Privacy-Preserving Sharing and Correlation of Security Alerts

Patrick Lincoln* Phillip Porras† Vitaly Shmatikov‡

SRI International

lincoln@csl.sri.com porras@sdl.sri.com shmat@csl.sri.com

Abstract

We present a practical scheme for Internet-scale collaborative analysis of information security threats which provides strong privacy guarantees to contributors of alerts. Wide-area analysis centers are proving a valuable early warning service against worms, viruses, and other malicious activities. At the same time, protecting individual and organizational privacy is no longer optional in today's business climate. We propose a set of data sanitization techniques that enable community alert aggregation and correlation, while maintaining privacy for alert contributors. Our approach is practical, scalable, does not rely on trusted third parties or secure multiparty computation schemes, and does not require sophisticated key management.

1 Introduction

Over the past few years, computer viruses and worms have evolved from nuisances to some of the most serious threats to Internet-connected computing assets. Global infections such as Code Red and Code Red II [21, 40], Nimbda [30], Slammer [20], MBlaster [18], and MyDoom [17] are among an ever-growing number of self-replicating malicious code attacks plaguing the Internet with increasing frequency. These attacks have caused major disruptions, affecting hundreds of thousands of computers worldwide.

Recognition and diagnosis of these threats play an important role in defending computer assets. Until recently, however, network defense has been viewed as the responsibility of individual sites. Firewalls, intrusion detection, and antivirus tools, are, for the most part, deployed in the mode of independent site protection. Although these tools successfully defend against low or moderate levels of attack, no known technology can completely prevent large-scale concerted attacks.

There is an emerging interest in the development of Internet-scale threat analysis centers. Conceptually, these centers are data repositories to which pools of volunteer networks contribute security alerts, such as firewall logs, reports from antivirus software, and intrusion detection alerts (we will use the terms *analysis center* and *alert repository* interchangeably). Through collection of continually updated alerts across a wide and diverse contributor pool, one hopes to gain a perspective on Internet-wide trends, dominant intrusion patterns, and inflections in alert content that may be indicative of new wide-spreading threats. The sampling size and diversity of contributors are thus of great importance, as they impact the speed and fidelity with which threat diagnoses can be formulated.

We are interested in protecting sensitive data contained in security alerts against malicious users of alert repositories and corrupt repositories. The risk of leaking sensitive in-

*Partially supported by ONR grants N00014-01-1-0837 and N00014-03-1-0961 and Maryland Procurement Office contract MDA904-02-C-0458.

†Partially supported by ARDA under Air Force Research Laboratory contract F30602-03-C-0234.

‡Partially supported by ONR grants N00014-01-1-0837 and N00014-03-1-0961.

formation may negatively impact the size and diversity of the contributor pool, add legal liabilities to center managers, and limit accessibility of raw alert content. We consider a three-way tradeoff between privacy, utility, and performance: privacy of alert contributors; utility of the analyses that can be performed on the sanitized data; and the performance cost that must be borne by alert contributors and analysts. Our objective is a solution that is reasonably efficient, privacy-preserving, and practically useful.

We investigate several types of attacks, including dictionary attacks which defeat simple-minded data protection schemes based on hashing IP addresses. In particular, we focus on attackers who may use the analysis center as a means to probe the security posture of a specific contributor and infer sensitive data such as internal network topology by analyzing (artificially stimulated) alerts. We present a set of techniques for sanitization of alert data. They ensure secrecy of sensitive information contained in the alerts, while enabling a large class of legitimate analyses to be performed on the sanitized alert pool. We then explain how trust requirements between the alert contributors and analysis centers can be further reduced by deploying an overlay protocol for randomized alert routing, and give a quantitative estimate of anonymity provided by this technique. We conclude by discussing performance issues.

2 Related Work

Established Internet analysis centers, such as DShield [34] and Symantec's DeepSight [32] gather alerts from a diverse population of sensors. For example, in April 2003, DShield reported a contributor pool of around 41,000 registered participants and around 2000 regular submitters, who submit a total of 5 to 10 million alerts daily [7]. These centers proved effective in recognizing short-term inflections in alert content and volume that may indicate wide-scale malicious phenomena [39], as well as the ability to track important security trends that may allow sites to better tune their security postures [31]. Other research has shown how to use distributed security information to infer Internet DoS activity [22], and how to improve the speed and accuracy of large-scale multi-enterprise alert analysis centers [38].

Alert sharing communities have not yet enjoyed wide-scale adoption, in part due to privacy concerns of potential alert contributors and managers of community alert repositories. Raw alerts may expose site-private topological information, proprietary content, client relationships, and the site's defensive capabilities and vulnerabilities. With this in mind, established systems suppress sensitive alert content before it is distributed to analysis centers (*e.g.*, field suppression is a configurable option in DShield's alert extraction software). Even with these measures, organizations such as DeepSight and DShield must be granted a substantial degree of trust by the alert producers, since suppression and anonymization must be balanced against the need to maintain the utility of the alert.

2.1 Packet trace anonymization

Several approaches have been proposed for anonymization of Internet packet traces [25, 36, 24]. For example, Pang and Paxson proposed a high-level language and tool [24] as part of the Bro package, enabling anonymization of packet header and content. They are interested in wide-scale network traces such as FTP sessions, while our application is alert management. Further, we examine strategies that mitigate dictionary attacks from adversaries who can stimulate and then observe alert production within the target's site.

2.2 Database obfuscation

The database community has examined the problem of mining aggregate data while protecting privacy at the level of individual records. One approach is to randomly perturb the values in individual records [1, 2] and compensate for the randomization at the aggregate level. This approach is potentially vulnerable to privacy breaches. If a data item is repeatedly submitted and perturbed (differently each time), much information about the original value can be inferred. In our context, an attacker could intentionally probe the same IP address using the same attack strings. If the (randomly perturbed) reports of the attack are disambiguated from other alerts based on the attack's unique statistical aspects, the attacker can use them to learn important details of the original alert.

