Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 179 (2007) 17–30
www.elsevier.com/locate/entcs

Building Reputations for Internet Clients

Songjie Wei[1] Jelena Mirkovic[2]

Computer and Information Sciences Department
University of Delaware
Newark, DE 19716, USA

Abstract

We propose a design of a client reputation system that can be used to reduce unwanted traffic in the Internet. Many reputation systems proposed in the trust literature are provider-oriented, but because of different use and adversary models, their techniques are not directly applicable to client reputation systems. We survey the challenges of building client reputations, discuss two different approaches to information collection — a *reporter* and a *monitor* model — and propose their combination that successfully handles major threats to reputation validity.

Keywords: Client reputations, unwanted traffic prevention, Internet reputations, behavior modeling.

1 Introduction

Reputation systems are commonly used to build a reputation of a service provider, which aids a user in selection of a trustworthy provider [1,6,3]. In this paper we discuss a *client* reputation system, which aids service providers in deciding to accept or decline interaction with a given client. Such a system could significantly advance defenses against major security threats, such as intrusions, distributed denial-of-service (DDoS) attacks and worm spread incidents. These threats are extremely difficult to address because the legitimate and malicious traffic look very much alike. A server usually has to process some malicious traffic before it can detect a security problem. This window of vulnerability can lead to a server's compromise (in case of intrusions and worms) or a failure (in case of DDoS). An advance knowledge of a client's trustworthiness, provided by reputations, would help lower a server's risk of compromise and failure.

Client reputations could also be used for traffic prioritization during congestion events. In case of heavy worm and DDoS traffic some packets must be dropped to

[1] Email: weis@cis.udel.edu
[2] Email: sunshine@cis.udel.edu

doi:10.1016/j.entcs.2006.11.033

relieve network congestion. Since legitimate and attack traffic are hard to separate, legitimate users' traffic usually suffers significant collateral damage. Client reputations would help prioritize drops so that traffic from clients with low reputations is dropped first and well-behaved clients experience fewer service disruptions.

A key insight behind client reputations is that a given host tends to be well-administered or poorly-administered over a considerable time, and that hosts that have behaved maliciously in the past warrant a lower trust since they are likely to misbehave in the future. A study of host scanning patterns [16] supports this assumption. It revealed that top 1024 scanners are responsible for 90% of all scans and that large scanners persist over a considerably long time. Blacklisting these repeat offenders would dramatically reduce scan traffic in the Internet. If each server used its own observations to assemble a list of good or bad clients, this list would be stale and incomplete because an average server communicates with a moderately large, slowly-changing circle of clients. Building a collective memory of client behavior by combining observations from multiple servers through reputations has a clear advantage over this isolated approach.

While some techniques from provider reputation systems can be reused in a context of client reputations, there are many unique and complex challenges that demand novel solutions. In this paper we provide a systematic examination of these challenges and propose a specific architecture for building and maintaining client reputations.

1.1 Contributions

This is the first paper that provides a systematic overview of differences between provider and client reputations. We survey challenges that are unique to client reputation systems, assuming a realistic adversary model and an open reputation system where any entity can participate. We also discuss two different architectures for collection of client behavior information that is needed for reputation building: a *reporter* model, where servers rank their experiences with clients, and a *monitor* model where independent observers rank client behavior using observations of its traffic patterns. Finally, we show how a combination of reporter and monitor models can overcome major threats to reputation validity.

2 Collecting Information for Reputation Building

We consider two approaches for collection of client behavior information. The *reporter model* works as a classic reputation system, in which servers who have communicated with a given client rate this client through reports. Reports are submitted to reputation centers that can be managed as a centralized or a distributed service. The *monitor model* engages large Internet service providers that relay traffic between many clients and servers, in monitoring this traffic and building models of clients' behavior. Monitors' vantage locations facilitate correlation of malicious behaviors and easy detection of scanning and worm propagation. Hosts involved in such incidents can be flagged as bad immediately, while other hosts can be flagged

as anomalous if they diverge from their usual behavior and a prolonged divergence can lower their reputation. The monitor model is necessarily centralized, as monitor's locations must be chosen so to observe majority of routes. Monitors must also be trusted to compute reputation scores, and may be used to store and serve reputations.

