Enhancing Mobile Malware Detection with Social Collaboration

Liu Yang, Vinod Ganapathy, Liviu Iftode
Department of Computer Science
Rutgers, The State University of New Jersey
{llyangru,vinodg,iftode}@cs.rutgers.edu

Abstract—Resource-constrained mobile devices pose a challenge to the design of security mechanisms. Existing host-based malware detection solutions are often resource-intensive. We present a decentralized and resource-aware malware detection architecture for mobile devices. Our approach leverages two key ideas: social collaboration and the concept of a hot set. The hot set concept states that not all malware signatures are equally important. At any given time, some signatures (the hot set) are more likely to be matched than the others. We leverage this concept by only keeping the hot set of signatures in the main memory of a mobile device, and distributing the whole signature database among devices belonging to the social group of the device owner. We demonstrate the feasibility of our approach by implementing a proof-of-concept (Social-AV) based on an open source anti-malware software, ClamAV. Experiments show that Social-AV reduces the memory consumption to about 55% of the amount consumed by ClamAV, while retaining the same detection capability.

Keywords—malware detection; social network; social collaboration; hot set

I. INTRODUCTION

The increasing use of smartphones for tasks such as personal communication, online banking, shopping, and social networking has made them a prime target for cybercriminals and enabled novel attacks (e.g., [1], [2]). Unlike desktops, smartphones have strict resource constraints on memory, computational ability, and energy. These constraints challenge the design of security mechanisms for smartphones [3].

Existing mobile malware detection techniques can be broadly categorized as host-based and cloud-based. Most mobile-specific versions of antivirus software by security vendors are similar to their desktop variants and provide limited detection with significant resource overhead. Advanced approaches, such as behavior detection engines, are capable of detecting sophisticated threats, but are also resource-intensive. Researchers have therefore proposed to offload detection to the cloud [4], [5]. Such centralized solutions do not consume resources on the mobile phone, but require the phone to be connected to the cloud to enable detection.

We propose an alternative mobile malware detection technique that is both decentralized and resource-aware. Our approach builds upon the recent trend of using mobile phones for social networking. We leverage this trend to enable collaborative detection of malware by distributing the set of malware signatures amongst the members of a social group. To do so, we rely on the observation that not all malware signatures are created equal [6]. At a certain time, some signatures are more likely to be matched by malware than others. We call this set of signatures the hot set. The concept of a hot set is backed by the top threat lists posted periodically by security vendors [7], [8]. Our approach is to keep the hot set in main memory of each phone to enable quick detection of common threats. However, the remaining signatures are important as well, and we divide and distribute the whole signature database among the social group of the mobile device user (cold sets). The user therefore relies on his social network of friends to detect uncommon threats. We call this approach Social-AV. Our experiments show that the memory consumption of a Social-AV-enhanced ClamAV installation is about 55% of the memory consumed by a traditional ClamAV installation.

II. RELATED WORK

Existing cloud-based mobile malware detection solutions often rely on centralized servers, which scan files for remote clients [4], host exact replicas of phone in virtual environments [5], or perform aggregated malware behavior analysis [9], etc. Our design is a decentralized architecture and does not add additional hardware cost.

Researchers have developed resource-aware techniques for PC-based malware detection, examples including HashAV [10] and SplitScreen [6]. Such techniques are also valuable for mobile platforms. Our work uses social collaboration as a way to ensure resource-awareness in mobile malware detection.

Collaborative approach has been explored for botnet detection [11] and SMS spam filtering [12]. Existing approaches adopted a client-server model, where clients report information to a centralized server, which performed aggregated analysis. Our work is also a collaborative approach, but we adopt a decentralized architecture for normal operation.