2.3 SMC schemes

Consider two or more parties who want to perform a joint computation, but neither party is willing to reveal its input. This problem is known as Secure Multiparty Computation (SMC). It deals with computing a probabilistic function in a distributed system where each participant independently holds one of the inputs, while ensuring correctness of the computation and revealing no information to a participant other than his input and output.

There exist general-purpose constructions that convert any polynomial computation to a secure multiparty computation [37]. Recent work has considerably improved the efficiency of such computations when an approximate answer is sufficient [13]. Applications include privacy-preserving data classification, clustering, generalization, summarization, characterization, and association rule mining. Clifton *et al.* [8] present methods for secure addition, set union, size of set intersection, and scalar product. Lindell and Pinkas [19] propose a protocol for secure decision tree induction, consisting of many invocations of smaller private computations such as oblivious function evaluation. Unfortunately, the cost of even the most efficient SMC schemes is too high for the purpose of large-scale security alert distribution.

3 Format of Security Alerts

Network data collected to support threat analysis, fault diagnosis, and intrusion report correlation may range from simple MIB statistics to detailed activity reports produced by complex applications such as intrusion or anomaly detection systems. So far, we have used the term *security alert* loosely to refer to site-local activity produced by a network security component (*sensor*) as it reports on observed activity or upon an action it has taken in response to observed activity. A security alert can represent a very diverse range of information, depending on the type of the security device that produced it. In this section, we consider the typical content of security alerts from the three primary types of alert contributors used in the context of Internet-scale threat analysis centers.

Firewalls reside at the gateways of networks, and contribute reports that indicate "deny" and "allow" actions for traffic across the gateway boundary. Most typically, firewalls contribute alerts flagging incoming packets that were denied. Volume, port, and source distribution patterns of such packets provide significant insight into the probe and exploit targets of malicious systems, new attack tools, and self-propagating malicious applications.

Intrusion detection systems include network- and host-based systems, and may employ misuse or anomaly detection. Unlike firewalls, intrusion detection reports may represent a wide variety of event types, and can report on anomalous phenomena that span arbitrarily long durations of time or events.

Antivirus software reports email- and file-borne virus detection on individual hosts. Reports include virus type, infection target, and the response action, which is typically to clean or quarantine the infection.

Table 1 summarizes the fields that constitute a typical firewall (FW), intrusion detection (ID), or antivirus (AV) security alert in its raw form, prior to data sanitization.

4 Threat Model

To support collaborative threat analysis, the alert repository will be published, at least partially, and thus made available to the at-

Source_IP	FW,ID	Typically refers to the source IP address of the machine that initiated the session or transferred the transaction that caused the alert to fire. In IDS alerts, this field may represent the victim, not the attacker, since some systems alert upon an attack reply rather than request.
Source_Port	FW,ID	Source TCP or UDP port of the machine that initiated the session or transferred the transaction that caused the alert to fire.
Dest_IP	FW,ID,AV	Typically refers to the destination IP address of the machine that initiated the session or transferred the transaction that caused the alert to fire. In AV systems, Dest_IP can identify the machine in which the infection is discovered.
Dest_Port	FW,ID	Destination TCP or UDP port of the machine that initiated the session or transferred the transaction that caused the alert to fire.
Protocol	FW,ID	Protocol type (*e.g.*, UDP, TCP, ICMP).
Timestamp	FW,ID,AV	May incorporate incident start time, end time, incident report time.
Sensor_ID	FW,ID,AV	May incorporate the brand and model of the sensor and a unique identifier for the individual instantiation of the sensor.
Count	FW,ID,AV	Often used to represent some notion of repeated activity, either at the alert or event (*e.g.*, packet) level.
Event_ID	FW,ID,AV	Uniquely defines the alert type for the given sensor.
Outcome	FW,ID,AV	Reports the status or disposition of the reported activity. For firewalls, it may report whether the log entry was associated with an allow or deny rule. For AV, it may indicate infection disposition (*e.g.*, Symantec's AV indicates whether the infected file is cleaned or quarantined). Outcome fields for IDS tools are highly vendor-specific.
Captured_Data	ID	Some IDS sensors have the ability to report part or all of the data content in which the alert was applied.
Infected_File	AV	Antivirus logs include the identity of the file that was infected.

Table 1: Summary of security alert content.

tacker. In the worst case, the adversary may be able to compromise the alert repository and gain direct access to raw alerts reported to that repository. It is thus very important to ensure that alerts are reported in a sanitized form that preserves privacy of sensitive information about the producer's network. In this section, we outline the goals of a typical attacker and the means he or she may employ to subvert our alert sharing scheme.

4.1 Sensitive fields

IP addresses. Any field that contains an IP address such as Source_IP or Dest_IP is sensitive, since it reveals potentially valuable information about the internal topology of the network under attack. Knowing the relationship between IP addresses and various types of alerts may allow the attacker to track propagation of the attack through a network which is not normally visible to him (*e.g.*, located behind a firewall). Even though the Source_IP field is usually associated with the source of the attack, it may (a) contain the address of an infected system on the internal network, or (b) identify organizations that have a legitimate relationship with the targeted network. For example, the attacker may be able to discover that attacking a particular system in organization A leads to alerts arriving from a sensor within organization B with A's address in the Source_IP field, and thus learn that there is a relationship between the two organizations.

Popular intrusion detection systems such as Snort [28] include rules that are highly prone to producing false positives, while other rules simply log security-relevant events that are not specifically associated with an attack. An attacker who is aware of such behavior can closely analyze the source IP addresses of these alerts to gain a sense of the sites with which the producer regularly communicates.

Captured and infected data. Data contained in Captured_Data and Infected_File fields are extremely sensitive. File names, email addresses, document fragments, pieces of IP addresses, application-specific data and so on may leak private information stored on infected systems and reveal network topology or site-specific vulnerabilities.

4.2 Sensitive associations

The attacker may use certain associations between the fields of a security alert to learn the security posture of the producer site.

Configurations. Sensitive information includes the site's set of network services, protocols, operating systems, and network-accessible content residing within its boundaries. While some of this information may be revealed through direct interactions with external systems, the breadth of probing can be monitored and controlled by the target site. Associations between security alert fields that could potentially lead to undesirable disclosures include [Source_IP, Source_Port, Protocol] and [Dest_IP, Dest_Port, Protocol].