We denote a server's report or a monitor's observation with a common term *info-item* and we call a reporter or a monitor an *information source*. In both models, reputations could be served to requesting parties in a form of an aggregated score or "raw" — as a history detailing a client's behavior and leaving the score computation to the requester. While the second approach is preferable because it can accommodate individual polices for score calculation, it endangers privacy because info-items leak information about a client's communication patterns. We now propose an anonymization approach that addresses these privacy concerns. We will assume that only bad info-items are generated, and we will discuss shortly whether good info-items would be valuable.

Let an info-item $i = \{a, c, con\}$ be submitted by the information source a about the client c. The info-item specifies more details (context) about the observed malicious activity in the *con* field. We assume that this field is standardized and specifies the type of a security incident (scan, worm, DDoS, anomaly) and some type-specific details, e.g., the range of scanned ports, the port targeted by a worm, the rate, the duration and the type of a DDoS attack, etc. The info-item i is anonymized by the reputation center a follows:

$$anon(i) = \{p, i.c, np, i.con\}; p = pseudonym(i.a), np = pts(i.con) \quad (1)$$

The function $pseudonym(a)$ is a non-reversible function that maps the identity of the information source a into a pseudonym. This mapping must be consistent even if the reputation system is distributed, and can be achieved by applying a private hash function to the identity a using a secret key, which is shared among reputation centers. The *pts* function maps in a standardized manner the type of a security incident and its packet rate, extracted from the context field, into negative points *np* to be added to the reputation score. Reputation users may either use the *np* value specified in the anonymized info-item, or apply their customized *pts* function to the *i.con* field to calculate the update for the client's reputation.

Definition 2.1 *Adversary model*: We assume that an attacker can compromise no more than M other machines, and organize them into a botnet. We will refer to these compromised machines, controlled by one attacker, as *bots*. Given that botnets of $500,000$ nodes have been discovered [10], M could be very large. An attacker can use bots to perform malicious activities in any chosen engagement pattern, and to fully participate in the reputation system by submitting info-items about bots and about other clients.

If good info-items were allowed, bots could file good info-items about each other, boosting their reputations. They could also, at any time, elicit good info-items from legitimate servers (in the reporter model) or from monitors (in the monitor model)

by engaging in legitimate transactions. For these reasons, we conclude that good info-items are meaningless and eliminate them from further discussion. We will assume that a client is good if no bad info-item was filed against it, as proposed in [1].

Reputation systems frequently use the reputation of an information source to judge validity of its reports. In [2] authors propose a reporter model in which users of the reputations vote for or against reports they used, rating their value. Since our adversary model assumes that an attacker has control over a large number of bots, the attacker can ensure a majority vote in any scenario.

Additional drawback of voting in client reputation systems is that a server cannot vote for bad info-items that it has used. If an interaction with a client c was rejected by the server s because of a bad info-item filed by the information source a, s cannot confirm that the activity would have been malicious and thus cannot vote in favor of a.

While we have not thoroughly explored the possible space of voting techniques for client reputations, from this preliminary discussion it appears that voting cannot be sufficiently secured to provide useful input for client reputation building.

3 Definitions and Assumptions

We now list several definitions that set foundations for a client reputation system. For simplicity, we only consider a system that prevents interaction with clients who have participated in scanning, DDoS or worm incidents. This model could be easily extended to penalize other malicious behaviors.

Definition 3.1 *Good client* is a client that has never been an object of a bad info-item. *Bad client* is a client that has been an object of a bad info-item at least once.

Definition 3.2 *Current-bad client* is a client that has been an object of a bad info-item at least once in the recent T_{int} seconds. Similarly, *current-good client* has not been an object of a bad info-item during the recent T_{int} seconds.

Definition 3.3 *Long-term reputation* is a reputation that takes into account all info-items about a client, regardless of the item age. Similarly, *short-term reputation* only takes into account items created during the recent T_{int} seconds.

