below and details about the performance are illuminated in section 5.

3.4 Matching Mechanism

The design of matching mechanism in RScam is inspired by two purposes we desired: (1) taking the demands of both security vendors and clients into account and (2) ensuring high performance in file scanning. Hence we divide the process of matching into two steps, fast matching and suspicious bucket filtering, for the client and cloud respectively. Detail descriptions are listed below:

(1) Fast Matching: In the RScam system, the reversible sketch structure, which contains the reversible buckets and digest, is designed to store the summarization of signature and service for matching. The digest is the crux of fast matching process which is stored in the client when the system is firstly installed. The files need to be initialized with w and q before scanning because of their diverse types and sizes, that is the file content should be incised into regular fragments and then do the modulo arithmetic. Let F be the set of file fragments after initialization, the purpose of fast matching is picking out the suspicious set of fragments F_{sus} and the corresponding set of sketch coordinates Π_{sus} with the digest D.

Algorithm 1 Fast Matching

Input: file fragments set F, digest D
Output: suspicious fragment set F_{sus} and sketch coordinate set Π_{sus}

```
1: F_sus, Π_sus = ∅
2: clear = 0
3: while each f ∈ F do
4:    calculate L(f);
5:    for i = 1 to H do
6:       if D[i][h_i(f)] = 0 then
7:          clear = 1, break; //f is not suspicious
8:       end if
9:    end for
10:   if clear = 0 then
11:      insert f into F_sus and L(f) into Π_sus
12:   end if
13: end while
```

For each fragment in F, we calculate its sketch coordinate in the digest and check the corresponding value to estimate its existence. Only successful matching in all H hash tables make the fragment suspicious, the others are normal because the hash functions bring no false negative during signature summarization. Algorithm 1 presents details of fast matching mechanism. This process is easy to be applied in the client due to its lightweight and can largely reduce the number of file fragments to be further confirmed. Considering the privacy protection of client, we send the sketch coordinates of suspicious fragments to the cloud after fast matching, which also can cut down the communincation consumption for the client.

(2) Suspicious Bucket Filtering: This process aims at confirming the suspicion of the fast matching result. The basic idea is checking every reversible bucket according each sketch coordinate sent from the client to find the signature fragment which exists in all the H hash tables. As we describe above, different types of signature need to be initialized into regular segments. Let N_S

be the total number of signatures in the DB(including the signatures with wildcards after initialized), and $\bar{l}$ be the average length of the signatures, w is the size of sliding window, m is the size of per hash table. So the number of segments after initialization is $(\bar{l} - w + 1) \cdot N_S$, and each bucket averagely contains $t = (\bar{l} - w + 1) \cdot N_S/m$ segments. One possible heuristic to find the target signature fragment is take the intersections of each bucket, nevertheless this can lead to a enormous amout of fragments output that do not match and needless computation which called Reverse Sketch Problem [20]. So we build another small red black tree T_{mal} as a filtering buffer which is indexed by the signature fragments stored in the bucket to count their times of appearance.

Algorithm 2 Suspicious Bucket Filtering

Input: Sketch coordinates Π_{sus},reversible bucket RB
Output: Set of malicious signature fragments R_{mal}

```
1: T_mal, R_mal = ∅
2: while each L(f) ∈ Π_sus do
3:    for i = 1 to H do
4:       if RB[i][h_i(f)] ≠ ∅ then
5:          for k = 1 to t do
6:             insert X_k ∈ RB[i][h_i(f)] into T_mal
7:          end for
8:       end if
9:       else T_mal = ∅
10:   end for
11:   for each fragment X ∈ T_mal do
12:      if count(X)= H then
13:         insert X into R_mal
14:      end if
15:   end for
16: end while
```