Site vulnerabilities. Revealing the disposition of unsuccessful attacks may be undesirable. Associations between alert producers and the Sensor_ID, Event_ID and Outcome fields may potentially lead to such disclosures.

Defense coverage. Sites may not want to reveal their detection coverage, including information about versions and configurations of security products that are operating within their boundaries. Attacks and probes mounted against a site with the intention of observing, potentially through indirect inference, which sensors are running and their alert production patterns, would seriously impact the site's security posture. Associations between alert producers and the Sensor_ID and Event_ID fields are thus sensitive.

In current practice, these sensitivities are handled in a variety of ways. Sensitive fields are often suppressed at the alert producer's site before the alert is forwarded to a remote alert repository. For example, the DShield alert extractor provides various configuration options to suppress fields and an IP blacklist that allows a site to suppress sensitive addresses. The second approach is to apply cryptographic hashing to fields, allowing equality checks while maintaining a degree of content privacy (this approach may be vulnerable to dictionary attacks, as explained below). The third approach is simply to trust the alert repository with ensuring that neither content nor indirect associations be openly revealed.

4.3 Potential attacks

We describe several threats faced by any alert sharing scheme, in the order of increasing severity. The attacker may launch attacks of several types simultaneously.

Casual browsing. Alerts published by a repository may be copied, stored and shared by any Internet user, and are thus forever out of control. The mildest attack is casual browsing, where a curious user looks for familiar IP prefixes and sensor IDs in the published alerts. This attack is easy to defend against, *e.g.*, by hashing all sensitive data.

Probe-response. A determined attacker may attempt to use the alert repository as a verification oracle. For example, he may target a particular system and then observe the alerts published by the repository to determine whether the attack has been detected, and, if so, how it was reported. By comparing IP addresses contained in the reported alert with that of the targeted system, the attacker may learn network topology, sensor locations, and other valuable information.

Dictionary attacks. The attacker can precompute possible values of alerts that may be generated by the targeted network, and then search through the data published by the repository to find whether any of the actual alerts match his guesses. This attack is especially powerful since standard hashing of IP addresses does not protect against it. For example, the attacker can simply compute hashes for all 256 IP addresses on the targeted subnet and check the published alerts to see if any of the hash values match. Using semantically secure encryption on sensitive fields is sufficient to foil dictionary attacks, but such encryption also makes collaborative analysis infeasible because two encryptions of the same plaintext produce different ciphertexts with overwhelming probability. A polynomially-bounded analyst cannot feasibly perform equality comparisons unless he knows the key or engages in further interaction with the alert producer.

Alert flooding. If the repository publishes only the highest-volume alerts (or those satisfying any other group condition), the attacker may target a particular system and then "flush out" the stimulated alert by flooding the repository with fake alerts that match the expected value of the alert produced by the targeted system. This involves either spoofing source addresses of legitimate sensors, setting up a bogus sensor, or taking over an existing sensor. Flooding will cause the repository to publish the real alert along with the fakes. The attacker can discard the fakes and analyze the real alert.

Repository corruption. Finally, the attacker may deliberately set up his own repository or take control of an existing repository, perhaps in a manner invisible to the repository administrator. This attack is particularly serious. It eliminates the need for alert flooding and aggravates the consequences of probe-response, since it gives the attacker immediate access to raw reported alerts, as well as the ability to determine exactly (*e.g.*, by inspecting incoming IP packets) where the alert has arrived from. We describe several partial solutions in section 6. Solutions based on sophisticated cryptographic techniques such as oblivious transfer [26] currently appear impractical. They provide better theoretical privacy at the cost of an unacceptable decrease in utility and performance, but the balance may shift in favor of cryptography-based solutions with the development of more practical techniques.

5 Alert Sharing Infrastructure

To enable open collaborative analysis of security alerts and real-time attack detection, we propose to establish alert repositories which will receive alerts from many sensors, some of them public and located at visible network nodes and other hidden on corporate networks deep behind firewalls. Achieving this requires a robust architecture for information dissemination, ideally with no single point of failure (to provide higher reliability in the face of random faults and outages), no single point of trust (to provide stronger privacy guarantees against insider misuse in any one organization), and few if any leverage points for attackers.

The core of the proposed system is a set of repositories where alerts are stored and accessed during analysis. Each repository is very simple: it accepts alerts from anywhere, strips out source information, and publishes them immediately or after some delay. There is no cryptographic processing and no key management (unless the repository performs re-keying — see section 6.2). As described in section 6.3, multiple repositories make it more difficult for the attacker to infer the source of sanitized alerts. The repositories may share alerts, but they are not required to be synchronized, thus not every alert will be visible to every analysis engine. For performance reasons, analysis engines normally interact with a single repository or mirror site.

Figure 1 shows the major data flows among a small set of sensors, producers, repositories, and analysis engines. The sensor trapezoids consist of firewalls, intrusion detection systems, antivirus software, and possibly other security alert generators. The producer boxes represent local collection points for an enterprise or part of an enterprise. These boxes perform the sanitization steps such as hashing IP addresses, and are controlled by the reporting organization. The repository cylinders represent public or semi-public databases containing reported data. A repository may be controlled by a producer or by an analysis organization. The analysis diamonds represent analysis services which process the published alerts for historical trends, event frequency changes, and other aggregation or correlation functions.

An enterprise (such as a major research lab famed for computer security research) may be sensitive to public disclosure of possible attacks, and wish to keep private even the volume of alerts it generates. As described in section 6.3, the repositories can optionally form a randomized alert routing network. Although we have not implemented this feature, randomized routing can provide strong anonymity guarantees for alert sources. A

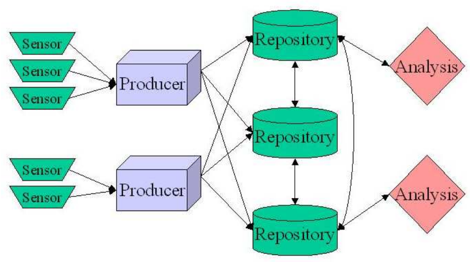

Figure 1: Data flows in alert processing.