Definition 3.4 *Reputation use.* Client reputations are used in the following manner: (1) During DDoS attacks and worm spread incidents long-term reputations are used to identify bad clients. Traffic to a DDoS victim or to a worm probe port is prioritized, serving first good clients and then serving or dropping traffic from bad clients. (2) Short-term reputations are used during normal operation to identify current-good and current-bad clients. All traffic from current-bad clients is dropped. Traffic from current-good clients is accepted.

4 Differences Between Provider and Client Reputations

Main differences between provider and client reputation models are the following:

Higher price of a false positive. In a provider reputation model, reputation users are the clients whose goal is to find at least one trustworthy provider. If a provider is mistakenly assigned a low reputation, clients can migrate towards high-reputation providers so client satisfaction remains high. In a client reputation model, service providers are reputation users. Their security goal to avoid malicious clients conflicts with their business goal to serve as many clients as they can. If a client is mistakenly assigned a low reputation score this leads to business loss for providers, so the price of a false positive is high.

Inconsistency of bad behavior. A provider has a consistent good or bad behavior. For example, an eBay seller may frequently deliver damaged goods, or refuse to replace them, a peer in a file-sharing network may always have high delay because it is on a slow link, etc. It thus makes sense to create a provider's reputation by a majority vote and to expect that customers' votes will match. Some reputation systems even use mismatching reports to identify lying customers [1]. Conversely, a malicious client does not need to act maliciously all the time. In fact, security incidents are much more infrequent than normal transactions. If a compromised machine is used daily by its owner for legitimate transactions, and only sporadically by an attacker for malicious activities, providers' majority vote would always result in a good reputation.

Sparse communication patterns. While an average provider is expected to provide service to a diverse customer population, an average client may only communicate with a few providers over a considerably long time. Provider reputation systems frequently use diversity of reporters to build confidence in the aggregated reputation score, e.g., a seller ranked as good by two customers will have a lower reputation than a seller that was ranked as good by a thousand customers, although their mean rank is the same. Information source diversity cannot be used to build reputation confidence in a client reputation system, because this diversity may naturally be low: a malicious client's behavior may affect a single server or be observed by a single monitor.

We identify two sets of conflicting design requirements from the above discussion:

Conflict 1: *Isolated bad info-items must be believed vs. False positives must be low.* A compromised client is expected to commonly invoke a few bad info-items. The only way to lower a bad client's reputation is to believe each bad info-item and reflect it in the reputation score. If reporters or monitors can lie, i.e. file a bad info-item against a good client, false positives will be too high. **Resolution:** Since the monitor model is centralized, it assumes that monitors are trusted and do not lie. In a reporter model we must prevent reporter lying, i.e. each bad report must be somehow *verifiable.*

Conflict 2: *Bad behavior is naturally scarce vs. A bad client may become good when cleaned.* Because bad behavior is scarce, but repeatable, we must remember bad info-items for a sufficiently long time to identify repeat offenders. On the other

hand, a previously bad client that was cleaned and secured should have a way of redeeming itself and regaining a high reputation score. **Resolution:** Our reputation use model from the definition 3.4 provides short-term reputations that are used by servers to accept recently cleaned clients' traffic during normal operation. Their traffic will be dropped during DDoS and worm spread, but this effect will be infrequent, short-lived and limited to these clients' traffic traveling to a certain server (in case of DDoS) or to a certain port (in case of worms). There is also an option that a client redeems itself through human channels, e.g., proving to some trusted entity that it is no longer compromised. While this should be possible, human actions are slow and should not be the only path to client redemption.

5 Reporter Model

We now define how reports are generated in the reporter model.

Definition 5.1 *Report generation*: We assume that a server submits a report after an interaction with a client. Only bad reports are submitted if the interaction was malicious, i.e., if the client has sent a scan, a worm probe or participated in a DDoS attack on the server.

An advantage of the reporter model is that the reputation system can be designed in a fully decentralized manner, using approaches from peer-to-peer reputations systems [1,3], thus no participant needs to be trusted by all others. Another advantage is that a server has a very reliable knowledge if a client has interacted with it in a benevolent or a malicious manner, so false positive reports are not possible.