Supported in part by NSF grants CNS-0831268, CNS-0915394, CNS-0931992, and CNS-0952128.
III. OVERALL ARCHITECTURE OF SOCIAL-AV

The key intuition behind our approach is to leverage the social group of a user to enable collaborative malware detection for mobile devices. When a new piece of untrusted data is received by any one of the devices in this social group, the devices interact to detect whether the data is malicious. To reduce inter-device communication, we place a hot set of signatures in the main memory of each device, and distribute the whole signature database amongst the social group (cold sets). This approach reduces the memory consumption on each mobile device, thereby making malware detection resource-aware.

In our approach, each participating device only stores a portion of a full signature database, i.e., the full signature database is partitioned to multiple subsets and distributed among the mobile devices of a social group. For convenience, we call each subset a cold set. A phone that only stores portion of database may fail to recognize malware defined by signatures outside its portion. To address this problem, we design a data structure, called a prefilter, and deploy it on each phone. The prefilter is used to catch all suspects that are potential malware. A prefilter must satisfy three requirements: (1) it must be memory-efficient; (2) it must allow the majority of clean files to pass; and (3) it must have one-sided error, i.e., it must report potential matches of malware, but must never miss a match.

In Figure 1, we present the architecture of Social-AV. A Social-AV deployment consists of a number, say $n$, of AV agents installed on phones in a social group. Each AV agent is equipped with a prefilter $\mathcal{P}$, a hot set $\mathcal{H}$, and a cold set $\mathcal{C}$. The union of the cold sets in all agents of the social group is the full signature database. Each agent is logically divided into two components: a scan module and a service module. The scan module scans the data stream for malware, and the service module services signatures to phones in the social group. The group size $n$ is a configurable parameter.

When there is a scan need, e.g., a phone $A$ in the group downloads a file $f$ from a website, it performs detection using the method illustrated in Figure 2. At first, the scan module of $A$ scans $f$ using its prefilter $\mathcal{P}$. If there is no match in $\mathcal{P}$, then $f$ is clean and the scan completes. If $f$ triggers a match in $\mathcal{P}$, then $f$ is considered as a suspect file. Next, the scan module scans the suspect $f$ with signatures in its hot set $\mathcal{H}$. If there is a match, then a malware alert is triggered and the scan completes. If no match is found in $\mathcal{H}$, the scan module scans $f$ with signatures in its cold set $\mathcal{C}$. If a match is found in $\mathcal{C}$, then $f$ is a malware. If no match is found in $\mathcal{C}$, the scan module assembles a signature request and sends it to a friend phone $B$, which can potentially provide signatures to match $f$. We will describe how to find a friend phone to send in the next paragraph. Upon receiving a signature request, $B$ queries its cold set using information in the request and returns the retrieved results, assembled as a signature reply, to phone $A$. After receiving a signature reply, $A$ learns whether $f$ is clean by either performing a scan on $f$ using the received signatures or by interpreting the signature reply.

Now, we describe how to find out which friend to contact if a phone could not decide the innocence of a suspect file with signatures in its hot set and cold set. In Social-AV, signatures are sorted in a specific order before being partitioned to $n$ cold sets. After partitioning, each cold set can be described by a signature range interval, denoted by the start and ending values of signatures in the cold set. Each phone stores the $n$ range intervals along with the IDs of phones in which the $n$ cold sets are stored. If a file $f$ has a match against the prefilter of a phone $A$ and $A$ could not decide whether $f$ is clean by its hot set and cold set, then $A$ checks its stored $n$ range intervals to find out an interval $I$ such that the matched pattern is between its start and ending values. Phone $A$ then assembles a request and sends it to a
phone $B$ that has an ID associated with the interval $I$.

In practice, the computation of hot set, prefilter, and cold sets can be performed by an AV provider. The initialization of Social-AV can be performed using a centralized service. When a new member joins a social group, he is given a prefilter, a hot set, and a cold set by an administrator. After such initialization, the system operates in a decentralized fashion. This centralized service can be provided, say, as a Facebook application.