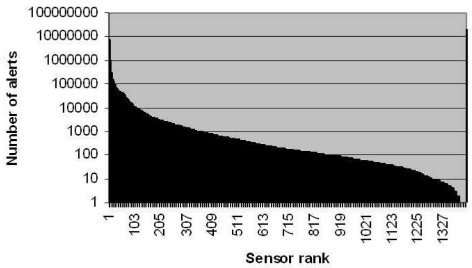

Figure 2: Alert volume per sensor (semi log scale). Data courtesy DShield.

repository may also be configured so that only events whose volume exceeds a certain threshold are published. This will have relatively little impact on historical and inflection analysis (see section 7), but may disable identification of stealth attacks associated with low alert volumes.

As shown in figure 2, sensors vary greatly in the volume of alerts they produce in a given day, but the total alert volume is substantial. This graph depicts the number of alerts produced on a single day by 1,416 sensors reporting to DShield. At the high end, over 7 million alerts were produced by one firewall, apparently experiencing a certain DoS-like attack. Several other sensors were near or above a million alerts. The median sensor produced only 177 alerts.

The total alert volume of 19,147,322 alerts reported on that day, across a total of 1,416 different sensors from many organizations spread over a wide geographic area, constrains practical implementation choices. In particular, secure multiparty computation (SMC) approaches (see section 2.3), and many privacy-preserving data mining techniques add impractical levels of overhead to alert analysis. With over a thousand reporting sensors, naive SMC approaches would require tremendous network bandwidth and unsupportable CPU or cryptographic coprocessor performance for even moderate levels of analysis query traffic. It *is* possible that special-purpose SMC schemes developed specifically for this problem would prove more practical. In this paper, we propose simple solutions which enable a broad set of analyses on sanitized alerts that would normally require raw alert data.

6 Alert Sanitization

We propose several techniques that are used in combination to protect the alert sharing infrastructure from threats described in section 4. Some of the mechanisms are "heavier" than others and impose higher communication and computational requirements on alert contributors. On the other hand, they provide better protection against serious threats such as complete corruption of the alert repository. The exact set of techniques may be selected by each organization or contributor pool individually, depending on the level of trust they are willing to place in a particular repository or set of repositories.

6.1 Design requirements

We do not consider solutions that require alert sources to trust the repository with protecting privacy of the reported data. In the context of completely open public repositories, as opposed to trusted services such as DeepSight [32] and DShield [34], such solutions are both impractical (a commercial enterprise is unlikely to trust an open repository to be careful with business secrets) and dangerous for the repository operator, as she may be exposed to legal liability if the repository is attacked and private alert data compromised.

We also rule out solutions that require sharing of secret keys between sensors. An obvious solution might involve encrypting sensitive data with a common key to enable alert comparison by infrastructure participants, while hiding the data from a casual observer. This approach may solve the corrupt repository problem, but it is vulnerable if the attacker signs up as a participant, gains access to the common key, and breaks privacy of alerts generated by *all* other participants.

Finally, solutions that require multiple producers to collaborate and/or interact to protect a single alert are impractical in our context. Given the volume of alerts, especially when the network is under attack, the communication overhead is likely to prove prohibitive. This eliminates mechanisms based on threshold cryptography [11, 14] such as proactive security [15, 6], and secure multiparty computation (see section 2.3) even though they are secure if a subset of participants has been corrupted by the adversary.

6.2 Basic privacy protection

Scrubbing sensitive fields. Before an alert is sent to the repository, the producer must remove all sensitive information not needed for collaborative analyses described in section 7, including all content in Captured_Data, Infected_File and Outcome fields. A more advanced version of our system may enable privacy-preserving analysis based on commonalities in the Captured_Data field, *e.g.*, presence of "bad words" associated with a particular virus. Possible techniques include encryption with keyword-specific trapdoors in the manner of [29, 5].

The Sensor_Id field may be either remapped to a unique persistent pseudonym (*e.g.*, a randomly generated string) that leaks no information about the organization that owns it, or replaced with just the make and model information. The Timestamp field is rounded up to the nearest minute. Although this disables fine-grained propagation analyses, it adds additional uncertainty against attackers staging probe-response attacks.

Hiding IP addresses. Suppose the attacker controls the repository. He may launch an attack and then attempt to use the alert generated by the victim's sensor to analyze the attack's propagation through the victim's internal network. Therefore, the producer must hide both Source_IP and Dest_IP addresses before releasing the alert to the repository.

Encrypting IP addresses under a key known only to the producer is unacceptable, as it hides too much information. With a semantically secure encryption scheme, encrypting the same IP address twice will produce different ciphertexts, disabling collaborative analysis. Hashing the address using a standard, universally computable hash function such as SHA-1 or MD5 enables dictionary attacks. If the attacker controls the repository, he can target a system on a particular subnet and pre-compute hash values of all possible IP addresses at which sensors may be located or to which he expects the attack to propagate. This is feasible since the address space in question is relatively small — either 256, or 65536 addresses (potentially even smaller if the attacker can make an educated guess). The attacker verifies his guesses by checking whether the received alert contains any of the pre-computed values.

Our solution strikes a balance between privacy and utility. The producer hashes all IP addresses that belong to his *own* network using a *keyed* hash function such as HMAC [3, 4] with his secret key. All IP addresses that belong to *external* networks are hashed using a *standard* hash function such as SHA-1 [23]. This guarantees privacy for IP addresses on the producer's own network since the attacker cannot verify his guesses without knowing the producer's key. In particular, probe-response fails to yield any useful information. Of course, if these addresses appear in alerts generated by other organizations, then no privacy can be guaranteed.

