Privacy-Preserving Distributed Event Correlation

Thesis proposal

Janak J. Parekh
Department of Computer Science
Columbia University
janak@cs.columbia.edu

Advisor: Prof. Gail E. Kaiser

November 7, 2005

Abstract

Event correlation is a widely-used data processing methodology for a broad variety of applications, and is especially useful in the context of distributed monitoring for software faults and vulnerabilities. However, most existing solutions have typically been focused on "intra-organizational" correlation; organizations typically employ *privacy policies* that prohibit the exchange of information outside of the organization. At the same time, the promise of "inter-organizational" correlation is significant given the broad availability of Internet-scale communications, and its potential role in both software maintenance *and* software vulnerability exploits.

In this proposal, I present a framework for reconciling these opposing forces in event correlation via the use of *privacy preservation* integrated into the event processing framework. By integrating flexible privacy policies, we enable the correlation of organizations' data without actually releasing sensitive information. The framework supports both *source anonymity* and *data privacy*, yet allows for the time-based correlation of a broad variety of data. The framework is designed as a lightweight collection of components to enable integration with existing COTS platforms and distributed systems. I also present two different implementations of this framework: XUES (**X**ML **U**niversal **E**vent **S**ervice), an event processor used as part of a software monitoring platform called KX (**K**inesthetics e**X**treme), and Worminator, a collaborative Intrusion Detection System.

KX comprised a series of components, connected together with a publish-subscribe content-based routing event subsystem, for the autonomic software monitoring of complex distributed systems. Sensors were installed in legacy systems. XUES' two modules then performed event processing on sensor data: information was collected and processed by the *Event Packager*, and correlated using the *Event Distiller*. While XUES itself was not privacy-preserving, it laid the groundwork for this thesis by supporting event typing, the use of publish-subscribe and extensibility support via pluggable event transformation modules.

Worminator, the second implementation, extends the XUES platform to fully support privacy-preserving event types and algorithms in the context of a Collaborative Intrusion Detection System (CIDS), whereby sensor alerts can be exchanged and *corroborated*—a reduced form of correlation that enables collaborative verification—without revealing sensitive information about a contributor's network, services, or even external sources as required. Worminator also fully anonymizes source information, allowing contributors to decide their preferred level of information disclosure. Worminator is implemented as a monitoring framework on top of a COTS IDS sensor, and demonstrably enables the detection of not only worms but also "broad and stealthy" scans; traditional single-network sensors either bury such scans in large volumes or miss them entirely. Worminator has been successfully deployed at 5 collaborating sites and work is under way to scale it up further.

The contributions of this thesis include the development of a cross-application-domain event correlation framework with native privacy-preserving types, the use and validation of privacy-preserving corroboration, and the establishment of a practical deployed collaborative security system. I also outline the next steps in the thesis research plan, including the development of evaluation metrics to quantify Worminator's effectiveness at long-term scan detection, the overhead of privacy preservation and the effectiveness of our approach against adversaries, be they "honest-but-curious" or actively malicious. This thesis has broad future work implications, including privacy-preserving signature detection and distribution, distributed stealthy attacker profiling, and "application community"-based software vulnerability detection.

Contents

1 Introduction

Event correlation is becoming increasingly useful in the context of distributed monitoring for software faults and vulnerabilities, as software continues to grow in size, configurations become more complex, and event/alert logs become increasingly difficult to understand. This proposal primarily concerns *distributed* event correlation, where application-specific data may be exchanged by any number of Internet peers.

The semantics (i.e., *types*) of this event data can differ greatly based on the application: it may contain information about method calls in an instrumented software program, database transactions being executed, or intrusion-detection alerts triggered by a malicious source. Such events may contain significant amounts of data as it is transmitted and processed, *or* may contain metadata about system behavior. As a result, event streams are generally confined to an individual organization, much like other strategic organizational data—and correlation systems remain within the organization's network as well. However, application semantics are no longer local area network-based; instead, the pervasiveness of Internet communication dictates that applications be robust to a broader variety of errors and vulnerabilities. Additionally, Internet-scale correlation gives any individual node far richer correlation data, as a wider range of nodes and their diversity enable a broader variety of correlation and debugging scenarios ([45], etc.).