We identify the following challenges of the reporter model:

Challenge 1: *A malicious reporter may lie about good clients.* We define lying as either submitting an unsolicited bad report (for an interaction that did not occur) or a bad report after a legitimate transaction. Lying would enable malicious reporters to ruin good clients' reputations. **Countermeasure**: Bad reports must be *verifiable*. A commonly used report verification approach in provider reputation systems is to have a *witness* agree by producing a matching report. In client reputation systems, witness model is less feasible because some malicious actions may be isolated and because witness testimonies could be falsely submitted by bots. We believe that the lying challenge cannot be overcome in the reporter model, because of colluding bot actions. We propose in Section 7 a solution for the lying reporter problem in the reporter-monitor model, by using monitors as *verifiers* who vouch that the malicious traffic has been observed by them.

Challenge 2: *An attacker can use spoofing to elicit bad reports about a good client.* These reports are solicited and report a true activity, but they assign the guilt to the wrong party. IP spoofing is a major threat that is largely unhandled in the Internet. **Countermeasure**: There are several approaches to filter spoofed traffic at the edge node [8,12,5], that provide varying degrees of security. These approaches do not filter all spoofed packets but they can identify and drop a majority of them. Another approach is to use only traffic from established TCP connections

for reporting, but this would prevent filing of reports about scanning traffic, unsuccessful worm probes and DDoS attacks via UDP, ICMP or TCP SYN floods, seriously limiting utility of the reputation system. The third approach requires a reporter to reply to scans and worm probes sent to closed ports and to a dark address space (not used by a live machine). If a connection is established, this confirms that the traffic was not spoofed and that it was malicious. Such functionality of interacting with malicious traffic is provided by Honeynets [14], and used by many networks to monitor malicious activities. These three approaches could be combined to provide a layered defense that reduces spoofed traffic.

Challenge 3: *An attacker can fabricate a report from a given server.* **Countermeasure**: A fabricated report will not be verifiable and will be rejected by the reputation system. Because we do not evaluate credibility of information sources via voting, fabricated reports cannot lower credibility of an alleged server. However, if we enforce a quota on the number of reports accepted from a given server in some interval, to control the load of the reputation centers, report fabrication could lead to a DoS attack on the server whose identity was stolen. To prevent this, each server must share a secret with the reputation centers and use this secret to sign its reports.

6 Monitor Model

In this model, monitoring nodes reside on routers and collect observations about the clients from the traffic they relay.

Definition 6.1 *Client behavior* is a record containing statistics about the traffic from and to a given client over some period of time, the client's frequently visited destinations, requested services, and scans sent and received by this client.

Each monitor summarizes a client's behavior from the traffic it relays and uses it to build the client's profile. Because of the distributed monitor locations and routing patterns, each monitor may build a different profile for the same client. Monitors should periodically exchange client profiles to synchronize them. Monitors should be deployed so to observe a significant portion of Internet communications and thus be able to accurately model behaviors of most clients. One possible deployment strategy is to place monitors at core Internet nodes that relay traffic between a large number of source-destination pairs. 50 largest autonomous systems can observe more than 97% of source-destination paths [9].

Some behavior patterns can be identified as malicious instantly, such as high scan traffic to multiple destinations at the same destination port (worm spread), high-volume traffic from many sources to a common destination (DDoS attack), high scan traffic to multiple ports on the same destination (preparation for intrusion), etc. These patterns always trigger a generation of a bad info-item. Monitors further compare current observations of a client's behavior with its profile periodically and record significant differences as anomalies. A client may behave anomalously because its user has changed his network-usage habits, or because it has been com-

promised. Since anomalous behavior does not necessarily signal maliciousness, a prolonged anomalous behavior leads to generation of a *suspicious* info-item. Suspicious info-items are similar to bad info-items; they carry a context field that provides more details about the observed anomaly (e.g., client contacted new destinations on a new service port, client started sending large traffic volume, client has been contacted by another bad client and possibly compromised). We will call a client that has been an object of at least one suspicious info-item, and no bad info-items, a *suspicious client*. Reputation users can devise their own policy how to treat suspicious clients during various security incidents, and based on the context of suspicious info-items.

