Exploiting Altruism in Social Networks for Friend-to-Friend Malware Detection

Matthew J. Probst*, Jun Cheol Park**, and Sneha Kumar Kasera+
VMWare*, Adobe**, University of Utah+

Abstract

We propose a novel malware detection application—SocialScan—which enables friend-to-friend (f2f) malware scanning services among social peers, with scanning resource sharing governed by levels of social altruism. We show that with f2f sharing of resources, SocialScan achieves a 65% increase in the detection rate of 0- to 1-day-old malware among social peers as compared to the the detection rates of individual scanners. We also show that SocialScan provides greatly enhanced malware protection to social hubs.

I. Introduction

Over the past several years, there has been an explosion in the popularity and utility of online social networks (OSNs). Traditional out-of-band communication methods—including phone calls, traditional mailed letters, and face-to-face conversations—are in many cases being replaced by interaction and collaboration over OSNs. In this paper, we present a novel application, *SocialScan*, that leverages altruism between peers in an OSN for distributed malware detection for web content. SocialScan is an N-version [1] malware detection system. The protection mechanisms within SocialScan scale with the number of system participants by leveraging existing anti-malware resources on the clients of social peers.

A scanning service could be offered by clusters (cloud) of well-managed dedicated malware detection servers [1] that provide a large diverse set of malware checkers for a fee and are very likely to accurately detect malware. However, a centrally controlled service offers its own challenges. First, a centrally controlled service can become a single point of failure or a single target of attack, even with replication. This centrally controlled service might not be always available when needed. A clever adversary can target the centralized service while introducing new malware in the networks. Second, the centrally controlled services, in many cases, give a *false sense* of privacy to their users. The privacy of all of the user access behavior is contingent upon how well protected the user data are at these centralized service clusters. These are always a prime target of attack because an adversary, for the amount of effort required, can obtain a large amount of private information (in comparison to the adversary attacking individual users). An even bigger problem is that some centralized service providers might sell the private information of their users to other businesses or use that information in unauthorized or unethical ways. Third, fundamentally, simple performance analysis of p2p and client-server architectures shows the expected lack of scalability of the client-server model. Centralized- and cloud-based malware protection mechanisms are much more limited in their resources and thus cannot provide all clients on the Internet with effective but resource-intensive anti-malware techniques including both N-version [1] and execution-based [2] [3] virus detection. We believe that the p2p model, combined with social networks, offers a new and unique opportunity to explore the distributed applications space not just for higher reliability and performance but also for creating a level playing field when it comes to user privacy.

In this paper, we first design the SocialScan system. We propose the use of an existing social altruism metric [4] to determine the peers for scanning objects (files retrieved from the Internet) and for exchanging logs of object scans. Then, we develop our methodology for deciding whether access to an object should be allowed or denied. This methodology uses the results of the data available for that particular object being requested, including the results of previous local scans, as well as peer scans obtained from directed scan requests. We propose three decision making strategies - one based on local scans only, a Paranoid strategy requiring a minimum number of diverse scanners, and a Dynamic strategy that adjusts the required diversity of scanners based on the age of the object. Next, we implement SocialScan as a set of Python modules and services on top of an SQLite database. Using our implementation on the Amazon EC2 cloud, we show that with sharing of resources across social peers, both the Paranoid and Dynamic strategies achieve a 65% increase in the detection rate of 0- to 1-day-old malware among social peers as compared to the the detection rates of local scans only. Using simulations of both scanner diversity and scan result sharing, driven by social connectivity from real-world Facebook data, we find that SocialScan provides the top 5% of socially connected users—the social hubs—with the greatest levels of both scanner diversity and shared scan results. We also show through simulation that even relatively small groups of social peers can provide significant levels of scanner diversity.

II. Adversary Model

SocialScan is primarily designed to increase the probability that malware will be detected when accessed by a client running the SocialScan software. The principle set of adversaries for SocialScan are the creators of common malware—software

978-1-4799-5890-0/14/$31.00 ©2014 IEEE

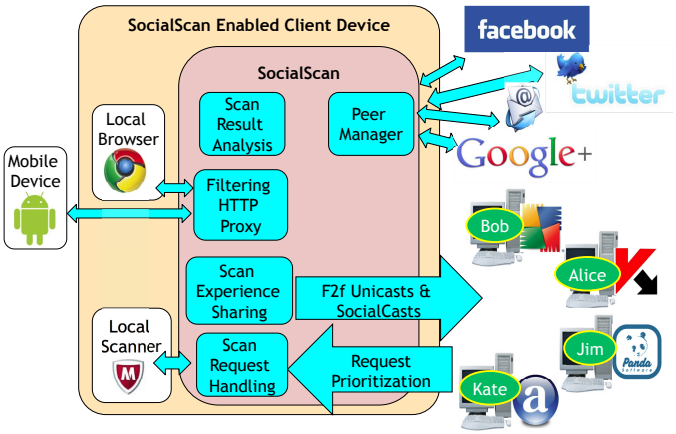

Fig. 1. SocialScan Overview

that has the goal of infecting and exploiting computers when retrieved and opened by unsuspecting Internet users. The design outlined in this paper is not focused on attacks against SocialScan itself. Such attacks by infected SocialScan nodes (client systems running SocialScan on behalf of their respective users) might include: false negative scan responses and false positive scan responses (denial of service/object blocking attempts). While each SocialScan participant can dynamically tune the level of scanning redundancy and scanner diversity required before accepting an object in an attempt to mitigate false scan responses by requiring larger and more diverse quorum responses, we defer the evaluation of such threats and candidate mitigation methods to be the subject of future work.