To date, however, "inter-organizational" correlation (for brevity's sake, this proposal refers to this hereforth as *collaboration*) has been limited at best to organizations that forge business and legal relationships with each other—to avoid releasing sensitive information, such as trade secrets, or whose distribution may be limited by compliance requirements [52]. Most event correlation techniques to date focus on the expressiveness of correlation capability, as the fundamental working assumption is that almost any organizational data can be encapsulated into an event. These approaches do not work with anonymized, limited data exchanges between nodes (as discussed in section 4.1). Instead, a solution has to accommodate the notion of an organization's *privacy policy*, such as the disclaimer provided on websites disclosing appropriate practices with collected customer data. My practical experience in dealing with a broad variety of organizations, including academic, governmental, and business entities, demonstrates that such entities are either ignorant of the possibilities of collaboration or are actively against it due to the information disclosure problem.

Therefore, any solution that forms Internet-scale collaborations must be able to comply with privacy and information disclosure policies, yet still meaningfully correlate event data. This proposal directly addresses this tradeoff, building a type-driven framework adaptable to organizational privacy policies yet effectively correlating data between sites. This proposal is organized as follows: first, section 2 formalizes the definition of *privacy* and the requirements associated with it to effectively *corroborate* data—a restricted form of general correlation. The hypotheses and model are then presented in section 3. Section 4 discusses past and current related work as it relates to this problem. Two implementations of this model, XUES (**X**ML **U**niversal **E**vent **S**ervice), an event processor used as part of an autonomic software monitoring platform called KX (**K**inesthetics e**X**treme), and Worminator, a collaborative Intrusion Detection System, are introduced in section 5. Each discusses the "sub-problem" with respect to their appropriate application domain, the solutions adopted, and how they fit within the proposed model. The remaining steps of this thesis research and the schedule for completion are discussed in section 6. I discuss the expected contributions of the proposed work in section 7, and outline future work and conclude in section 8.

2 Problem, Definitions and Requirements

2.1 Definitions

I begin by formalizing some of the terms used in this proposal.

- **Events** are discrete, structured data objects generated at a specific point in time [55]. In other words, the semantics of events are left undefined, as they generally differ significantly based on the problem domain. The data that is stored as part of an application method call, for example, may differ significantly from that of an intrusion detection alert. A distinction *is* made, however, between events, which have an implicit *ordering* or explicit *timestamp*, as opposed to generally-structured data, which may or may not be ordered in a meaningful fashion.

- **Distributed** refers to Internet-scale communication and information sharing. The work in this thesis may also apply to large private networks, but the focus is on the most general Internet-based distribution mechanisms. *Peers* or *sites* may form collaboration groups via a variety of different communication protocols and topologies.

- **(Event) Correlation** has no single well-established definition, apart from the statistical definition of correlation that does not generally apply to Computer Science. For the purposes of this proposal, I adapt [35]'s definition: "Event correlation is the process of monitoring", in space and time, "what is happening on...systems in order to identify patterns of events that might signify attacks, intrusions, misuse or failure."

- **Privacy** also takes many different forms [20]. Some of the more relevant ones to event correlation include:

  1. **Source anonymity** refers, in particular, to the producer of an event. A source that is anonymous cannot be traced by recipients of the event: there is no explicit identifier linking the event to a known producer, and the data in the event cannot reliably be linked to the producer.

  2. **Data privacy** is related to, but not equivalent to, source anonymity: it specifically refers to the semantics of the data in the event and whether they contain information that may be deemed sensitive by the producer of the event.

  3. **Physical privacy** refers to the access of sensitive information or resources via direct access to the repositories or interference with servers of data. This includes intruders, malicious insiders, and resource starvation (e.g., denial-of-service) mechanisms.

  4. **Time privacy** corresponds to the fact that this thesis considers an event as being time-stamped. The distribution of event arrival times could yield some aggregate information; more interestingly, the correlation of curious or insidious activities with event arrival times could potentially violate the source anonymity stated above.

  This thesis focuses on the first two forms of privacy. It is possible to maintain data privacy without maintaining source anonymity (e.g., an event came from source X but it is free of what X deems sensitive), as well as vice-versa (e.g., it is unknown exactly who the event came from, but it contains classified information privy to only a small number of organizations). Of course, both can exist in tandem. With both, I argue that recipients cannot trace the source or information for relevant applications (i.e., those that are fulfilled by the requirements in section 2.3).