An advantage of the monitor model is that a low number of deployment points are required to observe a significant portion of Internet traffic. The system is naturally centralized, and since monitors are located in large organizations that are already critical for Internet operation, they should be trusted. Another advantage of a monitor model is that, in addition to client profiling, monitoring large portion of the Internet traffic would be useful for early detection of Internet anomalies, such as heavy scanning activity and worm propagation, or even for discovery of DDoS attack preparation via botnet recruitment.

We next identify challenges faced by a monitor model and propose countermeasures.

Challenge 1: *Client profile integration and update.* Due to the different locations, monitors may profile different behaviors for the same client. The profiles should also be updated continuously with newly observed traffic. **Countermeasure**: A client's profiles from different monitors are periodically integrated using different weights. The weight of each monitor's profile is determined according to the observed traffic volume used to create this profile. Profiles are periodically updated using newly observed traffic if the client's behavior has not been flagged as anomalous.

Challenge 2: *Monitoring overhead.* Monitors need a large memory to store client profiles, and they have to build profiles and detect anomalies online. **Countermeasure**: Monitors should only profile clients that produce sufficient traffic to warrant building a profile. Profiles of clients with similar behaviors should be combined to save space. Our previous research on host clustering using behavior profiles [15] has shown that many hosts behave similarly and can be grouped into large and representative clusters. Host clusters can be used for anomaly detection, while saving memory. It is also possible to build quality profiles from sampled traffic, which can reduce per-packet processing overhead. [15]

Challenge 3: *An attacker can elicit false observations through spoofing.* **Countermeasure**: Core-based filtering techniques, such as [11,8,5], can be used to filter out a majority of spoofed packets. Approaches such as using traffic from established TCP connections for observations are not applicable because a monitor may reside on asymmetric traffic paths and may not be able to detect established connections. The monitor model is thus more vulnerable to spoofing than the reporter model. In section 7.2 we discuss how a combination of these two models can lead to a strong

spoofed traffic countermeasure.

Challenge 4: *Monitors can be bypassed.* While monitors reside on a large number of paths, they can be bypassed if an attacker is aware of their locations. **Countermeasure**: We could deploy monitoring nodes at more points; e.g. an ideal coverage can be achieved if monitors are deployed on the vertex cover of the Internet topology [11]. However, the required number of monitors is large (around 4,000 for the current topology).

7 Reporter-Monitor Model

Both the reporter and the monitor model suffer from serious drawbacks such as lying, false positives and spoofing, and thus cannot provide a reliable reputation input. We now propose the combined *reporter-monitor* model, that addresses these remaining challenges.

7.1 Handling Reporter Lying

A monitor in the reporter-monitor model marks each packet it relays with a secret mark. This mark is bound to the packet, to prevent its stealing and reuse, by using a private hash function to hash a packet's contents and header with the monitor's secret. The mark is placed in the packet's IP header, either in the options field or in the IP identification field, which can be safely overwritten if the packet is not fragmented. Servers file reports about bad clients. Additionally, monitors may collect client behavior statistics, build profiles and generate bad and suspicious reports about a client.

Each report is verified for accuracy by a *verifier*, using the secret mark knowledge. The verifier can be the monitor who placed the mark, or some trusted third party, which knows the monitor's secret. Since monitoring routers relay large traffic volume it would be reasonable to minimize their overhead by delegating verification to other nodes.

When a bad report is submitted to a *verifier*, it examines the context field in the report and challenges the reporter to submit traffic samples chosen by the verifier in a non-predictable manner. For example, if a report's context field was "scan traffic on ports 1—1024" a verifier may request submission of scans for ports 32 and 1011. Samples are verified for authenticity by verifying their secret mark, and the report is accepted and stored in the reputation system if all samples pass the verification. Otherwise the report is discarded. The confidence in the report grows with the number of samples checked, but a verifier's overhead grows also, so there is a tradeoff between performing a thorough check and minimizing verification overhead.