III. SocialScan Malware Detection System

With their work on Cloud-AV [1], Oberheide et al. have shown the benefits of utilizing a cluster of servers running a heterogeneous set of anti-virus software. This is referred to as N-version virus scanning. The authors show that although the anti-virus signature sets used by diverse vendors intersect, there is no single signature set that can effectively act as a superset of all other vendor signatures. Individual AV (anti-virus) products have the potential to be compromised [5]. This provides motivation for using divergent signature sets and AV products to scan for malware. Individual home and small office users rarely have the budget or technical expertise to construct and maintain a cluster of N-version virus scanners, but they commonly have social peers that are willing to share resources (CPU, memory, bandwidth, etc). A user's social peers likely run diverse anti-virus scan engines and signature sets. A licensing issue *might* arise due to SocialScan requiring a peer to scan malware for other peers by using their single-user virus scanner. We believe that this issue will get addressed in legal frameworks in a broader sense as it applies to other application software and other distributed systems (including those promoting cyber foraging) as well. The legal aspects of this issue are beyond the scope of this paper.

We design SocialScan to leverage these existing anti-malware resources to improve malware detection for all system participants. An overview of SocialScan is shown in Fig. 1. SocialScan provides two important services: first, passive sharing of the results of object scans among social peers; and second, providing a medium by which social peers can request immediate scans of particular objects; SocialScan has the goal of improving malware detection accuracy while minimizing additional object scanning latency as described below:

Malware Detection Accuracy: By leveraging friend-to-friend (f2f) scanning services, we design SocialScan to improve a node's accuracy in detecting malware within the objects requested by the user. Utilizing the same N-version philosophy of the Cloud-AV [1] project, we design SocialScan to increase the malware detection accuracy of individual nodes by encouraging the sharing of scan results among social peers running diverse virus scanners. With the level of resource sharing—i.e., servicing scan requests and sharing results—governed by levels of altruism among social peers, SocialScan maximizes the anti-virus resources of social-hubs. As the recipients of larger amounts of scanning resources, social-hubs will be armed with the highest levels of protection against malware.

Object Scanning Latency: We design SocialScan to minimize any latency penalty created by relying on social peers to assist with malware detection. In contrast with Cloud-AV, which relies on a dedicated local cluster of scanning hardware and software, SocialScan relies on the sharing of surplus memory, bandwidth and CPU resources across geographically dispersed nodes. Given the unpredictable availability of social peer resources as well as the lack of guarantees on the level of willingness of a peer to service scan requests, we design SocialScan with a no-assurances approach as to scan request response times from peers.

A. Social Relationship Identification

Before SocialScan enabled nodes can collaborate directly with their peers, they must first identify true social peers. SocialScan utilizes one or more social networks (e.g., Facebook, Weibo, etc) so as to identify active social relationships among users based on their relative levels of social connectivity. The traditional SMTP email social network can also be used. A significant body of research has been conducted in analyzing and quantifying relative levels of altruism—commonly referred to as distance—between social peers. We design SocialScan to support any analysis module that will yield a quantifiable and relative level of altruism among social peers. For our particular implementation of SocialScan, we approximate relative levels of altruism among peers within a social network using the *SocialDistance* metric, which is explained in detail in our earlier work, SocialSwarm [4]. Briefly, a proportional and direct level of altruism, $A(a,b)$ between a peer a and another peer b is given by the ratio $I(a,b)/I(a,all)$, where $I(a,b)$ is the number of reciprocal interactions a has had within a given time window with b, and $I(a,all)$ is the number of reciprocal interactions a has had with all of its peers during the same window of time. The SocialDistance between a and b is the inverse of $A(a,b)$. For indirect peers that are more than one

hop away in the social network, the altruism is depreciated by a hop decay factor resulting in a larger SocialDistance between the indirect peers.

B. IP Address and Key Exchange

To bootstrap direct f2f interaction after identifying true social peers, SocialScan—running on each peer device—needs to learn the current IP address and TCP port numbers used by their social peers. SocialScan also generates a simple public/private key pair for f2f message encryption and shares the public key with the user's social peers. SocialScan leverages established messaging mechanisms on social networks to exchange IP address and public key information among peers.

C. Active Scan Requests

SocialScan places all of its scanning logic into a central module called DecisionHandler (described in detail in Section IV). This module can be tuned based on the preferences of the system user. The DecisionHandler module promulgates scan requests via SocialCast to all social peers within a given SocialDistance or to send unicast scan requests to specific peers. When a prioritized request is received from a social peer SocialScan will service that request and send a response message to the requester containing the scan results, information on the scan vendor, and the date/time stamp of the scanner signature set used.

D. Request Queuing and Prioritization

Given that SocialScan nodes have limited resources and give the very common possibility that active scan requests from peers—across one or more applications—would exceed a client's resource capacity, it becomes critical to be able to arbitrate contention for resources. SocialScan prioritizes peer requests based on the SocialDistance of the peers that sent the requests. SocialScan has no assurance that an active scan request sent to other peers will be serviced at all or handled within a desired time window. Active scan requests in SocialScan are asynchronous and best-effort only.

E. Passive Scan Experience Sharing

Over time, a node will accumulate scan results from both local user requests as well as active scan requests from peers. Scan results are stored in a local database table of scans. A node will also store the results in a quickly searchable hash table known as a *Scan Log*. The hash table key for one entry in the Scan Log is the SHA256 hash of: (URL + log creator ID + a nonce for the Scan Log). The hash table key for a second entry in the Scan Log is the SHA256 hash of: (file contents + log creator ID + a nonce for the Scan Log). The value for both entries in the hash table is the scan result with a 0 indicating the object was found to be benign and a 1 indicating the object was found to be malicious. The nonce associated with a Scan Log is part of the Scan Log metadata that is sent to peers along with the Scan Log. Peers thus can look up arbitrary URLs and objects in the Scan Log using the same nonce.

