# Collaborative Distributed Intrusion Detection

Michael E. Locasto* Janak J. Parekh Sal Stolfo
Angelos D. Keromytis Tal Malkin Vishal Misra
Department of Computer Science
Columbia University in the City of New York
{*locasto,janak,sal,angelos,tal,misra*}*@cs.columbia.edu*

**Abstract**

The rapidly increasing array of Internet–scale threats is a pressing problem for every organization that utilizes the network. Organizations often have limited resources to detect and respond to these threats. The sharing of information related to probes and attacks is a facet of an emerging trend toward "collaborative security." Collaborative security mechanisms provide network administrators with a valuable tool in this increasingly hostile environment.

The perceived benefit of a collaborative approach to intrusion detection is threefold: greater clarity about attacker intent, precise models of adversarial behavior, and a better view of global network attack activity. While many organizations see value in adopting such a collaborative approach, several critical problems must be addressed before intrusion detection can be performed on an inter–organizational scale. These obstacles to collaborative intrusion detection often go beyond the merely technical; the relationships between cooperating organizations impose additional constraints on the amount and type of information to be shared.

We propose a completely decentralized system that can efficiently distribute alerts to each collaborating peer. The system is composed of two major components that embody the main contribution of our research. The first component, named *Worminator*, is a tool for extracting relevant information from alert streams and encoding it in Bloom filters. The second component, *Whirlpool*, is a software system for scheduling correlation relationships between peer nodes. The combination of these systems accomplishes alert distribution in a scalable manner and without violating the privacy of each administrative domain.

**Keywords:** Collaborative Security, Intrusion Detection, privacy-preserving alert correlation

## 1 Introduction

With the growing threat to critical infrastructure computer networks and systems, the ability to rapidly and correctly identify, rank, and react to probes and attacks is of acute importance. We posit that probes and attacks can be detected accurately when multiple organizations share alert information. One noteworthy advantage is the detection of very stealthy attackers: those who distribute their reconnaissance activities over multiple sites and a long time period.

Classical intrusion detection systems (IDS's), whether host–based or network–based, are typically constrained within one administrative domain. In such an environment, information about the global state of network attack patterns is necessarily unexamined. This global information

*Contact Author

could prove crucial to organizations in ranking and addressing threats that they do perceive and alerting organizations to threats they would not otherwise have recognized.

Most current IDS's do not adequately address the need for information exchange between systems under attack. In addition, the fail–open nature of intrusion detection systems make them a natural target for attacks and subversion. As the recent Witty worm demonstrates [19], security systems themselves can be prime targets for attack. Any distributed IDS must enforce mechanisms that support the reliability of its nodes as well as the integrity and confidentiality of the alerts exchanged between those nodes.

### 1.1 Challenges for Collaboration

Collaborative methods for distributed intrusion detection seem poised to revitalize the arsenals of system administrators by offering the fusion of alerts from many different domains and types of IDS's (*e.g.,* both host and network–based anomaly detection and signature matching engines). The exchange of alert data can effectively supplement the knowledge gained from local sensors.

However, several critical problems must be addressed before intrusion alert information can be safely distributed among cooperating sites. First, the sheer volume of alerts can cause evidence of stealthy scan/attack activity to go unnoticed. Second, the currently observable rates of alert generation preclude the use of any solution incorporating a centralized system to aggregate and correlate alert information. Administrators need a reasonable chance to respond to the threats that are perceived by the system; replicating all alert information and forwarding it to a common central node places considerable trust in that node and risks both a single point of failure and the increasing congestion of the network near that node. Third, exchanging alert data in a full mesh quadratically increases the complexity of communication and bandwidth requirements. Fourth, information exchange between administrative domains needs to be carefully managed, as proprietary or confidential network data may escape the boundary of each domain. Finally, any partitioning of the alert information into distinct sets defined by the different domains may cause information loss by disassociating data that would otherwise have been considered in the same context.

### 1.2 Our Approach

We propose a collaborative and distributed approach to intrusion detection and present a system that addresses many of the difficulties present in information sharing of this type. By exchanging information between administrative domains, we gain the tantalizing benefit of near–global intrusion knowledge at the cost of communicating with a small number of peers.

Previous work on distributed intrusion detection has focused mainly on the exchange of data within a single organization. Only recently has research been performed on systems that support the exchange of data between administrative domains. The most notable of these is the DOMINO [25] system. DOMINO is an overlay network that utilizes the Chord [20] protocol to distribute alert information based on a hash of the source IP address. However, DOMINO does not address privacy requirements. In addition, we propose a completely decentralized architecture more akin to a peer–to–peer application rather than the dual–layer hierarchy that DOMINO employs.

Our innovative claims are based on the combination of Worminator[1] and the *Whirlpool* distribution mechanism to provide privacy to a large number of collaborating sites. Our two main contributions over previous work are:

[1] `http://worminator.cs.columbia.edu/`

1. We enable sharing of information between administrative domains by protecting the privacy, confidentiality, and integrity of messages.

2. We propose a peer-to-peer architecture that scales much better than the centralized or distributed (mainly 2-layer hierarchy) solutions previously proposed.

In a series of experiments and simulations we demonstrate the practicality and efficacy of our approach, showing that we can detect distributed attacks within an average of six time slices or exchange periods without the overhead of replicating and distributing data to all peers.

**Paper Organization** The remainder of this paper is organized as follows. In Section 2, we outline our approach to the problem and examine the threat model. In Section 3 we discuss in detail the implementation of our system, and Section 4 presents performance measurements to support our claims. We give an overview of the related work in Section 5. We discuss our plans for future work in Section 6, and conclude the paper in Section 7.

# 2 Architecture

We adopt two primary mechanisms in order to cope with the difficulties of distributed correlation and the potential volume of data being correlated. First, the construction of Bloom filters by Worminator [21] is employed to protect the confidentiality of the data being exchanged between domains. Second, efficient information exchange is accomplished with a distributed correlation scheduling algorithm. The scheduling algorithm dynamically calculates subsets of correlation peers that should communicate to exchange Bloom filters. Since information is also compacted by the Bloom filter, correlation between peers becomes extremely cost-effective in terms of bandwidth and processing power.

## 2.1 Motivation

A key motivating factor for organizations to join a collaboration group in performing distributed intrusion detection is that participants can implement fast mitigation strategies against threats they otherwise would not have known about; for example, DOMINO discusses the advantage of small blacklists (around 40 entries) that retain their efficacy even when data is fairly stale. Other responses include the content filtering strategies proposed by Moore *et al.* [13], prosecution, or military action.