Note that the verification process does not prove that the traffic was malicious, it only verifies that header values in the samples fit the context field in the report. Reported traffic rate can also be verified if monitors change their secrets periodically, e.g. every second. This is highly desirable to prevent reuse of accumulated old traffic by malicious reporters to fabricate new reports. A reported attack with the rate of

Incident	Context	Verification
Scan	Range of ports scanned (R), rate in pps (P), start time, end time	Ask for N chosen samples and verify mark, destination port and SYN flag (if TCP) or request flag (non-TCP), ask for 2P consecutive samples to verify rate.
Worm	Port scanned (p), rate in pps (P), addresses of scanned machines (A) start time, end time	Ask for N chosen samples and verify mark, destination port, SYN flag (if TCP) and destination address, ask for 2P consecutive samples to verify rate
DDoS	Type of attack (e.g., SYN flood) relevant header values for this attack (h), rate in pps (P) start time, end time	Ask for N chosen samples and verify mark, and header values, ask for 2P consecutive samples to verify rate

Fig. 1. Report verification steps

P packets per second can be verified by requiring $2 \cdot P$ consecutive traffic samples to be submitted, and verifying that P of them are marked using the same secret. Figure 1 illustrates verification steps for several context types, for our incidents of interest.

Each monitor must share each new secret with a verifier, which brings additional overhead. We can minimize this overhead if we share a secret and rules for new secret generation, between each monitor node and all verifiers, and we assume that their clocks are loosely synchronized so that the drift between any two clocks is smaller than the secret change interval. A verifier can then calculate all the monitors' past, present and future secrets without any further communication. During a report verification, a verifier needs to try at most three secrets for each monitor (previous, current and future secret, because of loose clock synchronization) before finding the right one for the first sample. After this, the same secret can be used to verify the rest of the samples for this report.

7.2 Handling Spoofing

In Section 5 we proposed a combination of several approaches to handle spoofing. In the reporter-monitor model, monitors should perform some chosen core-filtering approach [11,5,12], and reporters should respond to each scan or a worm probe that they plan to report. For example, if a reporter observes scans on ports 10—24 with a rate of 100 packets per second, it needs to close 200 connections (to verify the rate) and to ensure that at least one scan for each port 10—24 belonged to a connection that was closed. Verification samples should contain all three packets from a TCP handshake and two of them (the SYN and the final ACK) should carry an authentic monitor mark, while sequence and acknowledgment numbers in all three packets should match as described in [13].

The above strategy verifies the absence of spoofing for scan and worm probe traffic, and it could also be used for DDoS flash-crowd type attacks because they require a malicious client to establish a TCP connection with the server. However, there are many DDoS attacks that can be successful without an established TCP connection, e.g.., UDP flood, ICMP flood, TCP SYN flood. Such DDoS traffic may be spoofed and cannot be verified using our proposed technique. To differentiate

between reports that can or cannot prove the absence of spoofing in the reported incident, we add a *no-spoofing* flag to each report. This flag is stored along with the report and served to requesting parties.

7.3 Aggregating Reports Into Reputations

Reports must be aggregated into a reputation score, either by the reputation system or by reputation users. We now propose an aggregation method assuming the reporter-monitor model and that only bad and suspicious reports are generated. Bad reports that can be elicited via spoofed traffic will have a no-spoofing flag reset. Our aggregation rules ensure that such reports facilitate traffic prioritization during congestion events such as worm spread and DDoS attacks, but they cannot harm good clients during normal operation.

Definition 7.1 *Reputation use (revised).* Client reputations are calculated using the following equation:

$$rep(c) = \begin{cases} R, & \text{if } count(ubrns) = 0 \text{ and } count(ubrys) = 0 \\ \sum_{r \in brns} r.pts, & \text{if } count(brns) > 0 \\ \sum_{r \in brys} r.pts \cdot d, & \text{if } count(brns) = 0 \text{ and } count(brys) > 0 \end{cases} \quad (2)$$

where R is a default reputation value, assigned initially to a client, $brns$ is a set of bad reports with no-spoofing flag set, $brys$ is a set of bad reports with no-spoofing flag reset, $count(\cdot)$ counts elements in the set and d is a positive discount factor, $d \ll 1$. The calculation yields positive reputations for good clients, large negative reputations for bad clients whose maliciousness and identity can be verified, and small negative reputations for bad clients whose bad reports contain no-spoofing flag reset. Note that suspicious reports are not used for reputation calculation but are served raw to requesting parties.