Both of these keys are also added to a Bloom Filter known as a *Scan Digest*. The Scan Digest contains an entry if the object has been scanned and the result stored in the Scan Log. The motivation for using a Bloom Filter to pre-share information on URLs comes from Cache Digests [6] [7] which are widely used among caching HTTP proxy servers to identify peer proxy server contents. As with Cache Digests, Scan Digests are small in comparison with a full list of URLs, including those contained in a Scan Log. Scan Digests are small enough to reside in ram, whereas larger Scan Logs would typically reside on disk. Given the possibility of Bloom Filter collisions, Scan Digests may contain false positives, Scan Digests are always used in conjunction with Scan Logs. Scan Digests may be viewed as a manifest of the Scan Log keys with lossy compression.

We now describe the method used by SocialScan for Scan Digest and Scan Log rotation, sharing, and eviction. Scan Digests by their nature do not allow existing entries to be deleted. Individual entries within a Bloom Filter do not contain semantic data about when the scans were conducted, what scan engine was used, and the date of the signature set used. We design our Scan Digests to include scanner vendor and signature set information as accompanying metadata. Thus, Scan Digests and Scan Logs must be rotated at least each time these metadata change (e.g., when the scanner signature set is updated). To maintain a steady flow of new information on objects to social peers, Scan Digests and Scan Logs may be rotated several times each day. When Scan Digests and Scan Logs are rotated, the current "active write" pair of a Scan Digest and a Scan Log are closed and a new Scan Digest and a new Scan Log are opened for writing.

At the time of rotation, SocialScan places the Scan Log in a hidden (retrievable but not searchable) location on a locally web service. SocialScan then sends a Socialcast to trusted peers with the following pieces of information:

- The URL of the new Scan Log
- The Scan Digest associated with the Scan Log
- Metadata, including the scan engine that was used, the signature set that was used, and the date/time of the rotation creation.

Peers that receive this notice retrieve the associated Scan Log when and if their SocialScan instance so chooses. SocialScan appraises the potential value of a peer's announced Scan Log by evaluating the perceived SocialDistance to the creator of the Scan Log; with Scan Logs generated by closer peers being considered as more reliable and valuable. Peers may also compare their own recent access history with the contents of the Scan Digest. Should the peer have a close-enough perceived SocialDistance and should the Scan Digest indicate a sufficient level of correlated access history, SocialScan will retrieve the full Scan Log associated with that Scan Digest.

When SocialScan is queried to evaluate whether an object is malicious or not, it will first identify any scanning experiences for the object that it has already received from its social peers in the way of Scan Logs. Scan Digests are sufficiently small to be cached in RAM, whereas larger Scan Logs are more suitably stored on slower and lower cost media. For this reason, SocialScan will first perform a lightweight search of

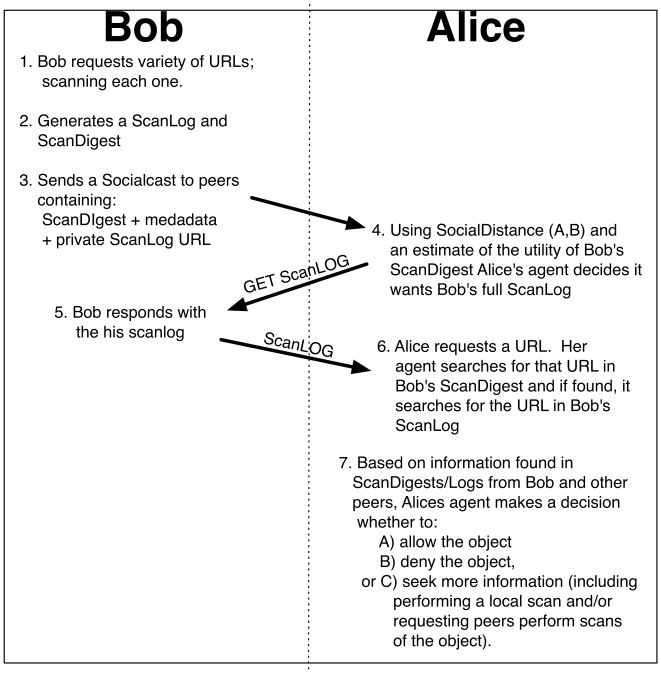

Fig. 2. SocialScan Experience Sharing

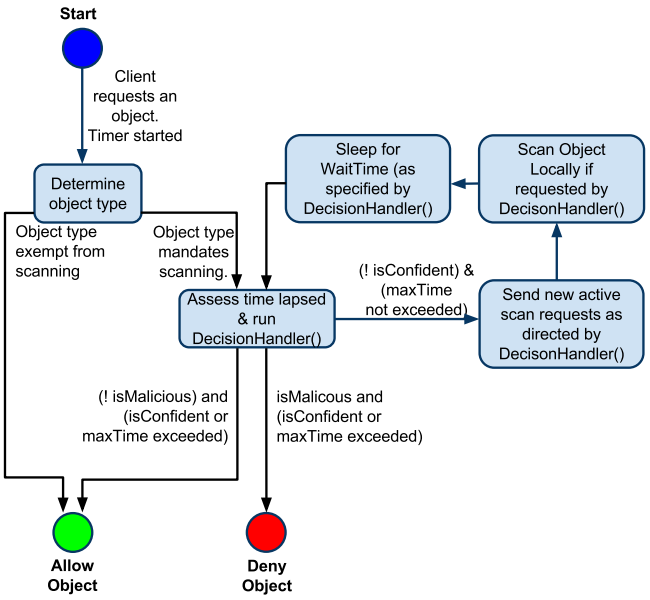

Fig. 3. SocialScan State

each memory-resident Scan Digest looking for the particular URL. If the URL is found in a particular Scan Digest, then SocialScan will proceed to search for an entry in the disk-based Scan Log. Given the basic nature of hash tables, the Scan Log search is a $O(logn)$ operation. If a value is found during the look up, then this value is the scan result claimed by the peer who performed the scanning. Scan result values that are found are passed to the DecisionHandler (described in section IV). If no value is found during the hash table look up, then a Bloom Filter collision has occurred in the Scan Digest and no result is considered. The process of creating, sharing, and using Scan Digests and Scan Logs can be seen in Fig. 2.