As for the latter two forms, physical privacy poses a unique set of challenges on its own—most systems secure from remote access have physical backdoors—and is considered outside the scope of this thesis. Meanwhile, the definition of events and event correlation assume an ordering amongst events. Some of the data privacy approaches in the proposal do indirectly provide time privacy, but full time privacy poses its own unique correlation challenges; a complete discussion is outside the scope of this work.

- Finally, a **privacy policy** is both a promise by an organization to originators of data contained within the organization, as well as a compliance statement to consumers of data produced by the organization. It may contain one or both of the first two privacy requirements, as well as other additional requirements. Section 8.1 discusses some of these other policies, and how the approach presented in this proposal can be extended to suit them.

Given these definitions, I can now formalize the problem statement as follows.

2.2 Problem Statement

While Internet-scale collaboration offers the opportunity for greater data collection, information sharing and event correlation, the privacy policies of many organizations prevent traditional correlation approaches from being used. Additionally, traditional end-to-end cryptography mechanisms are insufficient, as the problem is not of "man-in-the-middle" attacks (although any collaboration framework should be robust to these), but rather the release of the information itself to cooperating third-party organizations.

Any event correlation framework that wants to support Internet-scale collaboration must provide privacy preservation as an integral feature, and balance the two constraints to provide effective correlation.

2.3 Requirements

Based on our definitions and initial problem statement, I now establish a set of requirements necessary for an effective solution to the problem. Ideally, these requirements should be established as a minimum, but not *imposed* on the target application, as certain collaboration groups *may* choose to share more information where appropriate.

1. **Support event source anonymity.** As described in section 2.1, an event source must, if it prefers, be able to contribute anonymously. Note that this *does not* imply that a given event source cannot be differentiated from other sources, but rather that any one event source cannot be traced back to its origin. As discussed in section 3.3, differentiation is an extremely useful technique for enriching correlation. In certain situations, one may also want to support *classification* in addition to differentiation, e.g., classify source identifiers into aggregate groups without revealing individual identities.

2. **Support data privacy.** In addition to source anonymity, data in an event must correspond to the producer's privacy policy. At the simplest level, this may refer to data sanitization, where

rules are established to determine what part of an event is kept, and what part of an event is removed prior to publication [54]. However, the applications cited in this thesis are interested in correlating the sensitive data itself; scrubbing the data from the event eliminates this possibility, and so a more general approach to transforming event data is required that enables correlation without revealing content.

3. **Support event corroboration.** Clearly, it is impossible to support every correlation scenario when the original data is not present. Instead of requiring completely generalized correlation, which is the research focus of many existing approaches, I relax this definition to require, as a minimum, the concept of *corroboration*, e.g., common dataset intersection—a reduced form of correlation that enables collaborative verification of observed application behavior. While the correlator should, if possible, support richer event correlation when shared data allows it, corroboration remains an option where necessary.

4. **Support privacy-preserving (group) authentication.** While the correlator may be source-agnostic, the producers of events should ideally be authenticated as a means of providing simple trust management (e.g., to differentiate benign and malicious entities). However, any such authentication should not require a producer to reveal their identity to collaborating peers unless they volunteer that information. While there exist approaches that provide a trust model that don't require authentication, they are considered outside of the scope of this thesis. It is important to note that this does *not* require that peers actually transmit event streams through the authenticator.

5. **Support heterogeneous privacy policies.** As already indicated, an organization may volunteer to contribute more information than necessary as per the above requirements. Others may employ a finer-grained differentiation; for example, internal identities may be prohibited from being exchanged, while externally-originating identities may be communicable. The framework must be able to support these heterogeneous requests and support correlation between them.

6. **Heterogeneous application support.** Not only should the framework support different privacy policies, it should support different datatypes for different applications. The notion of privacy-preservation is broadly applicable to more than just one specific problem, and this proposal puts forward the notion of separating the privacy-preservation requirement from the application at hand.

7. **Near real-time performance.** While the framework does not need to support hard real-time requirements, it should support the ability to keep up with event streams during runtime. This is especially significant for applications that generate large numbers of events.

Other requirements have been previously implied, so we simply list them here: The framework must be *Internet-scale*, supporting cross-site and cross-organization collaboration, and it must support *temporal correlation*, enabling the creation of correlation rules with temporal constraints.