We note that each phone evolves its hot set during operation to keep the most useful signatures in its main memory and discard the less useful ones. The replacement strategies for signatures in hot set can be LFU (Least Frequently Used), LRU (Least Recently Used), etc. In this way, each device has its own hot set, which may be different from others in the social group. Also, a device does not replace signatures in its cold set. When new signatures are available, a centralized service is responsible for updating cold sets for different individual phones.

The Social-AV architecture described above has three benefits. First, it is resource-efficient. Since each phone only stores a portion of the full signature database, the memory consumption of an individual device is reduced. We will quantify the resource-efficiency in Section V. Second, it uses existing telephony infrastructure for normal operation. Third, it can readily benefit the off-the-shelf AV software. For example, the scan module of a phone in Social-AV can run ClamAV with a reduced signature set.

IV. IMPLEMENTATION BASED ON CLAMAV

We demonstrate the feasibility of Social-AV by implementing a prototype based on an open source anti-malware application—ClamAV. version 0.96.1. Before presenting our implementation, we give a brief introduction on ClamAV.

A. ClamAV

ClamAV is a signature-based malware detection tool. Its database has two types of signatures for malware description: (1) MD5 checksums of malware; and (2) regular expressions (regexps) describing the patterns of malware body. About 88% of signatures in ClamAV are MD5-based and the remaining 12% signatures are regexps. MD5 checksums have fixed length of 16 bytes, while regexp signatures have length numbering from 8 to 450 bytes. During scanning, ClamAV checks a file sequentially against its signature database. It computes the MD5 checksum of a file or a file segment and compares the checksum and file size with entries in its MD5 signature set. It also checks whether the file contents matches any entries in the regexp signature set. If in either case there is a match, a file is reported as a malware.

B. Prefilter Implementation

In the general architecture of Social-AV, the scan module and the prefilter of an AV agent are system plug-ins. The scan module acts as a scan engine and can be realized using an off-the-shelf anti-malware application, e.g., ClamAV. For different scan engines (by different AV vendors), the construction of the prefilter may be different, but the idea remains the same. In this section, we show how to construct the prefilter from a ClamAV signature database.

Based on the fact that ClamAV has two types of signatures, our constructed prefilter consists of two parts: a Bloom filter and an index set, built from MD5 and regexp signatures respectively.

1) Bloom Filter (prefilter for MD5s): Bloom filter is a technique to represent a set of elements $S = \{s_1, ..., s_n\}$ to support efficient member queries [13]. A first step to construct a Bloom filter is to allocate a vector of $m$ bits, initialized with 0. Then choose $k$ independent hash functions $h_1, h_2, ..., h_k$, where each function has range of $\{1, ..., m\}$. For each element $s \in S$, the bits at positions $h_1(s), ..., h_k(s)$ of vector $v$ are set to 1. To query the membership of $a$, check the bits at $h_1(a), ..., h_k(a)$. If any of these bits is 0, then $a$ is not in set $S$; otherwise $a$ is in $S$ with certain probability $p$. The probability $1 - p$ is called false positive. In practice, the parameters $k$ and $m$ should be chosen such that the false positive is acceptable. We borrow the method in [6] to construct a Bloom filter from MD5-based signatures by choosing $k = 4$ and $m = 2^{32}$. Denoting the 16 bytes of a MD5 signature by $b_0, b_1, ..., b_{15}$, the 32-bit hash values of a signature are computed as linear combinations of two hash functions $h_1 = b_0 ... b_3 + b_4 ... b_7$ and $h_2 = b_8 ... b_{11} + b_{12} ... b_{15}$.

The constructed Bloom filter satisfies the three requirements described in Section III. The satisfaction of the first two requirements can be confirmed by the results in Section V. The third requirement, one-sided error, is naturally satisfied by the property of hash functions in the Bloom filter.