## 2.2 Environment

From our discussion above, a set of requirements that guides the design of our system has emerged:

1. The exchange of alert information must not leak potentiall sensitive data.

2. Large alert rates hide stealthy activity.

3. Centralized repositories are single points of failure and likely unable to correlate the burgeoning amount of alerts.

4. Exchanging alerts in a full mesh quadratically increases the complexity of the problem.

5. Any solution that partitions data among nodes risks information loss by disassociating evidence that should be considered in the same context.

We make several assumptions about the environment the system exists in and the alert information the system exchanges. Our assumptions and choices are intended to carefully balance the requirements of data privacy with the need to derive useful information and actionable intelligence from the alert exchange.

The environment and user base for a collaborative distributed intrusion detection system is an important consideration. We envision cohorts of 25 to 100 organizations exchanging information. Such cohorts can be organizations with similar interests, such as universities, financial institutions, military or government networks, energy companies, news organizations, *etc.* Each of these organizations already manages their own fairly large enclave. In such an environment, the potential amount of raw network events and low–level alerts is staggering. DShield[2] currently reports the contribution of around 10 million alerts per day. Our experimental sensors at Columbia report the processing of some 1.6 million network events per hour.

The sheer volume of alert streams is a critical consideration in the design and evaluation of any distributed intrusion detection system. The size of current (and foreseeable) alert streams demands low–cost processing and correlation. Alert streams can threaten to dominate network bandwidth if they are unnecessarily replicated.

Perhaps the most important decision we make is to employ the use of "watchlists," or lists of IP addresses suspected of subversive behavior. The task of the distributed detection system is not to analyze the network or host events of other domains, but rather to correlate summaries of alerts to identify attackers. Therefore, watchlists encapsulate the appropriate information to exchange. In addition, we believe that encodings of these watchlists in some binary format (as opposed to the IDMEF) represents a bandwidth savings in the environment described above.

We also use the Antura Recon system [6] as a major component of our input; Antura acts as our alert generator. The advantage of employing Antura is that Recon alerts provide more information than other sensors for stealthy long term reconnaissance activity [16], thus providing richer information about attack precursors. Other correlation systems do not keep such long term information; they usually fuse information about current network events.

## 2.3 Threat Model

We must consider a threat model that accounts for two major types of attackers. The first kind of attacker attempts to evade the distributed intrusion detection mechanism by performing very low rate scans and probes. The sophisticated attacker will often scan single organizations at slow rates in parallel in order to increase his productivity. Such an attacker, when faced with our system, can attempt to guess or derive the schedule at which organizations trade alerts, thereby avoiding the promotion of his scan activity.

The second type of attacker targets the intrusion detection system itself and engages in numerous behaviors intended to corrupt the system or violate the confidentiality or integrity of alert information. Such an attacker can compromise some subset of nodes in the system in order to discover information about the organizations he is targeting. This threat is called "proxy watermarking": an attacker compromises a node in the system and scans some target in another organization via a proxy. The attacker then listens for alert information at the compromised node to discover if his attack via the proxy was detected by any member of the system.

[2] `http://www.dshield.org/`

### 2.4 Detection and Correlation Mechanisms

Novel methods for detection abound in the literature. In this paper, we focus both on signature–based methods, such as those used by the Snort Intrusion Detection engine [17] and anomaly–based detection mechanisms, such as those discussed in [23].

Correlation mechanisms range from the very simple to intricately elegant. Qin and Lee [15] present a sophisticated statistical mechanism for assessing causality in alert data sets, allowing them to detect new relationships among attacks.

Our basic hypothesis is that the proper level of information can be represented by distributing and correlating *watchlists* of suspected attacker IP addresses. Additional information can be mixed into this, including source and destination ports, and codes for the particular type of attack.

### 2.5 Preserving Privacy

While IDS alerts themselves could be distributed, there are two substantial disadvantages to doing this: first, organizations may have privacy policies or concerns about sharing detailed IP data, some of which might uncover who they normally communicate with. Second, these alert files grow rapidly given substantial traffic. While parameters may be tweaked to reduce potential noise, a preferable solution would be to encode the relevant information in a compact yet useful manner.

We provide a compact format via the use of Bloom filters. A Bloom filter is a one-way data structure that supports two operations: insertion and verification, *e.g.,* while no data can be extracted after being inserted in the Bloom filter, it is probabilistically possible to see if specific data has been inserted if presented a second time to the Bloom filter. This is accomplished by creating a compact bit vector (typically between $2^{15}$ – $2^{20}$ entries). Entries are indexed by the hash of the original data, *i.e.,* a high-quality hash of original data (in this case, IPs and port information) is generated, broken up into parts, and these parts are used as indices into the bit vector. Each resolved index in the bit vector is set to 1. This process is typically repeated multiple times (for different parts of the hash and/or different hashes), thereby increasing resiliency to noise or data saturation. Verification is similar to insertion; instead of actually setting bits, the bit vector is examined to determine if the bits are already set. Therefore, a process is run on the computer running the IDS to parse its alert output and to generate Bloom filters corresponding to (for example) IP/port endpoint data.

Since Bloom filters are compact one-way data structures, we get three benefits:

- *Compactness:* A Bloom filter smaller than 10k bits in size is still able to accurately verify tens of thousands of entries.

- *Resiliency,* even when the Bloom filter is decreased in size: When the Bloom filter is saturated, it starts giving false positives (*i.e.,* multiple data entries resolve to the same locations in the bit vector), but never gives false negatives. The false positives can be ameliorated by tuning or by correlation against multiple alert lists.

- *Security:* By utilizing a one-way data structure, organizations can correlate watchlists without releasing actual IP data, satisfying privacy needs while being able to participate. If further security from outside observers is needed, the dissemination protocol can be encapsulated in a secure tunnel, like SSL, thereby only granting Bloom filter access to the set of participants in the alert list correlation.

### 2.6 Distributed Correlation

A distributed correlation function must overcome the performance problems of a centralized model while balancing the information loss inherent in partitioning alert data among different nodes. The most straightforward way to accomplish distribution (forwarding all data from every node to every other node) involves a quadratic increase in the amount of data exchanged (and the associated processing, however lightweight that may be).

More sophisticated approaches are based on two different theories. The first approach creates an explicit mapping from alert data (specifically, source and target IP addresses) to particular correlation nodes. The reasoning behind this approach is that source and target IP addresses are the two most important features (besides target port) of alert data. With data about various machines collected in one node, the majority of correlation can be accomplished at that node without communicating with other nodes (except to distribute results of global interest).

The shortcomings to this approach (an approach similar to that used in DOMINO [25]) is that nodes become special cases of the centralized model: they are single points of failure for information pertaining to the IP address range being hashed. In addition, participants in the system may be uncomfortable with storing their raw alert information at a single node. This approach invests too much trust in each node. While a self–healing approach like Chord [20] can ameliorate the loss of a node, previous information stored at that node is at best lost temporarily (for example, in the case of a denial of service,) or corrupted (in the case of the node being compromised). Some of these shortcomings can be mitigated by replicating the data to some number of other nodes; however, it is not clear what the appropriate balance is between fault–tolerance through replication and utilization of network bandwidth and storage space. If data is replicated to every other node, we see an unacceptable quadratic increase in the cost of the system.