3 Hypothesis and Model

I now state the two hypotheses of this research: privacy-preserving mechanisms and event typing. I then present a high-level overview of the model, followed by solutions to the problem of balancing the need for correlation with the need for privacy. Finally, I describe the model details, including event typing, middleware, and a publish/subscribe event infrastructure.

3.1 Hypotheses

I now explicitly state the hypotheses derived from the problems discussed in section 2.

1. *The addition of privacy-preserving mechanisms (source anonymity, data privacy) will enable effective correlation despite organizational privacy-preserving requirements.* More precisely, the implementation in section 5 will demonstrate that temporal event corroboration remains practical even after the addition of both source anonymity and data privacy-preserving transforms.

2. *A typed event-driven framework provides a solution for matching heterogeneous organizational information-sharing policies with different privacy mechanisms.* The model presented in this section, using privacy-preserving event structures and a type-enabled event processing infrastructure, maps to Worminator's use of Bloom filters, as discussed in section 5.2.

3.2 Model Overview

Figure 1 shows the basic workflow of the proposed correlation model.

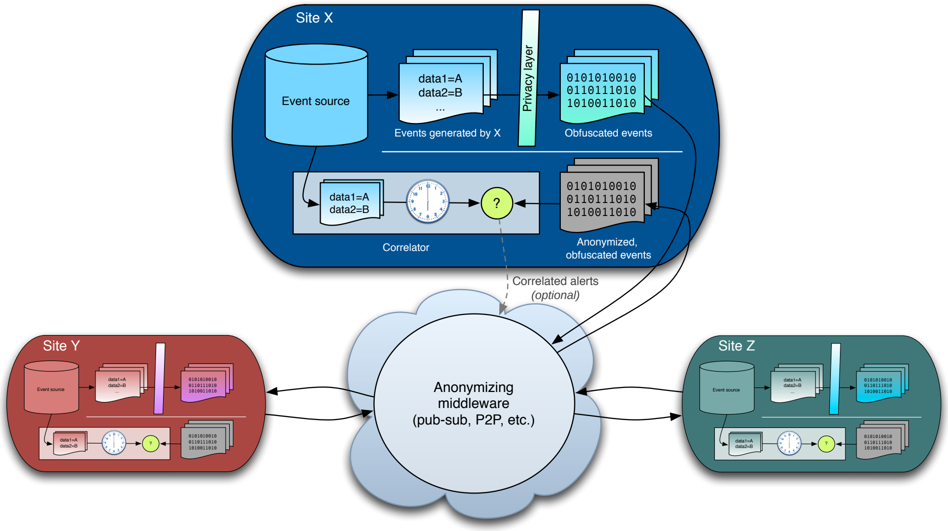

Figure 1: High-level view of model.

The framework runs on top of and connects to an event producer. These events are read, converted to a standardized type (section 3.4.1), and a privacy-preserving transform of the data is applied (section 3.3.2) before it is published via anonymizing (section 3.3.1) event middleware. Meanwhile, anonymized events are received from other sources and are corroborated, with temporal constraints, against existing local event data by applying the same privacy-preserving transforms. Finally, corroborated events may optionally be transmitted back to peers in raw form to enhance their source data set for further corroboration. The model is discussed in greater detail in section 3.4.

3.3 Solutions

This subsection outlines specific techniques used in our model. I address here most of the requirements originally stated in 2.3, including means of accomplishing source anonymity, solutions for data privacy, mechanisms to support efficient event corroboration, and mapping privacy policies to the proposed model. Performance is not discussed here; see the implementations in section 5 for an idea of practical throughput.

3.3.1 Event Source Anonymity

To support event source anonymity, two solutions are adopted: use of an authentication and anonymization authority, and a publish-subscribe event infrastructure. The proposed framework supports the ability to plug-in existing implementations for both, based on the requirements of the actual application. Four different levels of anonymity are defined based on the proposed approach.

- An *authentication and anonymization authority* can be viewed as a trusted third party (TTP) whose primary responsibility is to authenticate participants and to scrub source information before replicating producers' information. This introduces two potential sub-problems: malicious TTPs and centralization.