Over time, a node will accumulate a variety of different Scan Digests and Scan Logs from both close and distant social peers. Each Scan Digest and Scan Log will eventually need to be evicted. To assist in Scan Digest and Scan Log eviction, SocialScan maintains a maximum age limit for both Scan Digests and Scan Logs. Once a Scan Digest and Scan Log pair exceeds this age, the pair is automatically evicted. SocialScan also keeps an exponentially weighted moving average of the utility rating for each Scan Digest and Scan Log based on the number of objects of interest that were found in a particular Scan Digest and Scan Log over a given time period. Scan Digests and Scan Logs with the lowest utility rating (those with the fewest objects of interest to the evaluator) are chosen as candidates for eviction, should disk- or ram-cache capacity limits mandate evictions. Eviction based on a utility rating will automatically bias the SocialScan cache towards retaining objects from neighbors who are accessing similar content—those with correlated object-access behaviors. The greater the correlation of the objects accessed among peers the higher the probability will be that Scan Digests and Scan Logs received by a node from its peers will already contain scan results of value to that node.

F. Clickstream Privacy

The sharing of users' clickstreams (the list of objects that they have accessed or are attempting to access) is a significant privacy concern with SocialScan. The existing SocialScan design includes two built-in mechanisms to promote clickstream privacy. First, sharing of Scan Logs partially anonymizes clickstreams by using one-way hashes of URLs and object contents. An attacker may perform a dictionary attack by walking through a given set of known clickstream entries in an effort to reconstruct the exact clickstream followed by the Scan Log creator. For this reason, SocialScan also avoids inserting object types that are known to have a low probability of being compromised (e.g. static HTML). Avoiding inserting certain objects that are part of the clickstream reduces the certainty by which an attacker can positively reconstruct full clickstreams. Second, Scan Digests and Scan Logs are only shared with peers within a tunable SocialDistance radius. Likewise, SocialScan only sends active scan requests to peers within a separate—presumably smaller—tunable SocialDistance radius. This effectively limits the number of nodes from which a clickstream collection attack can be launched to a node's group of trusted peers. Additional privacy considerations associated with SocialScan can be found in Section VII.

IV. DecisionHandler Modules

Fig. 3 shows the states used when an object request is passed through SocialScan. At the core of SocialScan is a DecisionHandler module that contains the majority of the logic around how much data are collected to make a decision

TABLE I
SOCIALSCAN PARAMETERS

PARAMETER	PURPOSE
isConfident	Boolean indicating whether or not DecisionHandler is confident in its decision. If it is not confident yet in its decision, then it will loop and wait for additional data to be collected.
isMalicious	Boolean indicating the current decision by DecisionHandler (True=Malicious, False=Benign)
WaitTime	Number of uSeconds to sleep before calling DecisionHandler again if Confident ==False
doLocalScan	Boolean indicating whether a local scan should be performed on the object
ScanRequestRadius	Maximum SocialDistance to which SocialScan should send directed scan requests (via a SocialCast)
ScanRequestPeers	List of specific Social Peers to which should send directed scan requests (via unicast)

as well as the result of the decision. SocialScan provides the DecisionHandler module all of the available data on the particular object that is being requested, including all result records of previous local scans, peer scans found in Scan Logs, as well as peer scans obtained from directed scan requests. Along with each of the scan records, the DecisionHandler is provided with the scan engine name and date of signature set that created the scan results. The DecisionHandler returns the value of the six parameters shown in table I.

DecisionHandler is thus provided with all possible information on an object and has full control over the behavior of SocialScan. DecisionHandler can decide to make an immediate decision ($isConfident == True$) and either accept or reject the object. Alternatively, DecisionHandler can decide to seek more information and specify a list of peers or a SocialCast distance to send out active scan requests. There is a maximum decision delay time, however, and if the DecisionHandler call loop exceeds this maximum time, then the latest return value of $isMalcious$ is used to either accept or reject the object.

We design the following three interchangeable DecisionHandler modules:

A. Local-only Scanning

The local-only SocialScan DecisionHandler behaves like a traditional single scanner client. This module is designed only for comparison with other DecisionHandler modules.

B. Paranoid Scanning

The Paranoid SocialScan DecisionHandler requires virus scan results from a tunable minimum number of diverse scanners—within a tunable social radius. Each of those scanners must have been updated within specific recent windows of time to be considered valid. Lack of sufficient result diversity as well as any single positive (malicious) result induces the Paranoid SocialScan DecisionHandler module to block the object being requested. Intuitively, this module will likely yield the highest malware detection rates but will also require the greatest latency to certify benign objects.

C. Dynamic Scanning

The Dynamic SocialScan DecisionHandler module adjusts the required diversity of scanners based on the age of the object. The results of our experimentation as well as those of other researchers [1] show that as malicious objects age, their probability of being detected by one or more virus scanners increases. In order to achieve the same probabilities of malware detection, objects that are known to be older can be assessed using a lower diversity of scanner signature sets in comparison with newer objects. The detection of very recent malware—those being less than a few days old—is challenging due to the time required to add a new malware's signature to antivirus signature sets. Protecting systems against recent malware can also be challenging given that new malware commonly attempts to exploit newly exposed and not-yet-patched software vulnerabilities. Software vendors attempt to patch such vulnerabilities quickly [8], but windows of exposure frequently exist. Although it is impossible to certify the age of objects from arbitrary sites on the Internet, certain sites—including those that allow user submitted content—may be considered trustworthy keepers of file creation (upload) and modification times. With the assumption that an object's age can be assessed with a high level of accuracy from a list of trusted sites via the $last - modified$ header of HTTP, the Dynamic SocialScan DecisionHandler module dynamically adjusts the required diversity of scanners based on an object's age. Using the $last-modified$ HTTP header to determine the minimum age of an abject implies that the security of a web site can be trusted to prevent date/time stamp modifications by system users.