The second theory realizes the limitations of the first approach and instead observes that a theoretical optimal schedule exists for communicating information. If an oracle existed in the network that answered with the appropriate subset of nodes that should talk given a particular alert, communication links could be set up between these nodes in constant time without wasting effort talking to nodes with irrelevant data.

In this model, we assume that there is a set of nodes $S$ of size $N$. We assume that there is some reliable mechanism for any subset of nodes to communicate with each other. There is some whole piece of knowledge $K$, that if known would provide evidence of a distributed scan. During a distributed scan, some subset of $S$ is scanned and now contains a piece of knowledge $K_i$.

Normally, this knowledge would be discarded as insignificant. However, the optimal schedule allows this set of nodes to perfectly guess which of its neighbors also contains a piece of $K$. Note that each node could also find this information out via the $O(N^2)$ full mesh method of asking each other node in the network. However, we assert that this method is too costly in terms of trust, network bandwidth, and disk storage.

The key idea in this optimal schedule is that the correct subsets of nodes are always communicating. In order to mimic this behavior, the (approximately) correct nodes must talk to each other at (approximately) the right time. One naive way of accomplishing such a schedule is to pick relationships at random. However, a more robust model than the random approach provides some control over the rates at which node relationships change. When the correlation schedule is seeded with some secret high–entropy information, the task of intrusion detection can be viewed from a game theoretic perspective. Game theory approaches to intrusion detection have been explored by Kodialman and Lakshman [10].

### 2.7 Whirlpool

There is a clear need for efficient alert correlation in large–scale distributed networks. To address this need, we introduce the notion of *correlation scheduling:* the controllable formation and dissolution of relationships between nodes and groups of nodes in a network. These relationships can be envisioned as a dynamic overlay.

The *Whirlpool* mechanism is a procedure for coordinating the exchange of information between the members of a correlation group. The coordination mechanism is controlled by a dynamic and parameterizable correlation schedule.

Our approach is predicated on the previously described theoretical model of the optimal correlation schedule, the shortcomings of a fully interconnected mesh of correlation nodes, and the limitations of the ideal centralized approach to large–scale correlation. The main difficulty is that nodes would most likely discard data that in truth belong to a distributed alert. We must develop a mechanism whereby a node can quickly conference with other peers and determine whether or not a local alert is noise or signifies part of a distributed alert.

In this section, we identify the optimal correlation scheduler and present the *Whirlpool* scheduler, which closely approximates the optimal schedule and performs better than both a random subset selection model and the total exchange full–mesh model.

The basic mechanism behind Whirlpool is a set of dynamic federations of nodes that join and leave these federations at various rates. The variance in rates is intended to allow federation groups to retain some stability while expediting the import of new information into the group. Furthermore, this mechanism can be augmented with a distributed learning algorithm that assists in promoting alerts discovered by the distributed correlation.

Informally, the Whirlpool model can be thought of as a set of nodes of size $N$ arranged in concentric circles of size $\sqrt(N)$, much like electrons in the classic model of an atom. For convenience, we can assume that the rings are ordered such that the inner rings are spinning faster than the outer rings (although it makes no difference for the amount of information gleaned). To offer a concrete example, the outermost ring can advance clockwise by 1 position every 10 minutes, and the ring inside that advances clockwise by 1 position every 9 minutes, and so on, until the innermost ring advances by 1 position every minute. Rates for each ring are configurable.

If we draw a radius that passes through 1 node at each ring level, we have identified a temporary "family" or federation of nodes that can exchange alert data. Part of this family changes very rapidly (the nodes that belong to rings close to the center of the system) and part of the family is relatively stable. This combination provides both stability of the correlation mechanism as well as bringing fresh information into each family on a regular basis, increasing the chances that a particular family is capable of identifying a common watchlist–encoded behavior.

To further enhance this model, we can imagine that rings are arranged in a circular buffer: after the outermost ring completes a revolution, each ring advances inward, and the innermost ring becomes the outermost ring.

## 3 Implementation

The Worminator platform supports compact watchlist correlation via the replication and use of Bloom filters [2]. A Bloom filter is a one-way data structure that supports two operations: insertion and verification, *e.g.,* while no data can be extracted after being inserted in the Bloom filter, it is probabilistically possible to see if specific data has been inserted if presented a second time to the Bloom filter. This is accomplished by creating a compact bit vector (typically between $2^{15} - 2^{20}$

entries). Entries are indexed by the hash of the original data, *i.e.,* a high-quality hash of original data (in this case, IPs and port information) is generated, broken up into parts, and these parts are used as indices into the bit vector. Each resolved index in the bit vector is set to 1. This process is typically repeated multiple times (for different parts of the hash and/or different hashes), thereby increasing resiliency to noise or data saturation. Verification is similar to insertion; instead of actually setting bits, the bit vector is queried to see if the bits are already set.

The Worminator project currently supports the Antura [6, 16] IDS, and is implemented using about 2,500 lines of Java code. The program runs in two modes: generation and correlation. For generation, the Worminator process is run at specified intervals on the computer running Antura to parse its alert output and to generate Bloom filters corresponding to IP/port endpoint data. In the first version of the software, this Bloom filter data is then disseminated using HTTP to a centralized node at Columbia. Once such Bloom filters are generated and disseminated, individual sites can correlate their local alerts against one or many Bloom filters, and determine which IPs/ports match against others' results, identifying distributed attacks.