We pay a price in decreased functionality since alerts about events on the network of organization A that have been generated by A's sensors cannot be compared with the alerts about the same events generated by organization B's sensors. Recall, however, that we are interested in detecting large-scale events. If A is under heavy attack, chances are that it will be detected not only by A's and B's sensors, but also by sensors of C, D, and so on. Because A's network is external to B, C, and D, their alerts will have A's IP addresses hashed using the same standard hash function. This will produce the same value for every occurrence of the same IP address, enabling matching and counting of hash values corresponding to frequently occurring addresses. Intuitively, any subset of participants can match and compare their observations of events happening in *someone else's* network. The cost of increased privacy is decreased utility because hashing destroys topological information, as discussed in section 7.2. Naturally, an organization can always analyze alerts referring to its own network, since they are all hashed under the organization's own key.

An additional benefit of using keyed hashes for alerts about the organization's own events and plain hashes for other organizations' events is that the attacker cannot feasibly determine which of the two functions was used. Even if the attacker controls the repository and directly receives A's alerts, he cannot tell whether an alert refers to an event in A's or someone else's network. The attacker may still attempt to verify his guesses by precomputing hashes of expected IP addresses and checking alerts submitted by *other* organizations, but with hundreds of thousands of alerts per hour and thousands of possible addresses this task is exceedingly hard. Staging a targeted probe-response attack is also more difficult: the probe may never be detected by another organization's sensors, which means that the response is never computed using plain hash, and the attacker cannot stage a dictionary attack at all. Finally, note that keyed hashes do not require PKI or complicated key management since keys are never exchanged between sites.

Re-keying by the repository. To provide additional protection against a casual observer or an outside attacker when an alert is published, the repository may replace all (hashed) IP addresses with their keyed hashes, using the repository's own private key. This is done on top of hashing by the alert producer, and preserves the ability to compare and match IP addresses for equality, since all second-level hashes use the same key. This additional keyed hashing by the repository defeats all probe-response and dictionary attacks except when the attacker controls the repository itself and all of its keys, in which case we fall back on protection provided by the producer's keyed hashing.

Randomized hot list thresholds. For collaborative detection of high-volume events, it is sufficient for the repository to publish only the *hot list* of reported alerts that have something in common (*e.g.*, source IP address, port/protocol combination, event id) and whose number exceeds a certain threshold. As described in section 4, this may be vulnerable to a flooding attack, in which the attacker launches a probe, and then attempts to force the directory to publish the targeted system's response, if any, by flooding it with "matching" fake alerts based on his guesses of what the real alert looks like.

Our solution is to introduce a slight random variation in the threshold value. For example, if the threshold is 20, the repository chooses a random value T between 18 and 22, and, if T is exceeded, publishes only T alerts. If the attacker submits 20 fake alerts and a hot list of 20 alerts is published, the attacker doesn't know if the repository received 20 or 21 alerts, including a matching alert from the victim. There is a small risk that some alerts will be lost if their number is too small to trigger publication, but such alerts are not useful for detecting high-volume events.

Delayed alert publication. If the alert data is used only for research on historical trends (see section 7.1), delayed alert publication provides a feasible defense against probe-response attacks. The repository simply publishes the data several weeks or months later, without Timestamp fields. The attacker would not be able to use this data to correlate his probes with the victim's responses.

Examples of basic sanitization for different alert types can be found in tables 2 through 4.

6.3 Multiple repositories

We now describe a "heavy-duty" solution for the corrupt repository problem. Instead of using a single alert repository, envision multi-

Field ID	Raw firewall alert	Sanitized firewall alert
Source_IP	172.16.30.2	0x16e9368f
Source_Port	1147	1147
Dest_IP	173.19.33.1	0x78a65237
Dest_Port	135	135
Protocol	6	6
Timestamp	09032003:01:03:10	09032003:01:03:00
Sensor	PIX-4-10060231	PIX
Count	1	1
Event_ID	Deny	Deny
Outcome	none	none
Capture_Data	none	none
Infected_File	none	none

Table 2: Example firewall security alert sanitization.

Field ID	Raw IDS alert	Sanitized IDS alert
Source_IP	172.16.30.49	0xb09956c2
Source_Port	1299	1299
Dest_IP	176.20.22.43	0xd6e79b79
Dest_Port	80	80
Protocol	6	6
Timestamp	10132003:11:41:09	10132003:11:41:00
Sensor	EM-HTTP-90209321	EM-HTTP
Count	1	1
Event_ID	CGI_ATTACK	CGI_ATTACK
Outcome	NO_REPLY	none
Capture_Data	/scripts/.%255c%255c./winnt/system 32/cmd.exe?/c+dir	none
Infected_File	none	none

Table 3: Example IDS security alert sanitization.

Field ID	Raw AV Alert	Sanitized AV alert
Source_IP	none	none
Source_Port	none	none
Dest_IP	176.30.22.11	0xb4ddc807
Dest_Port	none	none
Protocol	none	none
Timestamp	11172003:09:39:00	11172003:09:39:00
Sensor	NORTON-AV-02209302	NORTON-AV
Count	1	1
Event_ID	W32.Sobig.F.Dam	W32.Sobig.F.Dam
Outcome	Left alone	none
Capture_Data	none	none
Infected_File	A0014566.pdf	none

Table 4: Example antivirus security alert sanitization.

ple repositories, operated by different owners and distributed throughout the Internet (*e.g.*, open-source code for setting up a repository may be made available to anyone who wishes to operate one). We do not require the repositories to synchronize their alert datasets, so the additional complexity is low. Information about available repositories is compiled into a periodically published list. An organization that wants to take advantage of the alert sharing infrastructure chooses one or more repositories in any way it sees fit — randomly, on the basis of previously established trust, or using a reputation mechanism such as [9, 12].

In this setting, it is insufficient for the attacker to gain control of just one repository to launch a probe-response attack because the victim may report his alert to a different repository. The costs for the attacker increase linearly with the number of repositories. The costs for alert producers do not increase at all, since the amount of processing per alert does not depend on the number of repositories.

While spreading alerts over several repositories decreases opportunities for collaborative analysis, real-time detection of high-volume events is still feasible. If multiple systems are under simultaneous attack, chances are their alerts will be reported to different repositories in sufficient numbers to pass the "hot list" threshold and trigger publication. By monitoring a sufficiently large subset of the repositories for simultaneous spikes of similar alerts, it will be possible to detect an attack in progress and adopt an appropriate defensive posture. Repositories may also engage in periodic or on-demand exchanges of significant perturbations in incoming alert patterns. This could further help build an aggregate detection capability, especially as the number of would-be repositories grows large.