Algorithm 2 shows the process of malicious bucket filtering. T_{mal} is a signature-fragment buffer for each sketch coordinate $L(f)$ in Π_{sus}. First, we pass over the $L(f)$ which any one of the corresponding reversible buckets is empty which is caused by the hashing collision. Then we insert all signature fragments contained in the targeted RB into T_{mal} and pick out the fragments whose count is H. The result R_{mal} consists of the confirmed signature fragments which can be utilized to claim the malice of file fragment in the client. The filtering shrinks the scope of malicious signature fragments in $O(H \cdot t)$ time at the price of slight memory cost. After suspicious bucket filtering, the cloud sends the result back to the client. The confirmed signature fragments and short signatures should be compared with the suspicious file fragments to make sure the veracity of matching mechanism. The cloud can take some simple transformation of the fragments to avoid direct exposure. This can be implemented by using a bijective reversible function from fragment space $[N]$ to $[N](N = 2^{8w})$. The security vendors can

also choose some classical encryption algorithms to ensure the secure communication which is beyond the scope of this work. The client will take some security measures, such as deletion or isolation, with the infected files after validate the matching results.

4 Discussion

In this section, we discuss the accuracy of the reversible sketch structure which is measured based on the false negative and false positive rates generally. A false negative occurs when a fragment summarized into the RS earlier is asserted as clean when matching. While the false positive occurs when a query fragment not summarized into the RS is incorrectly stated as present. There are two types of false positives in RScam. The first one is caused by the hash functions employed in the RS, which is called *hashing false positive*. Secondly, the modulo arithmetic adopted in the initialization brings the possibility of collision between two different fragments and modular hashing of signature fragments adopted in the storage mechanism. Here we call it *fragment false positive*. In what follows we will conduct the theoretical and statistical analysis of these measurements.

4.1 Fasle Negative

The false negative is caused by the initialization based on fixed-size slide window, rather than the hash function. For example, suppose the signature $''abcdefg''$ has been summarized into RS with window size of 6, which means two signature fragments are constructed and mapped into the RS: $''abcdef''$ and $''bcdefg''$. Now if we scan the file content $''bcdef''$ will respond that the file was clean which is incorrect. It is remarkable that false negative in RScam would occur only for the short file content whose length is less than w bytes. So it greatly depends on the length of the scanning content. However, this situation is seldom in prevalent security detection because sizes of files to be scanned are always larger than w bytes which we set in the evalutaion(more details in Section 5.4). Hence we can adjust the value of w to minimize the false negatives and ensure the false positives acceptable. Therefore we put our focus on calculating the false positive rate in the rest of this section.

4.2 Hashing False Positive

The hash functions we use above are 2-universal which make the hash results are nearly randomized. Hence the principle and accuracy of summarization is similar with the Bloom Filter. This type of false positive comes from the hash collisions which may lead to the conclusion that a specific fragment is suspicious when it is not. Alternatively, the false negative will never exist. We learn about the probability of false positive in a bloom filter can be calculated with following relation.

$$FP = (1 - (1 - \frac{1}{m})^{kN})^k \tag{1}$$

where m is the length of bloom filter, k is the number of used hash functions and N is the amount of inserted elements. We can easily conduct the hashing false positive of a hash table in RS. As described earlier, each hash table uses only one hash function and $(\bar{l}-w+1)\cdot N_S$ fragments are inserted into it. So the false positive of each hash table is:

$$\alpha = (1-(1-\frac{1}{m})^{(\bar{l}-w+1)\cdot N_S}) \tag{2}$$

There are H hash tables built in RS which makes the hashing false positive tenable if and only if collisions exist in all the H ones. According to the relation (2), let FP_h be the hashing false positive of RS that is

$$FP_h = (1-(1-\frac{1}{m})^{(\bar{l}-w+1)\cdot N_S})^H \tag{3}$$

4.3 Fragment False Positive

As we described in Section 3, the RScam system adopt the modulo arithmetic to deal with the wildcards in specific signatures. However, this will introduce collisions between different fragments. Specifically, there are two distinct scenarios lead to fragment collision discussed below.