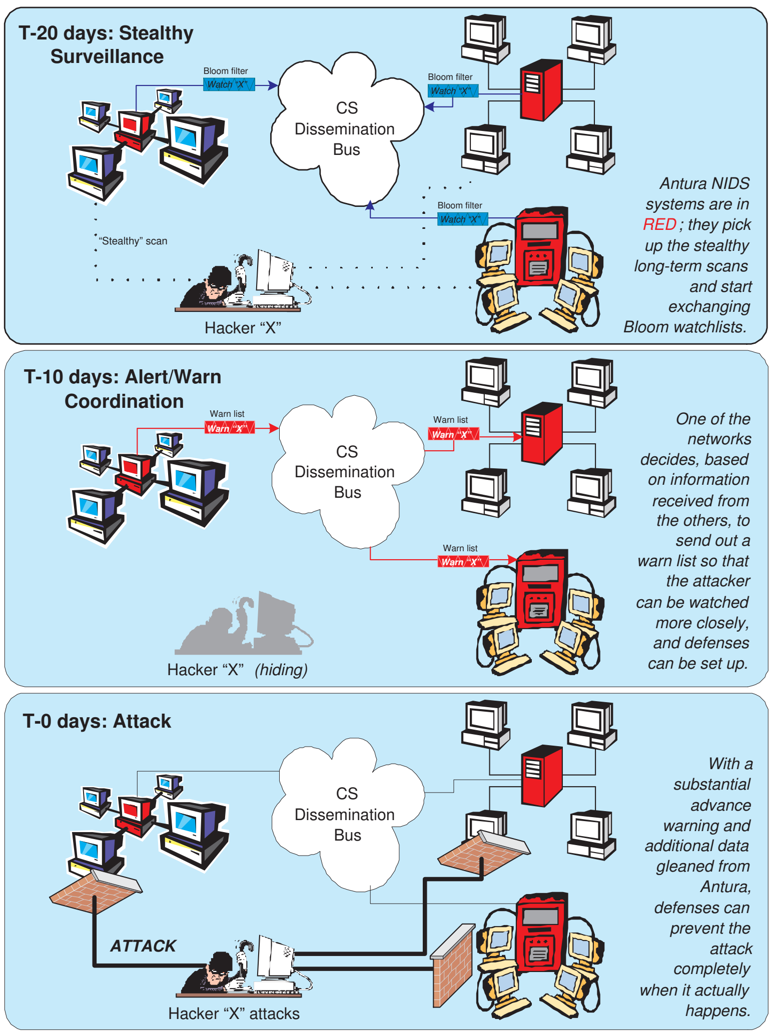


Figure 1: Collaborative Surveillance via Bloom filters.

The goal is to provide an easily replicated, secure watchlist that can be rapidly disseminated and correlated. Our first experiments focused on worm propagation rather than stealthy surveillance. Our results suggest that uncoordinated worm propagation are detectable by correlated watchlists from a set of collaborating sensor sites. In preliminary tests conducted at Columbia that correlated results from Worminator installations at Columbia and Georgia Tech, three common sources of stealthy surveillance were detected: one each in Beijing, the Phillippines, and a small western US community college. We are watching these sources to determine what behavior they exhibit over the coming weeks and months. We theorize that these sources are interested in Columbia and Georgia Tech to discover weakly protected or administrated university machines for nefarious purposes.

# 4 Performance and Whirlpool Evaluation

In order to validate our approach to the problems involved in exchanging alerts, we show that data reduction and aggregation before exchange is crucial. This can be observed through an analysis of publicly available alert rates. Bloom filters are a compact and powerful way to represent data that has been aggregated. In addition, we discuss the Whirlpool mechanism and compare its efficacy with the full mesh distribution mechanism.

## 4.1 Alert Rates

Intrusion detection systems face the very real threat of information loss from the sheer rate of available information. Schaelicke *et al.* [18] are decidedly pessimistic about the ability of relatively powerful commodity hardware and network links to absorb peak alert loads, noting that an IDS is effectively neutralized by the loss of alert data resulting from a database unable to keep up with incoming network data. This problem is compounded if multiple NIDS sensors report to the same database system [18]. They continue with the following analysis of the problem:

> For example, a dual-processor 800 Mhz Pentium-III system running MySQL is able to log 412 alerts per second into a MySQL database server configured for the ACID intrusion detection analysis console. A 100 Mbit Ethernet link at 50 percent utilization can carry approximately 23855 packets with an average size of 512 bytes per second. Given the maximum database throughput, at most 1.7 percent of all packets can trigger an alert before the database server is overloaded and alerts are lost. [18]

DShield reports about 10 million alert records added daily. For a year–long period between January 2002 and May 2003, DShield averaged the addition of 46 million alerts per week. For that same period, Figure 2 shows the increase in contributed data per month.

Experiments conducted on a workstation installation of Snort [17] indicate that currently observable alert rates (perhaps boosted by the production of false positives, depending on the IDS in use) can create large alert streams. In the course of a few hours, 375 attacking sources generated 3,139 alerts, totaling roughly 2MB (2,567,874 bytes) with an average size of 818 bytes. Alert size is greatly dominated by the capture of the payload content that triggered the alert. While most alerts are small (see Figure 3 and Figure 4), we consider the source IP address and destination port (and perhaps alert type, although it is not clear that such information is useful between domain employing different IDS's) to be the interesting pieces of information encoded in a Bloom filter.

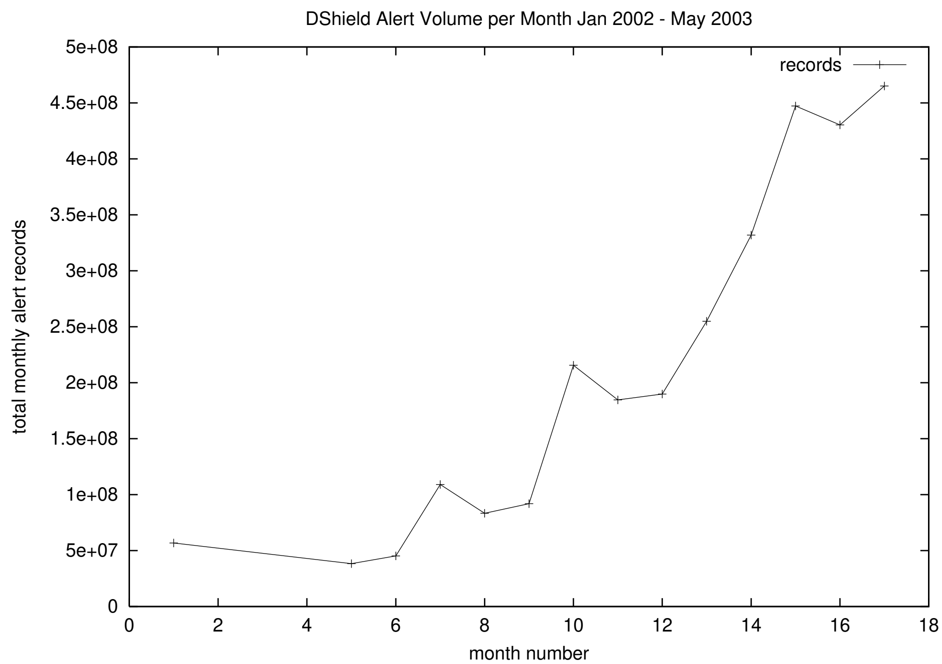


Figure 2: DShield monthly alert record contributions. The graph is not cumulative, but rather shows the rapid increase in contributed alert information per month as DShield grew in popularity.

### 4.2 Alert Aggregation and Reduction

Event reduction and aggregation is a critical part of our system. Since we construct watchlists from very little information (an IP address and a port), we are interested in ways of combining different alert information.