Randomized alert routing. For better privacy, we propose to deploy an overlay protocol for randomized peer-to-peer routing of alerts in the spirit of Crowds [27] or Onion routing [33]. Each alert producer and repository sets up a simple alert router outside its firewall. The routers form a network. When a batch of alerts is ready for release, the producer chooses one of the other routers at random and sends the batch to it. After receiving the alerts, a router flips a biased coin and, with probability p (a parameter of the system), forwards the alert to the next randomly selected router, or, with probability $1-p$, deposits it into a randomly selected repository. The alert producer may also specify the desired repository as part of the alert batch.

Such a network is very simple to set up since, in contrast to full-blown anonymous communication systems such as Onion routing, there is no need to establish return paths or permanent channels. The routers don't need to maintain per-alert state or use any cryptography. All they need to do is randomly forward all received alerts and periodically update the table with the addresses of other routers in the network.

When an alert enters the network, all origin data is lost after the first hop. Even if the attacker controls some of the routers and repositories, he cannot be sure whether an alert has been generated by its apparent source or routed on behalf of another producer. This provides probabilistic anonymity for alert sources which is quantified below. The disadvantage is the communication overhead and increased latency for alerts before they arrive to the repository (note that there is no cryptographic overhead).

Anonymity estimates. To quantify the anonymity that alert contributors will enjoy if the repositories and producers form a randomized alert routing network, we compute the increase in attacker workload as a function of the average routing path length. If p is the probability of forwarding at each hop, then the average path length $m = 2 + \frac{p}{1-p}$. Reversing the equation, the forwarding probability p must be equal to $\frac{m-2}{m-1}$ to achieve the average path length of m.

Suppose the network contains n routers, of which c are controlled by the attacker. The probability that a random path contains a router controlled by the attacker is $\frac{c(n-np+cp+p)}{n^2-np(n-c)}$ [27]. For large n, this value is close to $\frac{c}{n}$, which means that almost $1 - \frac{c}{n}$ alerts will *not* be observed by the attacker and thus remain completely anonymous.

For each of the $\frac{c}{n}$ alerts that *are* observed by the attacker, the probability that its apparent source (the site from which an attacker-controlled router has received it) is the actual source can be calculated as $\frac{n-p(n-c-1)}{n}$ [27]. We interpret the inverse of this probability as the attacker *workload*. For example, if there is only a 25% chance that the observed alert was produced by its apparent source, the attacker needs to perform 4 times the testing to determine whether the apparent source is the true origin. As expected, higher values of forwarding probability p provide better anonymity at the cost of increased latency (modeled as increase in the average number of hops an alert has to travel before arriving to the repository). This relationship is plotted (assuming $n = 100$ routers) in figure 3.

7 Supported Analyses

Alert sanitization techniques described in section 6 protect sensitive information contained in raw alerts, but still allow a wide variety of large-scale, cross-organization analyses to be performed on the sanitized data.

7.1 Historical trend analyses

This class of analyses seeks to understand the statistical characteristics and trends in alert production that have been observed over various durations of time. For example, [31] offers a compendium of the trends observed in firewall and intrusion detection alert production from a sample set of over 400 organizations in 30 countries.

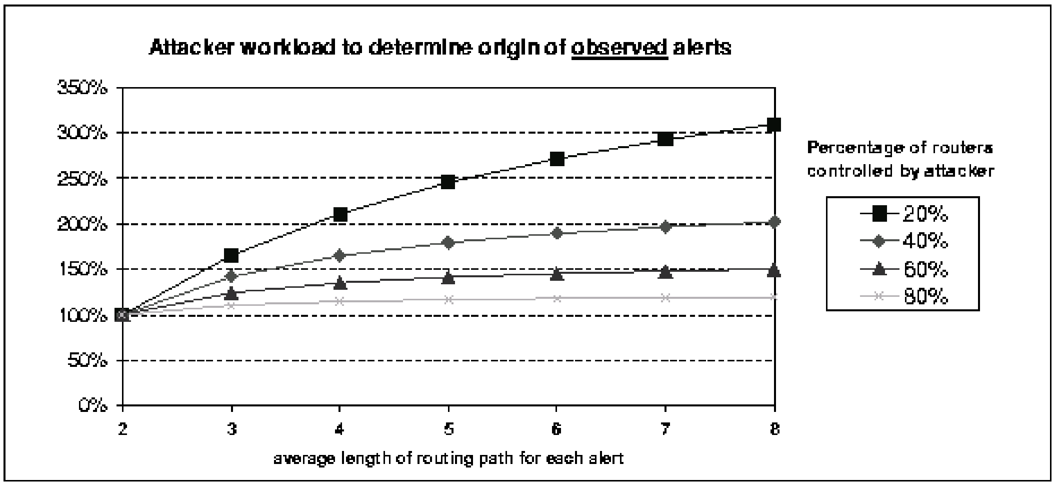

Figure 3: Estimated anonymity provided by randomized alert routing.

Source- and target-based. Given a large alert corpus, alert sources and targets may be categorized from various perspectives, such as event production patterns. Because of privacy-preserving data sanitization, geographical information and domain types cannot be inferred from the published alerts. One possible solution is to rely on self-classification and allow contributors to associate concise high-level profiles with each alert, including such attributes as country, business type, and so on (*e.g.*, "an academic institution in California"). This will enable some forms of trend/categorical analysis, but will also potentially make alert contributors more vulnerable to dictionary attacks.

We *do* enable identification of (anonymous) sources producing the greatest volume of alerts and alerts with the greatest aggregate severity. The activity of egregious sources is likely to be reported by multiple organizations, thus the corresponding address will be hashed using a universally computable hash function such as SHA-1. These sources can be blacklisted by distributing filters with the corresponding hash value. When installed, they would filter out all traffic for which the hash of the source IP address matches the provided value. There is a cost to this filtering, since it requires the firewall to hash the IP addresses of *all* incoming traffic to determine which ones need to be filtered out, although this may be acceptable when the network is under a heavy attack (this hashing is benign as opposed to dictionary attacks described in section 4.3). Repositories should beware of malicious blacklisting caused by the attacker submitting a large number of fake alerts implicating an innocent system.