V. IMPLEMENTATION

To provide some level of OS independence, we choose to implement SocialScan along with the three DecisionHandler modules as described in the previous section, as a set of Python modules and services on top of an SQLite database[1]. We leave the majority of the implementation details to a separate extended version of this paper.

A. Online Social Network Modules

We implement SocialScan to use proportional levels of reciprocal interactions between peers to approximate altruism and nominal SocialDistance [4] between those peers. We expand upon our earlier work by creating social network analysis modules as part of the SocialScan implementation not only for Facebook, but also for Twitter and standard email (SMTP + IMAP). For Twitter, reciprocal pairs of re-tweets as well as reciprocal pairs of directed tweets are both identified as forms of reciprocal communication. On Facebook, reciprocal

[1]Our implementation of SocialScan is available online [9].

pairs of wall postings are identified as reciprocal communication. With email, reciprocal pairs of email messages are identified as reciprocal communication [10]. Intuitively, there is a generally positive relationship between the frequency of social interaction and the strength of a social tie (e.g., members of the same family will tend to have more social interaction than members of the same political party). We use existing messaging mechanisms with Twitter, Facebook, and email to share user IP addresses as well as to distribute public encryption keys. We make the assumption that the existing messaging mechanisms over these forms of social media are reliable and secure enough for the purpose of IP address and key distribution. For each peer that sends an IP address and public key, SocialScan calculates the multi-hop nominal SocialDistance and maintains the IP, key, and nominal SocialDistance of each peer in a local database. SocialScan also sets a perceived SocialDistance in the database to equal that of the nominal SocialDistance.

B. f2f Messaging

As part of SocialScan, each node runs a Lighttpd web server and provides both XMLRPC services—for messaging, as well as file serving—for Scan Log sharing. When messages from peers are received over XMLRPC into SocialScan's incoming request queue, they are ordered based on the perceived SocialDistance of the peer that sent the messages. For our implementation of SocialScan, we use a fixed request servicing (dequeue) rate. In future implementations, we plan to evaluate adapting request dequeuing rates based on resource (CPU, memory, etc) availability. To distribute Scan Digests SocialScan sends a SocialCast containing the Scan Digest, the URL of the associated Scan Log, and the metadata for the Scan Digest. SocialScan obtains information on the scan engines and signature sets used by peers from the Scan Digest metadata received from each peer.

C. AV Local Scan Handlers

A SocialScan module known as the local scan handler is responsible for interacting with the anti-virus scanner installed on the machine. The local scan handler retrieves a URL, scans it with a specific anti-virus engine, and returns the result to SocialScan. The local scan handler also returns information on the virus signature set used in the scanning. For our SocialScan implementation, we implement scan handler modules for anti-virus packages from the following six vendors: AVG, Avast, Microsoft, Clamwin, Avira, and Kaspersky. Based on existing research [11] on worldwide market share of scanning engines, we believe this set of engines represents around 65% of the global install base of anti-virus software.

D. Browser Request Filtering

To maximize compatibility with a variety of desktop and mobile user agents (browsers), we implement a SocialScan module for the squid caching http proxy server. Applications that make outgoing http requests via squid automatically receive the scanning benefits of SocialScan. Squid feeds SocialScan each url that the browser is attempting to retrieve. SocialScan responds to squid indicating if it wants to allow the browser access the url. SocialScan can also respond with an alternative url if it wants to redirect the browser to different target content (e.g. a warning page). As part of the implementation, we confirm compatibility with three locally installed browsers (IE, Firefox, and Chrome) as well as three mobile browsers (Opera, Firefox, and Android Browser) by configuring the mobile device to proxy http connections through a non-mobile SocialScan enabled squid instance.

VI. Evaluation

In this section we evaluate SocialScan via both implementation and simulation. Using our implementation on the Amazon EC2 cloud, we evaluate SocialScan against its design goal of increasing malware detection accuracy by allowing peers access to diverse scanning of objects retrieved from the web. Given that the ratio of false negatives to false positives in signature-based commercial virus scanners is several orders of magnitude, our current experiments focus on rates of false negatives; we evaluate the detection rate of objects that have been verified to be malware (false negatives).

Malware is continuously changing and adapting in attempts to avoid security software signature sets. For the purpose of evaluating SocialScan, we collect and maintain a large repository of recent malware from the Internet [12].

1) Local vs SocialScan: Using the local DecisionHandler, we run each of six individual scanners through a random selection of approximately 3000 pieces of recent malware. We find similar results as found by Oberheide et al. [1] in that older objects had a higher probability of being detected by malware scanners. Fig. 4 shows the individual scanner results in relation to the age of the malware object being scanned as well as a weighted mean for those scans, with the weight based on the proportional global market share [11] of each scanner. The weighted mean provides a strong indication of the total effectiveness of individual scanners at detecting malware based on the age of the object scanned. Fig. 4 also shows the results of the Paranoid and the Dynamic DecisionHandlers. The Paranoid DecisionHandler uses collaboration with social peers and in this experiment requires responses from each of the six diverse scanners before a decision to allow the object is made. The Dynamic DecisionHandler varies the number of required responses from social peers based on the age of the object being scanned. For 0- to 1-day old malware, this figure highlights a change from 44% to 74% (a 65% increase) in the malware detection accuracy when using the Paranoid and Dynamic DecisionHandlers in comparison with the market-share average of individual scan results.