(1) Collision Before Summarization: This scenario occurs between two unsummarized fragments, that is, the hashing value of them is uniform. Suppose that S and S' are two different strings(signatures or files) with same length of $\bar{l}$. Assume that $S = s_1 s_2 \dots s_{\bar{l}}$ and $S' = s_1^{'} s_2^{'} \dots s_{\bar{l}}^{'}$, and the number of classes by q, then the collision happens if each byte of string belongs to same class after modulo. Let F_1 be the false positive before summarization, which is calculated by:

$$F_1 = (\frac{\lceil \frac{256}{q} \rceil}{256})^{\bar{l}} \leq (\frac{1}{q}+\frac{1}{256})^{\bar{l}} \tag{4}$$

(2) Collision After Summarization: This scenario occurs when the unsummarized file content is matched which is incorrect. Suppose that $S = s_1 s_2 \dots s_{\bar{l}}$ is initialized with the window length of w. As noted earlier, the number of w-byte fragments after initialization is $(\bar{l}-w+1)$. The collision happens when all these fragments are wrongly resulted in suspicion. Let F_2 be the false positive after summarization, we can conclude the relation below according relation (4):

$$F_2 = (\frac{\lceil \frac{256}{q} \rceil}{256})^{w\cdot(\bar{l}-w+1)} \leq (\frac{1}{q}+\frac{1}{256})^{w\cdot(\bar{l}-w+1)} \tag{5}$$

Consequently the probability of collisions are the sum of F_1 and F_2. However, we should negate the situation that all the bytes in the string are really equal.

Moreover, the collision is directly related to the number of signatures summarized into the RS. Let FP_f be the fragment false positive rate, then we have:

$$\begin{aligned} FP_f &= [F_1 + F_2 - (\tfrac{1}{256})^{\bar{l}}] \cdot N_S \\ &\leq [(\tfrac{1}{q} + \tfrac{1}{256})^{\bar{l}} + (\tfrac{1}{q} + \tfrac{1}{256})^{w\cdot(\bar{l}-w+1)} - (\tfrac{1}{256})^{\bar{l}}] \cdot N_S \end{aligned} \tag{6}$$

In conclusion, the false positive of RScam can be computed by the summation of relations (3) and (6). As observed in Fig. 3, the hashing false positive, denoted by FP_h, is much larger than the fragment false positive, denoted by FP_f, with different number of signatures after initialization. So FP_f is negligible compared to FP_h. It is reasonable that FP_h grows close to 1 when the number of signatures grows close to the size of hash table because empty reversible buckets get rare.

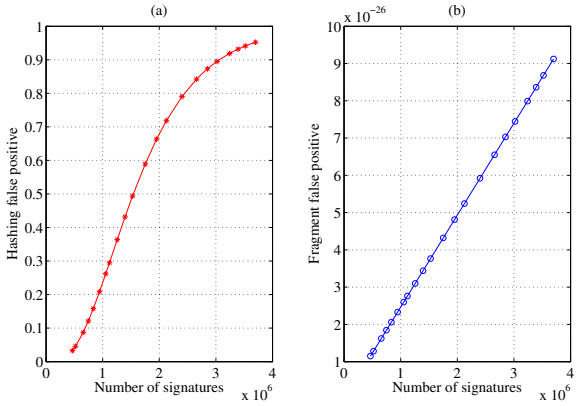

Fig. 3. Two types of false positive in RScam with $m = 2^{24}, w = 16, \bar{l} = 30, H = 4, q = 8$ and different number of signatures between 460000 to 3700000. (a) is hashing false positive and (b) is fragment false positive.

5 Evaluation

In this section, we evaluate the performance of the RScam system and make some comparison with the ClamAV and SplitScreen. We have implemented RScam based on the file and signature identification model of ClamAV with approximately 7K lines of C/C++ code which consist of 4.5K for cloud server and the rest for client. The signature databases which originate from the ClamAV open source platform contain two types of signatures: whole file or segment MD5 signatures and regular expression signatures. We employ several versions from Nov. 2008 to Nov. 2014, which the number of signatures ranges from 460000 to 3700000. If unspecified, we implement the evaluation with the latest database (main v.55 and daily v.19688) and show the average results over 20 runs. Our total 36GB suspicious data set consists of about 240000 unique samples by MD5

Table 1. Memory Cost

Memory	The number of signature					
	530K	860K	1M	2M	3M	3.7M
Cloud(MB)	110	198	274	642	1032	1500
Client(MB)	39	39	43	46	51	55
SplitScreen Client(MB)	58	63	67	74	78	84

hash, which are captured by specific IDS from the campus network. The experiments are performed on a CentOS 5.6 virtual cloud server(8 cores, 32-GB memory and 2.53 GHz) and a common open research network emulator based on OpenVZ which provides different types of virtual machines.