Port/protocol- and event production-based. These analyses may offer help in understanding which kinds of reconnaissance are performed as a precursor to a larger scale exploit, or help characterize the extent to which an attack has spread.

7.2 Event-driven analyses

Real-time alert data published by alert repositories offers compelling value as a source of early warning signs that a new outbreak of malicious activity is emerging across the contributor pool. The focus of this analysis is to identify significant changes or sudden inflections in alert production that may be indicative of a currently occurring attack.

- Intensity analysis identifies extremely aggressive sources causing a large number of alerts from multiple contributors.

Although the sources remain anonymous, hash values of their IP addresses can be published and/or distributed to contributors to help them adjust their filtering policies, as described above.

- Sudden and widespread inflections in the volume and ratios of event_IDs and Dest_Ports in the incoming alert streams may indicate the emergence of a new intrusion threat that is affecting a growing subset of the contributor pool.

- Aggregation of the volume and severity of alerts observed in the incoming alert streams may provide a basis from which to capture an overall assessment or "Defcon level" of the threats that the contributor pool is currently facing.

A more challenging task is to identify propagation patterns in the occurrence of event_IDs and volumes, which is necessary to analyze spreading behavior of Internet-scale intrusion activity. Both hashing and keyed hashing destroy all topological information in IP addresses, making it infeasible to determine whether two sanitized alerts belong to the same region of address space. A possible solution may be offered by prefix-preserving anonymization [36], but we leave these techniques for future investigation.

8 Performance

As illustrated in figure 2, large volumes of alert data are being generated, and alert production among members of the contributor pool can vary greatly. Security services can produce inundations of security alerts when they are the target of a denial of service attack, and when there is a widespread outbreak of virulent worm or virus. During such periods of significant stress, alert production and processing can pose significant burden on sensors, repositories, and analysts, and thus limit utility of the alerts. This is a significant motivator for work on alert reduction methods [35, 10], and places constraints on the acceptable costs of alert sanitization.

As we show below, the cost of providing privacy to alert producers in our scheme is very low: there is a small impact on the performance of alert producers, and virtually no impact on the performance of supported analyses (of course, some analyses are disabled due to data sanitization). We argue that our scheme provides a sensible three-way tradeoff between utility of alert analysis, performance of the alert sharing infrastructure, and privacy of alert producers.

Performance of alert producers. To understand the CPU impact of alert sanitization, we benchmarked IP hashing on large alert corpuses under the scheme proposed in section 6.2, using SHA-1 on external IP addresses (primarily Source_IP), and HMAC on internal IP addresses (primarily Dest_IP).

The experiment was conducted on a FreeBSD 1.4Ghz Intel Pentium III workstation using Mark Shellor's free software implementation of SHA and HMAC. [1] We employed two large alert repositories. One repository, produced from our laboratory firewall, consisted of 4,224,122 records collected over a three hour period during an intense exposure to the Kuang 2 virus [16]. The second repository consisted of 19,146,346 records collected over a 24 hour period by DShield.

Table 5 presents the results of the IP address hiding scheme on the DShield and laboratory alert corpuses, reported in CPU seconds per million records. The baseline represents the amount of seconds, in CPU time, required to read the alerts from secondary storage per 1 million records. The hashed and cached-8 times indicate the amount of CPU seconds required to apply SHA and HMAC hashing to the Source_IP and Dest_IP fields per 1 million records. The delta column represents the difference between the baseline alert reporting performance and the sanitized alert reporting performance.

Cached-8 represents a moderately optimized implementation with a very small cache holding the last 8 encountered IP addresses. Because our sanitization scheme is deterministic, we can use the previously

[1] Source code is available at `http://search.cpan.org/src/MSHELOR/Digest-SHA-4.1.0/src/`

	baseline	hashed	delta	cached-8	delta
DShield.org	29.81	64.16	34.35	56.84	27.02
Laboratory	75.80	110.34	34.54	106.20	30.40

Table 5: CPU Impact of IP Hashing (seconds per 1 million alerts).

hashed IP addresses from the cache. Caching makes sense in two cases:

- The site is hit by a scan across its full IP address space by a few infected or malicious external hosts. In this case, a few Source_IP addresses will occur with regularity, resulting in a high cache hit ratio.
- The site is hit by distributed-denial-of-service-type traffic against a subset of its valid servers. In this case, a few Dest_IP addresses will occur with regularity, resulting in a high cache hit ratio.

For the IP addresses not in the trusted domain (to which SHA is applied), caching achieved savings of about 65%.

The results reveal that the performance impact is modest, less than the cost of I/O in our implementation. For a sensor producing 1 million alerts per hour, the additional hashing expense is roughly 30 seconds of CPU time per hour. This overhead should be considered in the context of the much larger task of alert caching and periodic batched transmission to a remote alert repository. Key management is relatively cheap in our case: there is no need for PKI and keys are never distributed outside the producer's site.

The expected cost of randomized routing to anonymize alert sources depends on the parameters of the routing network such as the forwarding probability and is roughly linear in the number of hops. There is no cryptographic processing and alert routers are stateless (see section 6.3).

Performance of analysis. To achieve the balance between privacy and utility, our sanitization methods have been designed to have minimal or no effect on the performance of primary analyses. In particular, sanitized IP addresses are mapped into the same size record as the original IP addresses, and cross-alert comparisons can be carried out at the repository without any network interaction. Comparing hashes for equality takes the same time as comparing IP addresses, so there is zero impact on performance.

When a troublesome source IP address is identified, this information may need to be propagated back to the producer (this is infeasible in the randomized-routing setting due to the high overhead of maintaining a return path for each alert). The producer may opt to reveal the actual IP address of the offender. In the case of a widespread attack, many sensors may complain about a single IP address, and any of the victims may choose to reveal the source of the threat, to enable defensive filters to be tuned appropriately. Measuring the costs of such selective revelation is beyond the scope of this paper.