Reputations are used to prioritize traffic in the following manner:

- During DDoS attacks and worm spread incidents calculate long-term reputations using all bad reports for a client and the formula 2. Assign the highest priority to clients with positive reputation values, then serve the traffic in the following order of decreasing priorities: (1) traffic from clients with a small negative reputation, (2) traffic from suspicious clients. Reject traffic from clients with a large negative reputation.
- During normal operation calculate short-term reputations using observations during last T_{int} seconds only, and the formula 2. Accept traffic from clients with positive reputations. Use context information from reports to make a policy decision about suspicious clients and clients with small negative reputations (e.g., if a context indicates a recent communication with a compromised client we may want to reject this communication). Reject traffic from clients with a large negative reputation.

8 Cost

The proposed reputation system requires placement of monitors at large ISPs. A minimal functionality required from monitors is marking traffic to support verification process and filtering of spoofed packets. Packet-marking is a low-cost operation, while spoofed traffic filtering requires a monitor to build a table that links each source with some chosen parameter value and to perform per-packet checks of the packets' parameters against the stored values in the table.

The rest of the reputation system contains reporters, verifiers and reputation centers, and can be organized in a distributed manner. Servers would thus contact local reputation centers to submit reports and retrieve reputations, while reputation centers would periodically talk to exchange recent reports. Reputation centers can be organized into a peer-to-peer network to facilitate report exchange and existing approaches from provider reputation systems can be applied to ensure that compromise of a reputation center cannot jeopardize credibility of client reputations.

Communication overhead for report submission may be large in case of large-scale security incidents such as worm spread, so reputation centers may be overwhelmed. To control this overhead, we propose that a reporter aggregates all its reports within some interval into a combined report, and files only this report at the end of the interval. The combined report size may also be limited, e.g., by requesting that a reporter choose reports that describe top N security incidents in the previous interval.

Reputation scores or recent anonymized reports can be periodically downloaded by reputation users. These downloads can be scheduled to minimize congestion and they can be retrieved from a local reputation center. Since a client is presumed good in absence of bad reports, reputation users need only store identities of bad clients. Given that the largest botnet up to date contained half a million bots [10], we estimate the cost of such storage to several million records.

The reputation system must be protected from report and reputation fabrication through standard cryptographic means. Each reporter must exchange a shared or a public key with its local reputation centers and sign each report with this key. A reputation center must sign each reputation update with its private key and local servers must possess the corresponding public key to verify the signature. It is likely that an extensive key exchange infrastructure would be needed to support secure operation of the client reputation system. Existing key exchange mechanisms, proposed for provider reputation systems, can be reused in this novel context.

9 Related Work

In [2] Allman et al. propose an architecture for behavioral history that could be applied to "actors" such as Internet hosts, mail servers, mail addresses, Web identities, etc. The goal of this system is to build an audit trail of an actor's behavior that can be used to form a policy at the traffic receiver's side and suppress unwanted traffic. While the proposed system builds client reputations, its architecture is dis-

cussed at a very high level, and the design leaves many opportunities for attackers to trick the system. For instance, authors propose that a report is valued higher if other reporters have observed similar behavior or if they voted that the report was useful to them. We discussed in Section 4 why it is not reasonable to require matching observations for all malicious behaviors, and in Section 2 how any kind of voting can be misused by an attacker who controls a bot network. Our verifier functionality requires minimal processing by the routers (marking packets) and enables report confirmation by a separate trusted entity in possession of the monitor's secret, while [2] proposes that reports be verified by witnesses that store packets and retrieve them on demand. Drawbacks of packet recording approach are: a significant memory requirement, short life of records and a required participation of routers in confirming reports. Finally, we consider the major threat posed by IP spoofing to client reputation system, and propose several approaches to alleviate this threat, while [2] discusses spoofing only superficially.

In [1], Aberer and Despotovic propose a decentralized model for trust management in P2P networks. Their distributed report storage architecture could be reused for a client reputation system, but the proposed trust calculation, that values matching reports higher, is not applicable to client reputations because of inconsistency of bad behavior.