Reduction can be accomplished by filtering events either at the source sensor or prior to correlation processing. As a trivial example of the latter, consider a series of 100 basic IDS alerts with the same source and destination IP information and alert type information (*e.g., "Host X probed host Y on port Z"*). These alerts can be reduced to a single alert and a frequency. If a significant portion of alert streams are amenable to this type of reduction, we can either perform more expensive processing on the resulting stream, or produce actionable intelligence more rapidly.

Inexpensive reduction strategies (like logically grouping attacking IP source addresses in the same /24 subnet) can result in substantial compression, as is aggregation of multiple scan alerts (one per port) by a single source into one overall alert announcing a scan. For an example of the successful application of these reduction strategies, Figure 5 shows how the Antura Recon IDS [6] reduces evidence of certain scans and other alerts into a more manageable set of long–term alerts.

### 4.3 Whirlpool Implementation and Evaluation

The prototype implementation of the Whirlpool system is some 800 lines of Java code. Peers communicate via a multicast publish–subscribe mechanism, joining and leaving channels according to a parameterizable schedule. The implementation itself is fairly straightforward, as it consists of exchanging Worminator–generated alerts encoded as Bloom Filters.

To evaluate the effectiveness of Whirlpool, we compare it against a full mesh distribution scheme

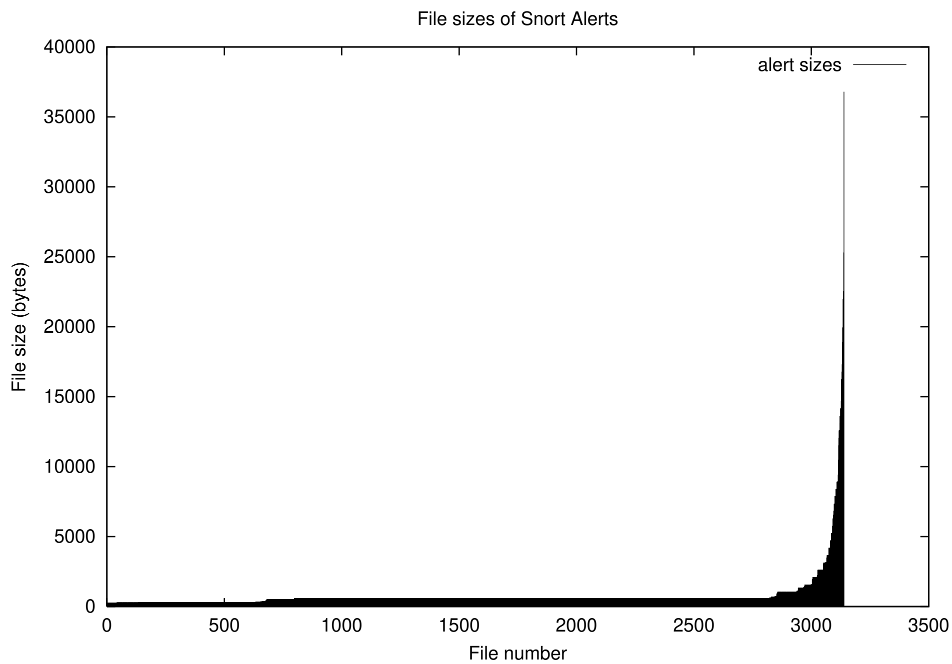


Figure 3: Range of alert sizes. A view of the distribution of various alert sizes generated by a workstation running Snort over the course of a few hours.

and a random selection distribution scheme. To that end, we introduce a *Bandwidth Effective Utilization Metric* (BEUM). The BEUM is defined as:

$$BEUM = \frac{1}{t * B}$$

where $t$ is the average number of time units it takes the distribution scheme to detect an attack and $B$ is the amount of bandwidth used by the distribution mechanism during that period. B is defined in terms of the total number of nodes, $N$, communicating via the distribution mechanism. Thus, for a full mesh scheme we have:

$$B = N * N$$

and the time to discover an attack is $t = 1$. The BEUM for a full mesh distribution strategy is therefore $\frac{1}{N^2}$. For a system of 100 nodes, the BEUM for a full mesh is $\frac{1}{10000}$

The BEUM for the Whirlpool mechanism is different, based on the calculation of the bandwidth consumed, $B$:

$$BEUM = \frac{1}{t * B}$$

$$B = N * \sqrt{N}$$

In general, if $t \leq \sqrt{N}$, the Whirlpool distribution mechanism wins. Specifically, for a system of 100 nodes, the BEUM for the Whirlpool scheme is $\frac{1}{1000t}$. If $t \leq 10$, the Whirlpool mechanism is a better choice than a full mesh. Our experiments show that Whirlpool is indeed more effective.

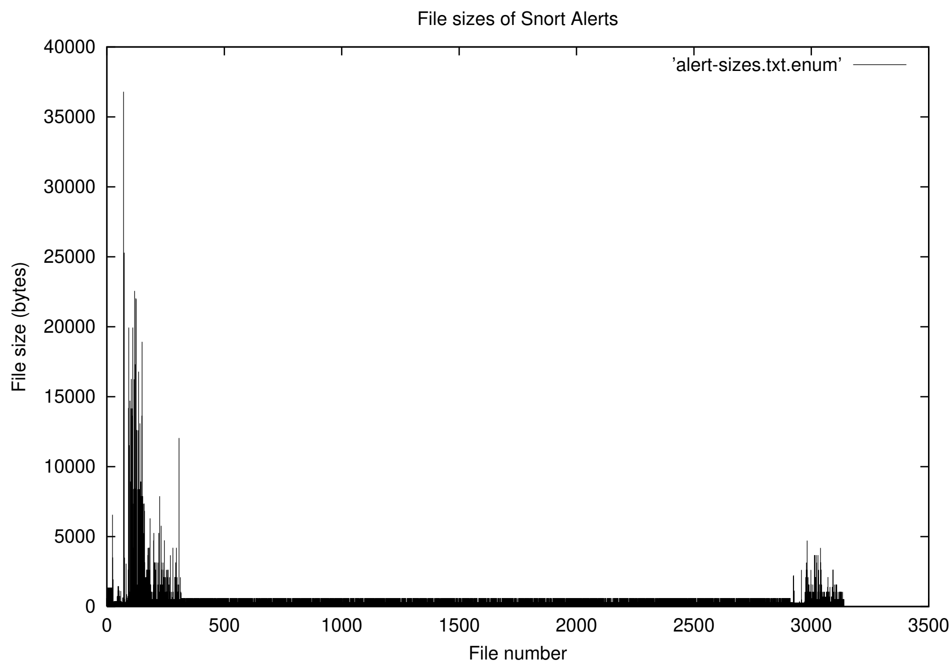


Figure 4: Range of alert sizes. Another view of the same workstation data unsorted by alert size. The data suggests that alert rates are unpredictable. Data reduction techniques should be employed to smooth out such rates.