A. Object Scanning Latency

Even though diverse scans across peers occur in parallel, the aggregation of results in SocialScan requires additional latency for communication overhead. We compare the mean, max, and average latency required across the local scanners with those required by the Paranoid DecisionHandler. We also

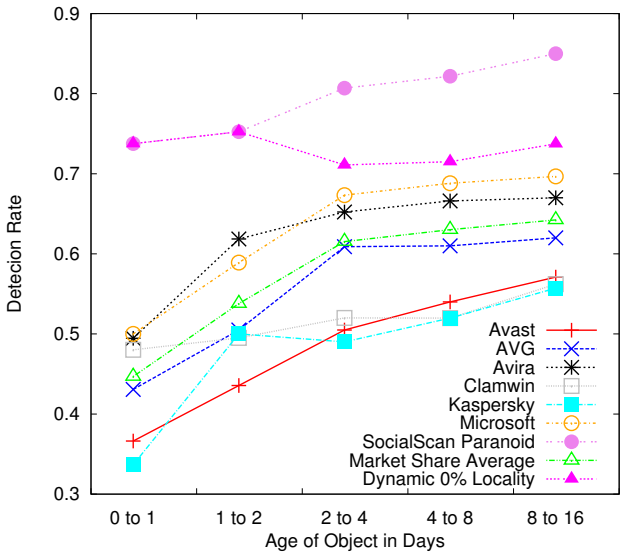

Fig. 4. Local vs SocialScan

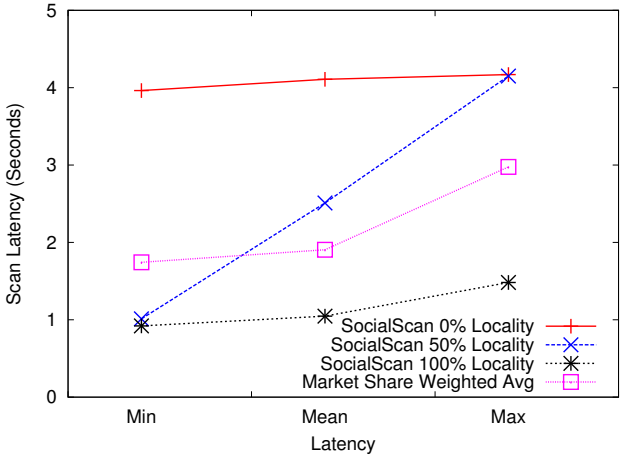

Fig. 5. Latency vs Object Access Locality

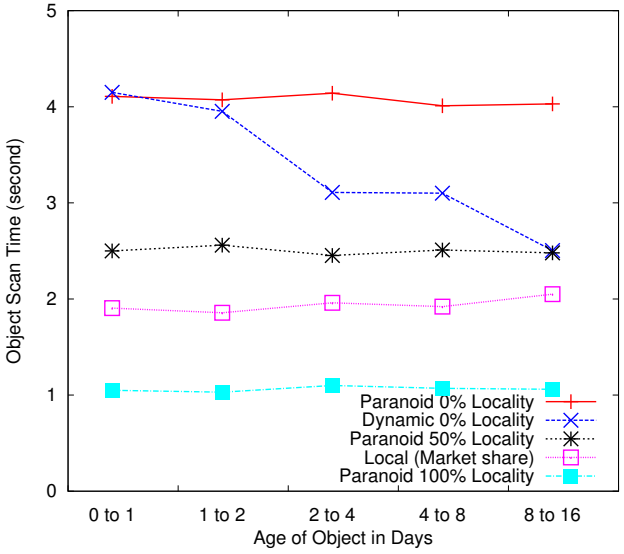

Fig. 6. Latency vs Object Age

test at three different levels of a priori object access and Scan Digest/Log sharing (locality) by peers: 0, 50, and 100%. At the 100% level each of the peers has already accessed an object, scanned it, and shared the result with the inquiring peer via passive Scan Digest/Log sharing. At the 50% level, half of the URLs have previously been scanned with their results shared and the other half are new/distinct to all of the peers in the system. Fig. 5 shows that the Paranoid DecisionHandler at 0% object access/scan locality requires the most time due to the real-time communication and collaboration that must happen between the different nodes. With 100% object access/scan locality, the Paranoid DecisionHandler clearly outperforms even the local scans due to the fact that with 100% object access/scan locality, SocialScan is able to avoid local scans by relying exclusively on previously collected information from peers. For the purpose of this evaluation, we only provide the extremes and do not attempt to approximate or predict levels of object access locality among peers within real-world social networks.

Fig. 6 shows that the Dynamic DecisionHandler requires the greatest scan times for the newest objects. As objects age, the Dynamic DecisionHandler decreases the required scanner diversity and thus the time required to scan objects decreases. Though our current experiments with the Dynamic DecisionHandler only use a 0% object access locality, we expect that testing of greater than 0% object access locality would reduce the scan times proportionally with those results from the Paranoid DecisionHandler because of the passive scan experience caching and sharing within SocialScan.

B. Scanner Availability and Diversity

We evaluate the effectiveness of SocialScan at providing significant scanner diversity to all SocialScan participants and at delivering the highest level of protection to Social Hubs to reduce the probability they will become infected and subsequently serve as valuable launch locations for social malware propagation. Our implementation of SocialScan supports social relationship analysis modules for Twitter, Facebook, and email. To evaluate the potential benefit real social network users would have when using SocialScan, we conduct simulations by using real-world social connectivity metadata extracted from Facebook. We take an anonymized data set of real-world Facebook wall posting from a network of 5,000 Facebook users [13] and feed that data through the SocialScan peer relationship analysis module to identify the nominal SocialDistance between each pair of social peers. We repeat several of our simulations while varying the Social Net Radius—the maximum SocialDistance whereby a peer will consider another node part of its social network and be willing to share resources with that node. As previously described in Section III-A, SocialDistance combines direct levels of altruism between peers with a decay for each hop to approximate indirect levels of altruism. Thus, Social Net