9 Conclusions

We have described a broad set of privacy concerns that limit the ability of sites to share security alert information, and enumerated a number of data sanitization techniques that strike a balance between the privacy of alert producers and the functional needs of multi-site correlation services, without imposing heavy performance costs. Our techniques are practical even for large alert loads, and, most importantly, do not require that alert contributors trust alert repositories to protect their sensitive data. This enables creation of open community-access repositories that will offer a better perspective on Internet-wide trends, real-time detection of emerging threats and a source of data for malicious code research.

As a first prototype to demonstrate basic alert sanitization with live sensors, we are developing a Snort alert delivery plugin that implements SHA/HMAC and field sanitization discussed in section 6.2. We also plan to analyze defenses against probe-response attacks in which the attacker artificially stimulates an alert with a rare Event_Id and then uses this Event_Id as a marker to recognize the response in the general alert traffic.

Acknowledgements. We thank Keith Skinner for his support in the initial performance analysis, and Johannes Ullrich from the SANS Internet Storm Center for providing access to data set samples from DShield.org. We are grateful to the anonymous reviewers for useful comments.

References

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proc. ACM SIGMOD '03*, pages 86–97, 2003.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proc. VLDB '02*, pages 143–154, 2002.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Proc. CRYPTO '96*, volume 1109 of *LNCS*, pages 1–15. Springer-Verlag, 1996.

[4] M. Bellare, R. Canetti, and H. Krawczyk. HMAC: Keyed-hashing for message authentication. Internet RFC 2104, February 1997.

[5] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. EUROCRYPT '04*, volume 3027 of *LNCS*, pages 506–522. Springer-Verlag, 2004.

[6] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: long-term protection against break-ins. *Cryptobytes*, 3(1):1–8, 1997.

[7] K. Carr and D. Duffy. Taking the Internet by storm. *CSOonline.com*, April 2003.

[8] C. Clifton, M. Kantarcioglou, J. Vaidya, X. Lin, and M. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations*, 4(2):28–34, 2002.

[9] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach to choosing reliable resources in peer-to-peer networks. In *Proc. ACM CCS '02*, pages 207–216, 2002.

[10] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proc. RAID '01*, pages 85–103, 2001.

[11] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Proc. CRYPTO '89*, volume 435 of *LNCS*, pages 307–315. Springer-Verlag, 1989.

[12] R. Dingledine, N. Mathewson, and P. Syverson. Reputation in P2P anonymity systems. In *Proc. Workshop on Economics of Peer-to-Peer Systems*, 2003.

[13] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. In *Proc. ICALP '01*, volume 2076 of *LNCS*, pages 927–938. Springer-Verlag, 2001.

[14] P. Gemmell. An introduction to threshold cryptography. *Cryptobytes*, 2(3):7–12, 1997.

[15] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or how to cope with perpetual leakage. In *Proc. CRYPTO '95*, volume 963 of *LNCS*, pages 339–352. Springer-Verlag, 1995.

[16] Internet Security Systems. Xbackdoor-kuang2v (4074). *ISS X-Force Advisory*, April 2003.

[17] J. Jegon. Security firm: MyDoom worm fastest yet. `CNN.com`, January 2004.

[18] J. Leyden. Blaster rewrites Windows worm rules. *The Register*, August 2003. `http://www.securityfocus.com/news/6725`.

[19] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Proc. CRYPTO '00*, volume 1880 of *LNCS*, pages 36–54. Springer-Verlag, 2000.

[20] D. Moore, V. Paxson, S. Savage, S. Staniford, and N. Weaver. Inside the Slammer worm. *IEEE Security and Privacy*, 1(4), 2003.

[21] D. Moore, C. Shannon, and K. Claffy. Code-Red: a case study on the spread and victims of an Internet worm. In *Proc. ACM Internet Measurement Workshop '03*, pages 273–284, 2003.

[22] D. Moore, G. Voelker, and S. Savage. Inferring Internet denial-of-service activity. In *Proc. USENIX Security Symposium*, pages 9–22, 2001.

[23] NIST. Secure hash standard. FIPS PUB 180-1, April 1995.

[24] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proc. ACM SIGCOMM '03*, pages 339–351, 2003.

[25] M. Peuhkuri. A method to compress and anonymize packet traces. In *Proc. ACM Internet Measurement Workshop '01*, pages 257–261, 2001.

[26] M. Rabin. How to exchange secrets by oblivious transfer. Aiken Computation Laboratory Technical Memo TR-81, 1981.

[27] M. Reiter and A. Rubin. Crowds: anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[28] Snort. `http://www.snort.org`, 2004.

[29] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proc. IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[30] S. Staniford, V. Paxson, and N. Weaver. How to own the Internet in your spare time. In *Proc. USENIX Security Symposium*, pages 149–167, 2002.

[31] Symantec. Symantec Internet security threat report. Technical report, Symantec Managed Security Services, February 2003.

[32] Symantec. DeepSight threat management system home page. `http://tms.symantec.com`, 2004.

[33] P. Syverson, D. Goldschlag, and M. Reed. Anonymous connections and onion routing. In *Proc. IEEE Symposium on Security and Privacy*, pages 44–54, 1997.

[34] J. Ullrich. DShield home page. `http://www.dshield.org`, 2004.

[35] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proc. RAID '01*, pages 54–68, 2001.

[36] J. Xu, J. Fan, M. Ammar, and S. Moon. On the design and performance of prefix-preserving IP traffic trace anonymization. In *Proc. ACM Internet Measurement Workshop '01*, pages 263–266, 2001.

[37] A. Yao. Protocols for secure computation. In *Proc. IEEE FOCS '82*, pages 160–164, 1982.

[38] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the DOMINO overlay system. In *Proc. NDSS '04*, 2004.

[39] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: global characteristics and prevalence. In *Proc. ACM SIGMETRICS '03*, pages 138–147, 2003.

[40] C. Zou, W. Gong, and D. Towsley. Code Red worm propagation modeling and analysis. In *Proc. ACM CCS '02*, pages 138–147, 2002.