We compare Whirlpool with a randomized distribution strategy by employing the BEUM. While detection with a random schedule performs badly when the number of nodes involved in the system grows (see Figure 6), our simulations for a system of 100 nodes (performed over 1000 trials) indicate that on average, it takes 5 to 6 time units before an attack is detected using a repeated random schedule. This time unit requirement satisfies the requirement for $t \leq 10$ we derived for the BEUM. Figure 7 shows that even though some pathological outliers exist, the vast majority of attacks are detected in a relatively short time.

The only remaining problems a randomized schedule has are unpredictable bandwidth requirements and potentially lengthy times to converge on a subset that detects a given attack. The latter difficulty is displayed in our simulation results of Figure 7. On the other hand, Whirlpool behaves quite well, displaying no outliers for a simulation of 100 nodes (see Figure 8) and having an average time slice of 6, well within the limit of $t \leq 10$ for 100 nodes. A consequence of the bounded detection with Whirlpool is that the necessary state for detection is also bounded: while with random execution we are forced to keep state for up to 90 time slices, with Whirlpool we only need to keep 9 past Bloom Filters. This significantly reduces the amount of state each node needs to maintain, and simplifies the correlation operation commensurately.

# 5 Related Work

Much of the traditional work in the area of distributed IDS alert correlation has focused on distributed collection and centralized correlation and often incorporates a significant offline component. Furthermore, even work that addresses decentralized correlation is limited in scope to a single ad-

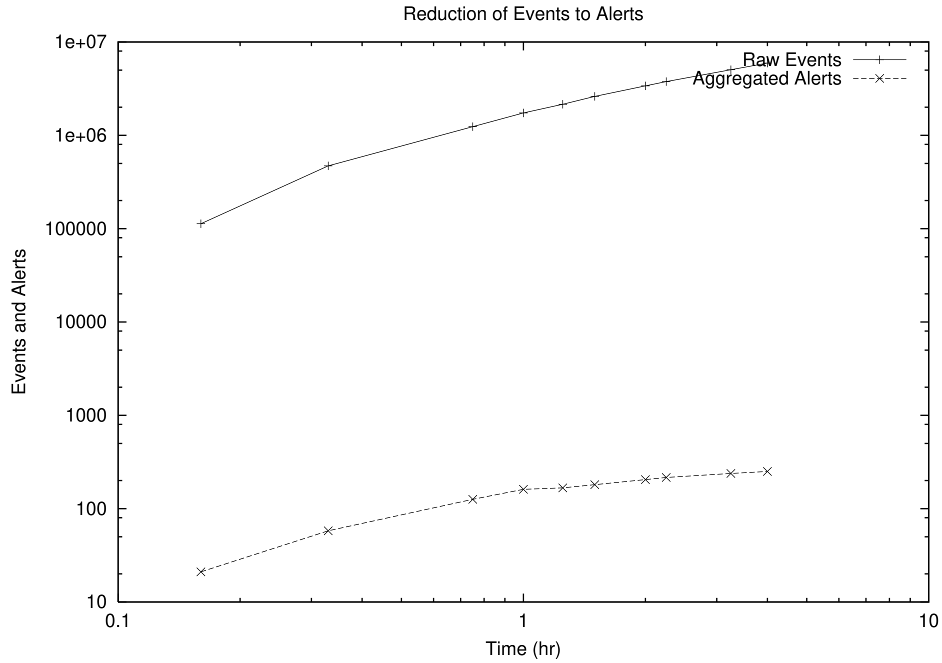


Figure 5: Reduction of Events to Alerts using Antura. The reduction of alerts over a four hour period is impressive. This approach shows the benefit of simple aggregation and reduction at the sensor source.

ministrative domain (although the basic mechanisms could conceivably extend across networks). The DShield project [22] is an example of a centralized repository that receives intrusion alerts from many distributed sources.

**Distributed Intrusion Detection** CARDS is a prototype distributed intrusion detection system realized by the *netforensics* product. CARDS uses "attack trees", or pre-defined sequences of attack steps as distributed signatures. CARDS decomposes global representations of distributed attacks into smaller units (called *detection tasks*) that correspond to the distributed events indicating the attacks, and then executes and coordinates the detection tasks in the places where the corresponding events are observed.

Attack trees are essentially a form of signature matching; attack trees can be modeled by Finite State Machines that specify the steps and relative order of an attack pattern. While fast algorithms for signature and string matching exist, the best known are of complexity $O(n \log^3 n)$ – quite impractical for dealing with millions of alerts per hour.

One notable follow–up work is constructing new attack sequences [14] to keep the signature database up to date and overcome the major shortcoming of signature–based schemes.

**Decentralized Alert Correlation** Cuppens and Miege [5] discuss methods for cooperatively correlating alerts from different types of intrusion detection systems in order to synthesize more meaningful alerts. Shortcomings of their system include the need to encode attacks in their LAMBDA language [4]. In addition, it appears that their system is constructed with an eye towards correlating information among IDS systems in the same administrative domain, as no

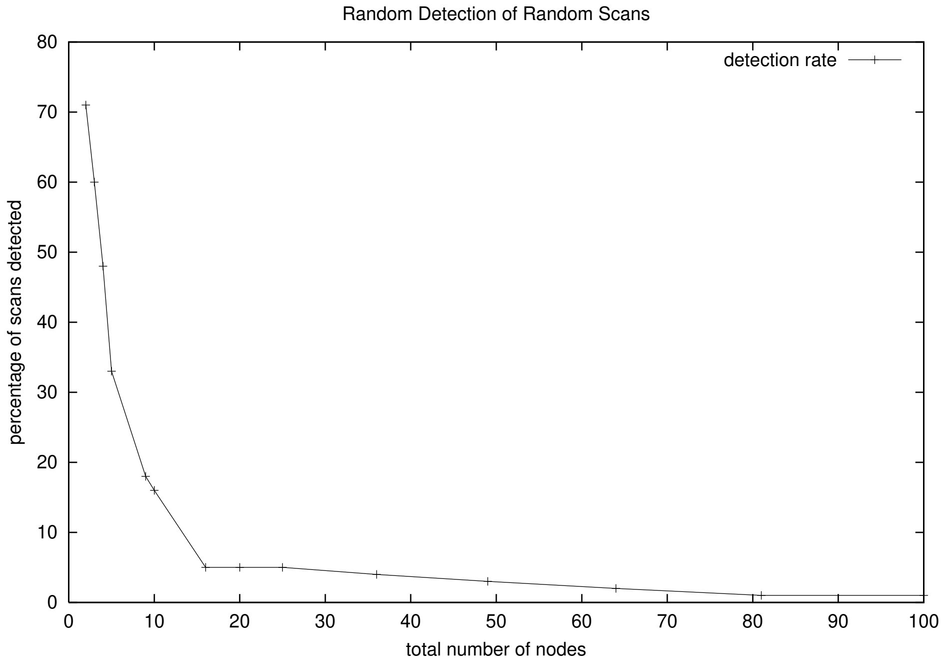


Figure 6: Detection performed with a random schedule.

privacy–preserving measures are discussed.