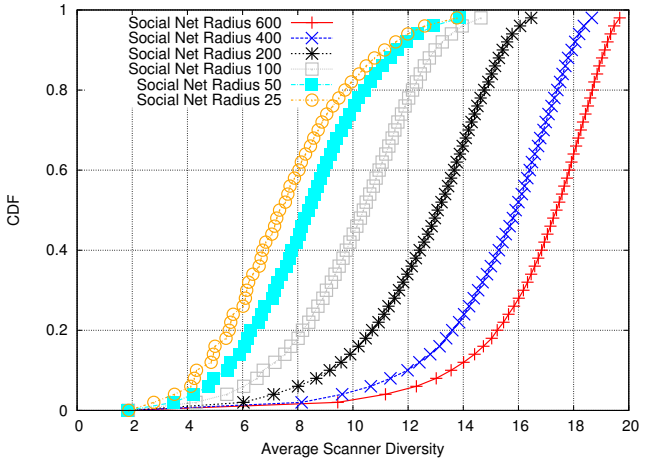

Fig. 7. Scanner Diversity Availability by Social Radius

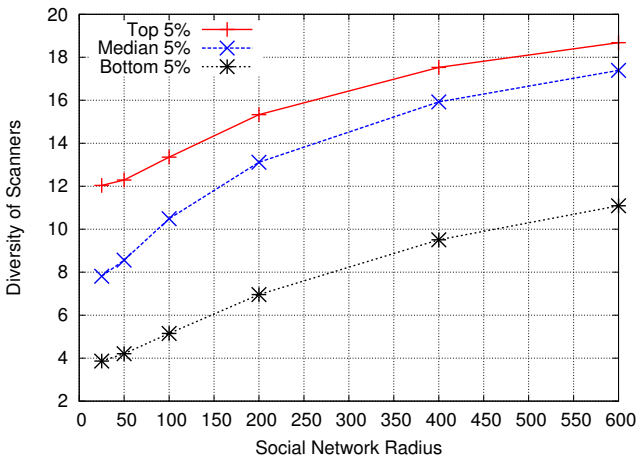

Fig. 8. Scanner Diversity Availability by Social Connectivity

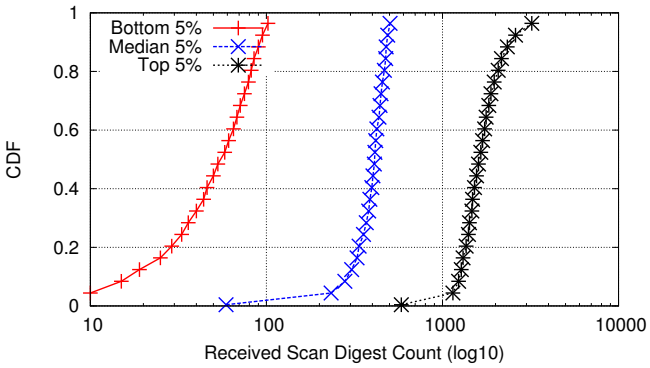

Fig. 9. Scan Digest/Log offers by Social Connectivity

Radius does not directly equate to a hop count between social peers.

1) Diverse Scanner Availability: We assign each of the 5,000 simulated social peers a Facebook identity. For each Social Net Radius tested, we perform 100 simulation runs. On each simulation run we randomly assign each of the 5,000 social peers an anti-malware scanner from a pool of 21 candidate scanners based on the proportional market share of each scanner. These 21 scanners represent 80% of the global market share for anti-malware software. In our simulations we do not consider correlated anti-malware usage among social peers; we leave such considerations to future work. We evaluate the level of scanner diversity that exists within each peer's social network and extract the mean diversity level for each peer across the simulation runs. Fig. 7 provides a CDF for each of the tested social network radius values and shows that even for smaller social network radius values, most peers have access to significant diversity of scanners across their social peers. These findings also imply that even if a significant portion of a user's social peers were unavailable or unwilling to share resources, the remaining peers would typically still be sufficient to a high diversity of scanners.

2) SocialHub Protection – Scanner Diversity: Fig. 8 compares the level of scanner diversity available to three different sub groups of nodes ordered based on levels of social connectivity: The top 250 (95th percentile), the median 250, and the bottom 250 (5th percentile) socially connected nodes. Fig. 8 clearly shows that SocialScan participants offer the highest scanner diversity to the most socially connected nodes.

3) SocialHub Protection – Scan Digests/Logs: We also run a simulation whereby every hour, each peer in the system randomly offers a single Scan Digest/Log pair to one of its peers. The Scan Digest/Log offers are probabilistically granted based on the proportional levels of altruism a peer has towards each of its peers. We run the simulation for 500 virtual hours and then count the total number Scan Digest/Logs received by each node. Fig. 9 shows three CDF lines representing peer sets with different levels of social connectivity—one for each of the same 95th percentile, median, and 5th percentile 250 node peer sets used in the previous simulation. This figure clearly shows that the most socially connected peers receive the highest number of Scan Digest/Log offers and thus they have access to more cached scan results in comparison with peers that have a lower levels of social connectivity. These results show that, as designed, SocialScan is effective at prioritizing allocation of resources among social peers—including levels of scan results sharing and availability of diverse scanners—with social hubs receiving the greatest benefit (and thus level of malware protection).

VII. Additional Privacy Considerations

SocialScan avoids centralized commercial services in favor of a distributed, f2f approach. The target set of users for SocialScan are those who have an existing willingness to relax their privacy requirements when collaborating with trusted and monitored social peers. It is also reasonable to assume that during browsing sessions where privacy is of critical concern, users would not only use their browser's "incognito" or "private mode" but they would also temporarily disable their usage of SocialScan. The two primary areas of concern for privacy for SocialScan users are those of clickstream (object access sequences) and anti-virus (vendor/version) privacy, which we

briefly discuss below.