5.1 Memory Analysis

As described earlier, we adopt the reversible sketch structure in the cloud server. Each bucket averagely contains $t = (\bar{l} - w + 1) \cdot N_S/m$ signature segments, so the entire memory cost is $w \cdot t \cdot m \cdot H$ bytes theoretically. We utilize the dynamic red black tree structure to store these segments and prune the reduplicate ones after initialization. Meanwhile, we assign each malware name a unique number in advance to reduce the overhead. This process takes up a period of time, but we don't count it in the performance of RScam because it performs only once at the starting of evaluation. Unless otherwise specified, we use $w = 20, m = 2^{24}, q = 4, H = 2, \bar{l} = 20$ for the RS in our experiment. Table 1 lists the average memory cost of the cloud server and client with various number of signatures after we adjust from different versions when scanning 600MB suspicious samples. As observed, the memory cost of cloud server in RScam mounts up with the growth of signatures. However, it is acceptable for security vendors. In the side of client, the cost does not grow with the number of signatures. We also evaluate the memory cost of SplitScreen client and find our client appropriator less memory, which means RScam is more applicable than SpliltScreen because the latter calls the accurate scanning of ClamAV after its first scanning.

5.2 Time Analysis

We evaluate the time performance of the RScam system in the virtual machine as a resource-constrained client with 350MB memory, 256KB L2 cache and 1GHz CPU, and the bandwidth between the cloud and client is 1MB/s. The testing data are the samples randomly chosen from our data set. The average size of each sample is 2MB. Meanwhile, we make comparisons with the system of ClamAV and SplitScreen in the same environment. We implement this with 1MB signature database(main v.54 and daily v.13810) because ClamAV exhausts the system memory when running with larger signature databases. Fig. 4 shows the details

of the time cost. RScam outperform the others with lower time consumption and smoother increment. We can conclude that small cache volume slows down the detecting speed of SplitScreen distinctly. In some condition, SplitScreen even runs slower than ClamAV.

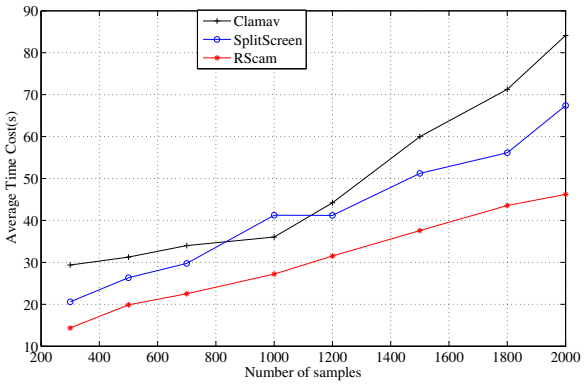

Fig. 4. Time performance of RScam, SplitScreen and ClamAV using different number of samples.

Moreover, we are concerned about the composing of the time cost illustrated in Fig. 5 which reveals the effect of our matching mechanism. The mean percentage of accurate scanning of SplitScreen is 74.3% while that of RScam is 16.4%. The fast matching takes account of all the file fragments which matched in the digest to avoid the accurate scanning of whole file content, while the fast scanning of SplitScreen only reserve the first matched file fragment to label the file to be accurately scanned. In this way, we cut down a mass of computation and time. Hence, we can confirm that the matching mechanism based on the reversible sketch structure can largely improve time performance.

5.3 Traffic Analysis

Another important inspiration of our design is data privacy protection with slight amount of traffic between the client and server. We achieve this through the communication mechanism labored above. The client sends the sketch coordinates of suspicious file fragments to the server and the server send the short signatures and transformation of malicious signature segments back to the client.