In [7], Hou et al. compute peer reputations based on satisfactory and unsatisfactory transactions; this last category is similar to the bad reports in our system. Authors do not consider cheating which is a major threat in the open client reputation system we proposed, and they assume that every peer rates every other peer, while in our system reporters cannot be rated through voting due to a strong adversary model.

In [3] authors propose a reputation system integrated with the Gnutella P2P network where participants use peer votes to discover trustworthy resource providers. Authors consider cheating via creation of false identities, and propose to discover clusters of such identities via grouping peers by their IP address. Proposed mechanisms do not apply to our attacker model, which enables bots to vote using their real identities. In [4] authors extend the peer reputation system with resource reputations. This helps establishment of more reliable provider reputations but is not directly applicable to client reputation systems.

10 Conclusions

We have proposed a client reputation system that could be used to reduce unwanted traffic in the Internet. Such a system faces a unique set of challenges that we have surveyed in this paper and proposed countermeasures. While much work remains to be done in designing a practical, secure and usable client reputation system, we believe that our proposed reporter-monitor model has provided a good first step in this direction.

References

[1] Aberer, K. and Z. Despotovic, *Managing Trust in a Peer-2-Peer Information System*, In Proceedings of the Tenth International Conference on Information and Knowledge Management, 2001.

[2] Allman, M., E. Blanton and V. Paxson, *An Architecture for Developing Behavioral History*, In Proceedings of SRUTI 2005, pp 45-51.

[3] Damiani, E., S. De Capitani di Vimercati, S. Paraboschi and P. Samarati, *Managing and Sharing Servents' Reputations in P2P Systems*, In IEEE Transactions on Knowledge and Data Engineering, vol. 15, n.4, July/August 2003, pp. 840-854.

[4] Damiani, E., S. De Capitani di Vimercati, S. Paraboschi, P. Samarati and F. Violante, *A Reputation-based Approach for Choosing Reliable Resources in Peer-to-Peer Networks*, In Proc. of the 9th ACM Conference on Computer and Communications Security, Washington, DC, USA, November 17-21, 2002.

[5] Duan, Z., X. Yuan and J.Chandrashekar, *Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates'*, Proceedings of INFOCOM'06, April 2006.

[6] EBay, *EBay Feedback Forum*, http://pages.ebay.com/services/forum/feedback.html.

[7] Hou, M., X. Lu, X. Zhou and C. Zhan, *A trust model of p2p system based on confirmation theory*, ACM SIGOPS Operating Systems Review, Volume 39 Issue 1, January 2005.

[8] Jin, C., H. Wang and K. Shin, *Hop-count filtering: an effective defense against spoofed DDoS traffic*, Proceedings of the 10th ACM conference on Computer and communications security, 2003.

[9] Mirkovic, J., N. Jevtic and P. Reiher, *A Practical IP Spoofing Defense Through Route-Based Filtering*, University of Delaware CIS Department Technical Report CIS-TR-2006-332.

[10] News, Z., *'Botmaster' pleads guilty to computer crimes*, http://news.zdnet.com/2100-1009_22-6030270.html.

[11] Park, K. and H.Lee, *On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets*, In Proceedings of SIGCOMM 2001.

[12] Perrig, A., D. Song and A. Yaar, *StackPi: A New Defense Mechanism against IP Spoofing and DDoS Attacks*, Carnegie Mellon University Technical Report, CMU-CS-02-208, February 2003.

[13] Postel, J., *Transmission Control Protocol*, IETF RFC 793.

[14] Spitzner, L., *Honeypots: Tracking hackers*, Addison-Wesley Publishing, 2002.

[15] Wei, S., J. Mirkovic and E. Kissel, *Profiling and Clustering Internet Hosts*, Proceedings of the 2006 International Conference on Data Mining, June 2006.

[16] Yegneswaran, V., P. Barford and S. Jha, *Global Intrusion Detection in the DOMINO Overlay System*, Proc. the Network and Distributed Security Symposium (NDSS) 2004.