Krugel *et al.* [11] propose a peer-to-peer system that recognizes attacks in a distributed manner; the system is interesting because their results indicate two things. First, only a relatively small number of messages (seldom more than two in their experiments) need to be exchanged to determine an attack is in progress. Second, their work confirms that a peer–to–peer approach to decentralized intrusion detection is feasible and appropriate. They present a language for expressing attacks as a sequence of steps relating hosts. This "Attack Specification Language" has the advantage that only local information is necessary to decide which messages should be forwarded and is specifically limited to avoid exponential growth of information sharing.

However, their approach has two significant drawbacks. First, it requires that attack signatures be known *a priori* and be expressed in their attack meta–language. The language explicitly expresses the connections between nodes as the connections in the steps of an attack. Second, they trade alert information in the IDMEF format, which is not as compact as a binary representation and potentially wasteful of bandwidth. In addition, the scope of their research is the exchange of alerts within a single administrative domain and so they do not address privacy requirements.

**Large–Scale Collaborative Distributed Intrusion Detection** The research closest to ours in aim and scope is the recent work by Lincoln, Porras, and Shmatikov [12]. The authors present a practical scheme for Internet–scale collaborative analysis of information security threats and claim that it provides strong privacy guarantees to contributers of alerts. The authors sanitize and replicate alert data to enable alert correlation between cooperating peers. While the authors are mostly concerned with the scrubbing of alerts, we examine the need for the construction of an efficient distribution mechanism as well as alert privacy.

Huang and Wicks [9] and DOMINO [25] propose systems similar to ours in organization.

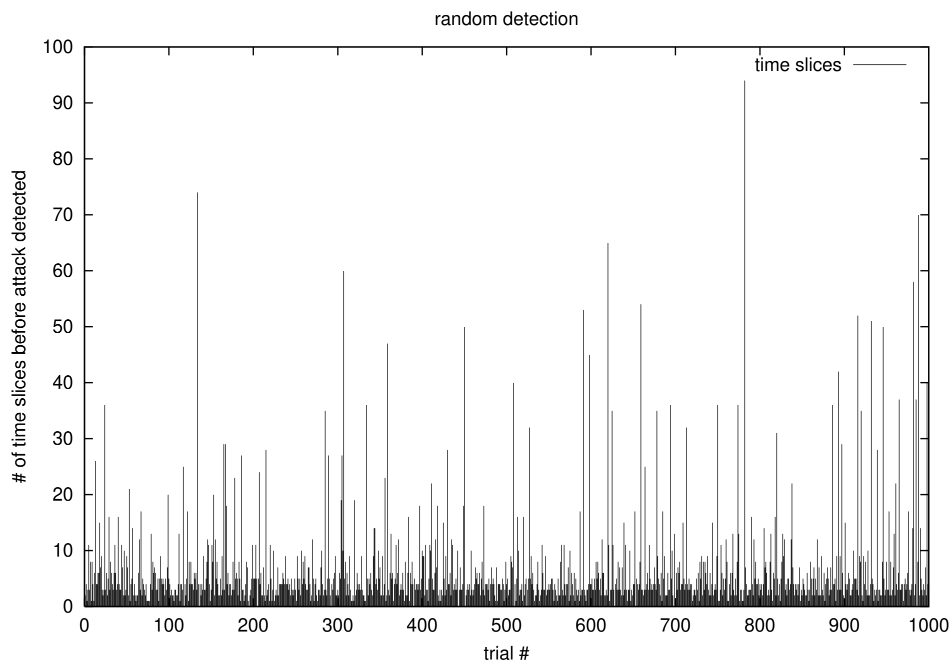


Figure 7: Number of time slices until random distribution detects an attack. The average number for this particular data plot is 6 time slices.

DOMINO, however, is directed at providing a large–scale monitoring facility to provide an early warning system for outbreaks of worms or other malicious traffic (DDos, etc.) and relies on the Chord protocol [20]. The work by Huang [9] presents a thoughtful exploration of the relevant features of attacks, concluding that reconstructing attack strategies is the appropriate method for balancing the cost of distributed correlation with the utility of global knowledge.

A recent study by Moore *et al.* [13] using the network telescope at UCSD defines the resources needed to counter worm propagation. They conclude that a response needs to be mounted in 2-3 minutes and that participation of nearly all major ASes is required to be effective. The authors are pessimistic about the preparedness of the general Internet for preventing and containing worm outbreaks. Their results suggest that both technological and administrative issues must be addressed before any effective defense can be mounted against such Internet–wide threats.

While these numbers are specifically derived in the context of quarantining Internet worms, their results show that current and foreseeable threats demand a cooperative and collaborative approach to achieving security. Our proposed system effectively addresses many of the primary technological and administrative concerns with sharing intrusion detection data.

**Secure Multi–Party Computation** Secure multi-party computation can be used to achieve stronger reliability and robustness, as well as to maintain privacy of the data to the maximal extent possible. Secure multi-party computation, initiated by [24, 8, 1, 3]), is one of the most fundamental achievements of cryptography in the last two decades. Du and Atallah [7] briefly discuss *privacy-preserving intrusion detection* in the course of enumerating practical applications of Secure Multi-Party Computation. The authors of [12] specifically avoid the use of SMC approachs due to their cost.