Fig. 6 illustrates the average traffic between the client and server with different number of signatures in RScam and SplitScreen when scanning 2GB suspicious samples. The experiment parameters are same with section 5.2, besides the client and server are connected with TCP protocol. As observed, the traffic in RScam is averagely 10 times smaller than that in SplitScreen, and stand

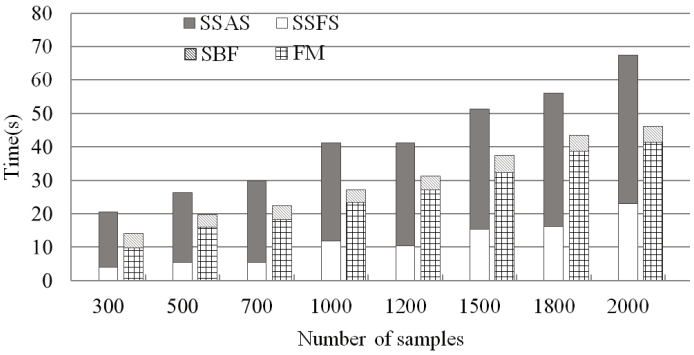

Fig. 5. The composing of time cost of RScam and SplitScreen. SSAS and SSFF stand for the accurate and first scanning of SplitScreen, respectively. SBF and FM stand for suspicious bucket filter and fast matching of RScam.

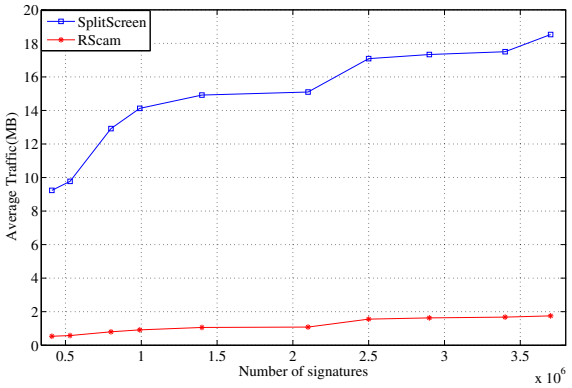

Fig. 6. The traffic between the client and server with different number of signatures.

smooth with the growth of signatures. The traffic of RScam during scanning is averagely 39.8 KB/S which is acceptable for the resource-constrained clients, such as mobile phones and pads.

5.4 Practical Accuracy

We discuss the accuracy of the reversible sketch structure in Section 4 and conclude that it can be measured primarily by hashing false positive. Moreover, we give a practical test of the accuracy in detecting 5972 clean PE files(totally 1.42GB) with different window size under the latest signatures database. Table 2 lists the details of the practical accuracy. The false positive of RScam is calculated by the number of suspicious file fragments divided by the total number of file fragments. For MD5 signatures, we fix the value of w to be 16, the other

variable values are for regular expression signatures. The false negative is calculated by the number of short signatures divided by the total number of signatures. Small window size cannot provide enough possibilities for the large amount of signature fragments which caused high false positive. While large window size will produce more short signatures which bring higher false negative and not be fine-grained enough. Hence we can ensure the high accuracy of RScam with considered window size and 20 seems to be the moderatest value.

Table 2. Practical accuracy of RScam

Window size	Fasle Positive	Short Sigs	False Negative
$w = 12$	7.861%	3467	0.092%
$w = 16$	5.726%	5741	0.152%
$w = 20$	3.380%	7676	0.203%
$w = 24$	2.371%	10929	0.289%

6 Conclusion

In this paper, we proposed RScam, a cloud-based anti-malware system which provide fast and trusted security protection for resource-constrained clients. In RScam, we design a novel signature-based detection mechanism based on the reversible sketch structure which dramatically reduce the scanning range and provide retrospective and accurate orientations of malicious data fragments. Meanwhile, we design the balanced interaction mechanism to protect the data privacy and reduce the traffic volume for both the clients and security vendors. Evaluations with suspicious campus network and normal files show that the system is able to achieve fast and accurate malware detection with slight traffic and acceptable memory requirement. As part of our future work, we are planning to address several challenges. The memory cost in the cloud serve side can be reduced ulteriorly by modular hashing of the signature fragments before they are inserted into the buckets, and we are trying to enhance the detection performance by adopting multiple hashing in each hash table.