SocialScan requires sharing of AV Vendor and signature set version information with social peers within a limited Social Distance. Sharing of AV Vendor and signature set information allows a node's trusted social peers to evaluate their own trust in that node's scanning results as well as to allow each peer to properly enforce its own requirements for scanner diversity. The common goal of malware detection should motivate social network participants to disclose to the peers with whom they will be collaborating the AV Vendor and version they are running. Even though adversarial entities running code on nodes within the social network may learn the scan vendor and signature sets of peers running SocialScan, those entities will not be able to fully estimate the diversity of scanners available to each node via peer collaboration.

In our future work, we will use a game-theoretic approach to tackle the privacy problem. Specifically, we will use the concept of *co-privacy* described in the recent work by Domingo-Ferrer [14]. Intuitively, co-privacy makes the preservation of privacy for each individual a goal that also benefits other individuals. In other words, co-privacy strives to ensure that preserving the privacy of a social peer seeking object scans will be in the best interest of the other peers.

VIII. Related Work

For malware detection on distributed devices researchers have proposed a variety [15] [16] [17] [18] [19] of methods based around the basic principle of workload offloading to a cloud-based service. SocialScan is complementary to these approaches in that it provides a decentralized service based on existing resources among social peers for servicing scanning requests.

The PeerSoN [20] system shares the SocialScan goal of facilitating direct p2p interaction between system users. However, PeerSoN lacks application supporting facilities for social relationship extraction from existing OSNs. Cutillo et al. propose SafeBook [21] a system for protecting communication privacy of social networking users via direct p2p interaction. In contrast to SocialScan, SafeBook relies on centralized administration of trust among users via a Trust Information Service (TIS). SocialScan is fully distributed and does not have such requirements for trust in a single centralized service. Both PeerSoN and SafeBook lack experimental implementations for use with real-world applications. Yang et al. propose [22] a structure for information sharing between social network users via direct p2p collaboration. As opposed to SocialScan which leverages existing trust among OSN users to provide f2f malware scanning services, the goal of the system proposed by Yang is to help individuals find relevant content and knowledgeable collaborators who are willing to share their knowledge. Friendstore [23] is a system that leverages existing social relationships to provide p2p distributed backup services. Node selection in Friendstore is limited to a static set of trusted nodes and does not account for variances in trust levels between users (or leverage the potentially valuable altruism of peers outside of that static set of trusted nodes).

IX. Conclusions and Future Work

In this paper, we presented SocialScan, a novel distributed malware detection system which provides f2f scanning of objects with priorities governed by levels of social altruism. We implemented SocialScan and found that with f2f sharing of resources SocialScan significantly enhances the ability of social peers to detect malware. We also showed that SocialScan provides greatly enhanced malware protection to social hubs by ensuring that they receive the most Scan Digest/Log offers and have access to the greatest diversity of scanner engines.

Areas of potential future research include design of mechanisms to increase user privacy, evaluation of the resilience of SocialScan against both innocuous and malicious peer misbehavior, additional incentives for peer participation, multi-agent (device) peers, and addition of execution-based (resource intensive) malware detection to SocialScan.

References

[1] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," in *USENIX Security Symposium*, 2008.

[2] A. Moshchuk, T. Bragin, D. Deville, S. Gribble, and H. Levy, "Spyproxy: Execution-based detection of malicious web content," in *USENIX Security Symposium*, 2007.

[3] L. Liu, S. Chen, G. Yan, and Z. Zhang, "Bottracer: Execution-based bot-like malware detection," in *ISC '08*. Springer-Verlag, 2008.

[4] M. J. Probst, J. C. Park, R. Abraham, and S. K. Kasera, "Socialswarm: Exploiting distance in social networks for collaborative flash file distribution," in *ICNP*, 2010, pp. 263–274.

[5] S. Musil, "Symantec says source code stolen in 2006 hack," http://tinyurl.com/oj5hnyn.

[6] M. H. Hamilton, A. Rousskov, and D. Wessels, "Cache digest specification - version 5," http://tinyurl.com/nfmhh3f.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *Trans. Networking*, vol. 8, June 2000.

[8] E. Zurich, "Security econometrics: The dynamics of (in)security," http://tinyurl.com/npyur9t, 2009.

[9] M. Probst, "Socialscan (f3ds) source repository," http://socialscan.us.

[10] M. S. Granovetter, "The strength of weak ties," *American Journal of Sociology*, vol. 78, 1973.

[11] OPSWAT, "Opswat security industry market share analysis," http://tinyurl.com/5svhwxq, June 2011.

[12] NetPilot, "Clean mx realtime virus database," http://tinyurl.com/5st69g5.

[13] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in Facebook," in *WOSN*, 2009.

[14] J. Domingo-Ferrer, "Coprivacy: towards a theory of sustainable privacy," in *PSD*. Springer-Verlag, 2010.

[15] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian, "Virtualized in-cloud security services for mobile devices," in *MobiVirt*, 2008.

[16] A. Bose, "Propagation, detection and containment of mobile malware," University of Michigan, 2008.

[17] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *HotOS'09*, ser. HotOS'09, 2009.

[18] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid android: versatile protection for smartphones," in *ACM ACSAC*, 2010.

[19] E. Cuervo et. al, "Maui: making smartphones last longer with code offload," in *Mobisys'10*, 2010.

[20] S. Buchegger, D. Schiberg, L. hung Vu, and A. Datta, "Peerson: P2p social networking. early experiences and insights," in *ACM WOSN*, 2009.

[21] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust," *Comm. Mag.*, Dec. 2009.

[22] S. J. H. Yang and I. Y. L. Chen, "A social network-based system for supporting interactive collaboration in knowledge sharing over peer-to-peer network," *Int. J. Hum.-Comput. Stud.*, no. 1, Jan. 2008.

[23] N. Tran, F. Chiang, and J. Li, "Efficient cooperative backup with decentralized trust management," *Trans. Storage*, Sep. 2012.