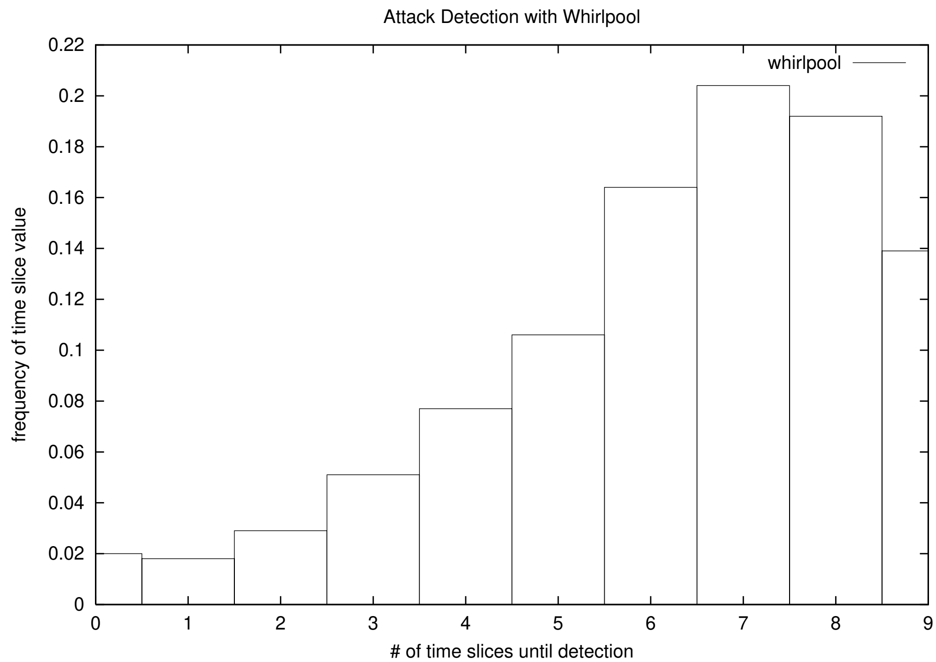


Figure 8: The number of time slices to detect an attack with Whirlpool over 1000 trials. For this graph, the average number of time slices was 6. This is not surprising, since the Whirlpool is a special case of a random schedule. Note, however, that Whirlpool does not have the outliers that a random schedule has because it converges.

# 6 Future Work

Our future work will especially focus on *collaborative security*, of which collaborative intrusion detection is a vital component.

We shortly expect to coordinate and participate in a large scale study of the techniques presented in this paper with other academic institutions and industry partners. We also intend to work with groups based in Europe to obtain a geo–global view of attack patterns.

We will continue to study methods for low cost alert distribution and propagation. In addition, we envision developing metrics for ranking the threat level for particular "verticals", or groups of organizations. We will also explore the development of a framework for typing and versioning of our Bloom filter watchlists and the necessary mechanisms for managing these typed alerts.

# 7 Conclusions

Distributed intrusion detection is being increasingly viewed as a powerful tool against sophisticated adversaries that do not generate high–visibility events that can be easily caught by traditional IDS's. In this paper, we seek to overcome two fundamental limitations of such systems: the disclosure of potentially sensitive information to collaborating entities, and the scaling of such systems to real workloads (*i.e.,* handling millions of alerts per hour), both from a systems engineering (*i.e.,* the impact of such workloads on system performance) and a granularity of detection perspective.

We have presented two components for our system, the *Worminator* alert generator, and the

*Whirlpool* alert-trade scheduler. Worminator creates compact watchlists of IP addresses encoded in Bloom Filters. The use of Bloom Filters minimizes the amount of sensitive information exposed to third parties (that are participating in the distributed IDS), without hampering the detection effort itself. Furthermore, this compact representation lessens the impact of the distributed computation on the network and simplifies the correlation process itself. Whirlpool creates federations of nodes that exchange alert information (encoded as Bloom Filters) at any particular time, varying the rate at which nodes join and leave these groups.

Our experiments and simulations show that our system can detect distributed attacks within six exchange periods without using an unacceptable amount of bandwidth to distribute alert information to peers. In our plans for future work, we intend to deploy Worminator using the Whirlpool distribution mechanism to a number of different organizations and entities to correlate watchlists. Having such a mechanism deployed can allow us to perform mitigation and preemptive threat response without having been directly attacked.

## References


[1] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proc. of the 20th Annu. ACM Symp. on the Theory of Computing*, pages 1–10, 1988.

[2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[3] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. of the 20th Annu. ACM Symp. on the Theory of Computing*, pages 11–19, 1988.

[4] F. Cuppens and R. Ortalo. Lambda: A language to model a database for detection of attacks. In *Proceedings of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID 2000)*, Toulouse, France, October 2000.

[5] Frederic Cuppens and Alexandre Miege. Alert Correlation in a Cooperative Intrusion Detection Framework. In *IEEE Security and Privacy*, 2002.

[6] System Detection. System Detection Releases Antura Vision, September 2003.

[7] Wenliang Du and Mikhail J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *Proceedings New Security Paradigms Workshop (NSPW)*, Cloudcroft, NM, 2001.

[8] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. of the 19th Annu. ACM Symp. on the Theory of Computing*, pages 218–229, 1987.

[9] Ming-Yuh Huang and Thomas M. Wicks. A large-scale distributed intrusion detection framework based on attack strategy analysis. In *Proceedings Recent Advances in Intrusion Detection (RAID)*, 1998.

[10] Murari Kodialam and T. V. Lakshman. Detecting network intrusions via sampling: A game theoretic approach. In *Proceedings of 22nd Annual Joint Conference of IEEE Computer and Communication societies (INFOCOM 2003)*, April 1-3 2003. San Francisco, CA.

[11] C. Krugel, T. Toth, and C. Kerer. Decentralized event correlation for intrusion detection. In *International Conference on Information Security and Cryptology (ICISC)*, December 2001.

[12] Patrick Lincoln, Phillip Porras, and Vitaly Shmatikov. Privacy–Preserving Sharing and Correlation of Security Alerts. In *USENIX Security. To appear.*, 2004.

[13] David Moore, Colleen Shannon, Geoffrey Voelker, and Stefan Savage. Internet Quarantine: Requirements for Containing Self–Propagating Code. In *INFOCOM 2003*, 2003.

[14] Peng Ning, Yun Cui, and Douglas Reeves. Constructing Attack Scenarios Through Correlation of Intrusion Alerts.

[15] X. Qin and W. Lee. Statistical causality analysis of infosec alert data. In *6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, September 2003.

[16] Seth Robertson, Eric Siegel, Matt Miller, and Salvatore Stolfo. Surveillance Detection in High Bandwidth Environments. In *2003 DARPA DISCEX III Conference*, April 2003.

[17] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of USENIX LISA '99*, November 1999. (software available from *http://www.snort.org/*).

[18] Lambert Schaelicke, Matthew R. Geiger, and Curt J. Freeland. Improving the Database Logging Performance of the Snort Network Intrusion Detection Sensor. Technical Report Technical Report 03-10, 2002.

[19] Colleen Shannon and David Moore. The Spread of the Witty Worm. Technical report, March 2004.

[20] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Application. In *Proceedings of ACM SIGCOMM*, August 2001.

[21] S. Stolfo. Worm and Attack Early Warning: Piercing Stealthy Reconnaissance. *IEEE Privacy and Security (to appear)*, May/June 2004.

[22] J. Ullrich. Dshield home page, 2004.

[23] K. Wang and S. Stolfo. Anomalous Payload-based Network Intrusion Detection. Technical report, 2004.

[24] A. C. Yao. Theory and application of trapdoor functions. In *Proc. of the 23th Annu. IEEE Symp. on Foundations of Computer Science*, pages 80–91, 1982.

[25] V. Yegneswaran, P. Barford, and S. Jha. Global Intrusion Detection in the DOMINO Overlay System. In *ISOC Symposium on Network and Distributed Systems Security*, February 2004.