Acknowledgement. This research is supported in part by the program of Changjiang Scholars and Innovative Research Team in University (No. IRT1012); Science and Technology Innovative Research Team in Higher Educational Institutions of Hunan Province ('network technology').

References

1. Symantec Corporation Internet Security Threat Report 2015, vol. 20 (2015). http://www.symantec.com/security_response/publications/threatreport.jsp
2. McAfee threats report: fourth quarter (2014). http://www.mcafee.com/us/mcafee-labs.aspx

3. Chen, Z., Ji, C.: An information-theoretic view of network aware malware attacks. IEEE Transactions on Information Forensics and Security **4**(3), 530–541 (2009)
4. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Comm. of the ACM **18**, 333–340 (1975)
5. Wu, S., Manber, U.: A fast algorithm for multi-pattern searching. Technical Report TR-94-17, University of Arizona (1994)
6. Clamav. http://www.clamav.net
7. Vasiliadis, G., Ioannidis, S.: GrAVity: a massively parallel antivirus engine. In: Jha, S., Sommer, R., Kreibich, C. (eds.) RAID 2010. LNCS, vol. 6307, pp. 79–96. Springer, Heidelberg (2010)
8. AV-comparative.: On-demand detection of malicious software. Technical Report, AV-comparative (2010)
9. Cha, S.K., et al.: Splitscreen: enabling efficient, distributed malware detection. In: Proc. of NSDI, pp. 12–25 (2010)
10. Oberheide, J., Cooke, E., Jahanian, F.: CloudAV: N-version antivirus in the network cloud. In: Proc. of the 17th USENIX Security Symposium, pp. 91–106 (2008)
11. Erdogan, O., Cao, P.: Hash-AV: fast virus signature scanning by cache-resident filters. International Journal of Security and Networks **2**, 50–59 (2007)
12. Venugopal, D., Hu, G.: Efficient signature based malware detection on mobile devices. Mobile Information Systems **4**(1), 33–49 (2008)
13. Oberheide, J., Veeraraghavan, K., Cooke, E., Flinn, J., Jahanian, F.: Virtualized in-cloud security services for mobile devices. In: Proc. of the First Workshop on Virtualization in Mobile Computing, pp. 31–35 (2008)
14. Xu, J., Yan, J., He, L., Su, P., Feng, D.: CloudSEC: a cloud architecture for composing collaborative security services. In: 2nd IEEE International Conference on Cloud Computing Technology and Science, pp. 703–711 (2010)
15. Jakobsson, M., Juels, A.: Server-side detection of malware infection. In: Proceedings of the 2009 Workshop on New Security Paradigms Workshop, pp. 11–22. ACM (2009)
16. Bloom, B.H.: Space/Time Trade-offs in Hash Coding with Allowable Errors. Comm. of the ACM **13**, 422–426 (1970)
17. Haghighat, M.H., Tavakoli, M., Kharrazi, M.: Payload Attribution via Character Dependent Multi-Bloom Filters. IEEE Transactions on Information Forensics and Security **8**(5), 705–716 (2013)
18. Muthukrishnan, S.: Data streams: Algorithms and application. Foundations and Trends in Theoretical Computer Science **1**(2) (2005)
19. Krishnamurthy, B., Sen, S., Zhang, Y., Chen, Y.: Sketch-based change detection: methods, evaluation, and applications. In: Proceeding of ACM SIGCOMM IMC, pp. 234–247 (2003)
20. Schweller, R., Li, Z., Chen, Y., Gao, Y., et al.: Reversible sketches: enabling monitoring and analysis over high-speed data streams. IEEE/ACM Transactions on Networking **15**(5), 1059–1072 (2007)
21. Tang, Y., Xiao, B., Lu, X.: Signature Tree Generation for Polymorphic Worms. IEEE Transactions on Computers **60**(4), 565–579 (2011)
22. He, M., Gong, Z., Chen, L.: Securing network coding against pollution attacks in P2P converged ubiquitous networks. Peer-to-Peer Networking and Applications **8**(4), 642–650 (2015)