2) Index Set (prefilter for regexps): The insight behind index is to support approximate and efficient matching test of a file against a regexp signature. We define the index of a regexp signature as a $w$-bytes substring (fragment) extracted from the signature. For example, if $w = 6$, then an index of “DOS.Chemmy.A (Clam)=0eb80001508cc805e00050b8700750cb” is a 6-bytes fragment extracted from the hexadecimal part. The indexes extracted from all the regexp signatures form a index set, which acts as a prefilter for regexp signatures. The choice of $w$ have impacts on the compactness and false positive rate of an index set. A small $w$ renders a compact index set, but may result in a high false positive. Thus, we need to find a tradeoff between the compactness and false positive. Even for a fixed $w$, some fragments may be more likely to match a clean file than other fragments in a signature. For each signature, we need to use a fragment which is the least likely to match a clean file. To find such fragments, we use DF (Document Frequency) technique. For a set of clean documents, the DF value of a fragment denotes the number
of documents which match this fragment. We need the DF value of an index be as small as possible.

The above index construction method assures that an index only makes one-sided error, i.e., if a file matches an index, it has a certain probability to match a signature; however, if a file does not match any index, it will not match any signature in the database. Thus, requirement (3) in Section III is satisfied. In our implementation, we chose \( w = 12 \). In Section V, we will show that the compactness and low false positive requirements are also satisfied.

C. File Scanning

Before describing how a file is scanned by Social-AV, we briefly describe how signatures from ClamAV were reorganized in Social-AV. We sorted MD5 signatures using MD5 as the key, and sorted regexp signatures using their indexes as the key. We then partitioned the sorted MD5 and regexp signatures to \( n \) subsets (the cold sets) and distributed them amongst \( n \) AV agents. To support signature retrieval, we compiled a signature range list \( L \), which labels the signature ranges in all cold sets. An entry in \( L \) has the form of \([ \text{ID}, (\text{L}_\text{MD5}, \text{U}_\text{MD5}), (\text{L}_\text{INDEX}, \text{U}_\text{INDEX}) ]\), where \( \text{ID} \) is the identity of a phone, \((\text{L}_\text{MD5}, \text{U}_\text{MD5})\) denotes the lower and upper bounds of the MD5 signatures, and \((\text{L}_\text{INDEX}, \text{U}_\text{INDEX})\) denotes the lower and upper bounds of indexes of regexp signatures in the cold set of a phone.

We now describe the implementation of file scanning. For an input file \( f \), the AV agent of a phone, say \( A \), scans \( f \) using its prefilter \( P \). In particular, it checks the MD5 checksum of \( f \) (denoted by \( MD5(f) \)) against the Bloom filter part of \( P \), and the content of \( f \) against the index part of \( P \). If no match is found in both cases, \( f \) is clean. If there is a match in either case, then \( f \) is a suspect. The AV agent then scans \( f \) with signatures in its hot set \( H \). If a match is found, then a malware alert is triggered; otherwise the AV agent scans \( f \) with signatures in its cold set \( C \). If a match is found in \( C \), then \( f \) is a malware. If no match is found in \( C \), the AV agent needs to consult its social group to determine the innocence of \( f \). To do so, we need to consider two cases: (1) \( MD5(f) \) matches the Bloom filter of \( P \); (2) content of \( f \) matches an index \( pt \) in \( P \). In either case, the AV agent checks the the signature range list \( L \) using \( MD5(f) \) (or \( pt \) in the second case) and finds out the ID of a phone, say \( B \), such that \( MD5(f) \) (or \( pt \)) falls into the associated range. It then assembles a signature request \( Req = (MD5(f), \text{size}(f)) \) in case (1) (or \( Req = pt \) in case (2)) and sends to phone \( B \), where \( \text{size}(f) \) denotes the size of \( f \). If phone \( B \) receives a request in the form of \( (MD5(f), \text{size}(f)) \), it queries its cold set using the request. If a match is found, then it returns the associated malware name; otherwise \( f \) is clean. Phone \( B \) then sends the query result (malware name or clean) as a reply \( Rep \) to \( A \). If \( B \) receives a request in the form of \( pt \), it queries its cold set using \( pt \) and assembles the retrieved signatures as a reply \( Rep \) and sends to phone \( A \). Upon receiving a reply, phone \( A \) can decide whether \( f \) is clean by either associating the content of the reply (malware or clean) to \( f \) or scanning \( f \) with the signatures encapsulated in \( Rep \).

D. Hot Set Construction

In our implementation, the size \( N \) of hot set \( H \) is configurable. Initially, \( H \) contains \( N \) signatures randomly selected from the signature database. \( H \) evolves during the AV scanning. In particular, if there is a match to the prefilter \( P \), then signatures associated with the matched pattern are added to \( H \) to replace same number of least useful signatures. We implemented the Least Frequently Used (LFU) mechanism as a signature replacement strategy.

V. EVALUATION

In this section, we present the feasibility evaluation of Social-AV under a simulation environment. We first show that Social-AV provides an improved resource-efficiency comparing with a traditional host-based detection software, ClamAV, followed by the demonstration of the effectiveness of hot set.

A. Experimental Environment and Data Sets

We simulated Social-AV by spawning a number of processes on an Intel Core2 Duo E7500 Ubuntu 10.04 LTS machine, running at 2.93GHz with 2GB of memory. The simulated Social-AV consisted of 10 participants. Each phone of Social-AV was simulated by a process running on a separate terminal. The cold sets of this Social-AV were partitioned from a ClamAV signature database having 845,276 signatures.

We used traffic data crawled from Twitter for evaluation. We crawled tweets from 3,295 users and gathered URLs appearing in their tweets. We then visited each of these URLs to simulate user browsing activity and gathered a total of 809MB worth of data (text, image, pdf, etc.) over five time periods (Table II).

B. Memory Consumption

Table I presents the memory consumption by ClamAV and a Social-AV consisting of 10 participants. For the 845,276 signatures, ClamAV requires 101MB memory, while a Social-AV agent requires 55MB, of which 41MB are consumed by the scan module, and 14MB are consumed by the service module. Part of the 41MB memory is consumed by the Bloom filter and index set. The average false positive rate of the prefilter is 5.25%. These results show that the prefilter satisfies the compactness and low false positive requirements described in Section III.

\(^{1}\text{Regexp signatures with length less than 12 bytes or not containing a non-wildcard substring of at least 12 bytes are stored on each device. Such signatures account for about 0.19\% of the full signature database in ClamAV. We did not construct indexes for these signatures.}
C. Hot Set Evolution

We use the approach in Section IV-D for hot set construction and evolution. We set $N = 1,000$ and simulated hot set evolution on an AV agent $A$ equipped with a prefilter, a hot set, and a cold set. Initially, the hot set $H_0$ consisted of 1,000 signatures randomly selected from the malware database. The hot set evolution was simulated through a number of iterations. In the first iteration, $A$ scanned Data Set 1 (shown in Table II) and found 20 unique suspect matches (indexes or MD5s), among them 0 was found in $H_0$, i.e., $A$ had to send out a signature request containing 20 suspect matches to its friend devices. The number of signature misses was 20 and the hit rate of $H_0$ is 0%, as is shown in the second line of Table III. At the end of scanning Data Set 1, $H_0$ evolved to $H_1$. Equipped with $H_1$, $A$ scanned Data Set 2 and the observed hit rate of $H_1$ was 50% (see Table III), higher than that of $H_0$. At the end of scanning Data Set 2, $H_1$ evolved to $H_2$. Using the same approach, $A$ scanned Data Sets 3, 4, and 5 sequentially and the corresponding hit rate of $H_2$, $H_3$, and $H_4$ were shown in Table III. As we see, the best hit rate of the hot set was 92%, which means that 92% of suspects can be resolved by signatures from the hot set. This can reduce the network traffic incurred by signature requests and replies.

VI. CONCLUSION

We designed Social-AV, a decentralized and resource-aware malware detection architecture for mobile devices. Our experiments show that an AV agent of Social-AV consumes about 55% of memory as the amount consumed by a traditional anti-malware software with the same detection capability.

REFERENCES