  A *malicious authority* can do one of several malicious acts: reveal the privacy of contributors; exchange or corrupt identities of contributors, making the differentiability data wrong; corrupt contributors' data; or feed invalid data. The worst-case scenario enables the authority to disrupt event correlation at each of the nodes. It is important to note that the malicious authority *still would not have access to the underlying data*, as the data is encoded in a fashion to preserve data privacy not only amongst collaborating peers but for the authority itself. As for data corruption, this can be mitigated by using a decentralized routing mechanism; this possibility is discussed in the next point. Finally, it is important to note that the problem of determining identities is not generally solvable without such an authority *or* significant computational expense [17], so while a completely decentralized model where *both* anonymized routing and authentication may be possible, this proposal does not address that scenario. [37] uses a TTP for group signature management, and [29] uses a "friendship" trust model. Neither directly enables anonymous differentiability, which is necessary in our correlation scenarios. Additionally, section 5.2 provides a mechanism for selecting such authorities.

  If such an authority has only one host, this introduces a single point of failure into the system, and is vulnerable to DoS attacks. There are numerous approaches to mitigate this. One is to have the authority certify a number of other hosts as trustable to perform the task. Alternatively, the authority can be relegated to just the role of authentication, and anonymity can be provided by utilizing various decentralized anonymous communication topologies, as described in the next point.

- A *publish-subscribe event infrastructure* is a communications substrate that allows for the Internet-scale routing of events by *interest*, as opposed to an explicit source and/or destination. Producers publish events into the event "cloud", and subscribers indicate interest in classes of events, either via subscription to channels or by declaring interest in certain classes of content. The latter requires Content-Based Routing (CBR); see section 4.1 for some examples of publish-subscribe infrastructures. One of these may be chosen as the communication substrate. The router may either exist in the authority's infrastructure, or be distributed, where the authority is used for authentication but events are then anonymously routed through a publish-subscribe cloud.

If necessary (e.g., the infrastructure cannot guarantee anonymity), techniques such as probabilistic memoryless random-walk routing (Onion routing [25], Freenet [12], etc.) can be employed to ensure that the source cannot trivially be traced without a very large number of cooperating malicious entities in the cloud. Note that this thesis is *not* concerned with developing such an infrastructure, but instead leverages it as a communication substrate, and runs on top of it. The implementations discussed in section 5 support the use of different architectures as requirements dictate.

Combined, these solutions enable four different levels of anonymity: non-anonymous, anonymous but categorizable, anonymous but differentiable, and completely anonymous. *Anonymous but differentiable* refers to the notion that while sources (and events produced by sources) cannot be traced back to the original producer, a distinction between sources can be made. This enables techniques such as thresholding on the number of corroborating peers. *Anonymous but categorizable* is a variation on differentiability, where a source may be unknown but can be differentiated *and* categorized into a certain class of sources; for example, a distinction can be made between friends, peers, or unknown people. An event coming from a friend may be valued more highly than an event coming from a truly unknown person. Given a sufficiently large body of friends, an individual friend's privacy is preserved, while providing valuable information for corroboration. This distinction, and applications for it, is discussed further in section 5.2.

3.3.2 Data Privacy

The key consideration to providing data privacy is to transform data before it is published on the collaboration network, yet still allow corroboration. I define the key operations required as *insert* and *verify*: a producer needs to insert values into the collaboration network, and peers need to verify them to determine if they can corroborate what the initial producer saw. For complete data privacy, no other data retrieval mechanism should be supported (e.g., enumerate, get, count, etc.). In other words, a one-way data structure is ideal for this problem. “Public” cryptographic hashing, such as SHA-1 [51], works especially well given this requirement—it is not computationally feasible to reverse the hash, and the likelihood of collisions is extremely low. In essence, the problem is reduced into a set matching problem, like [37].

However, there are three key challenges with using hashing as a model for data privacy: overhead, vulnerability to brute-force guessing the original data, and fast temporal correlation. The first two are addressed here, and the third is discussed later in this subsection.

- The **computational overhead** of hashing for privacy is insignificant, as demonstrated in [42]. However, for small raw events, the **size overhead** difference can be significant—not only for communication, but just as importantly for correlation. Hashes are not intelligent enough to support features like range matching, instead requiring every item in a set to be enumerated into 160 bits (SHA-1) or more. In order to ameliorate this, a compact hash-based structure, like a *Bloom filter* [9], can be employed.

A Bloom filter is essentially a bit array of n bits, where any individual bit i is set if the hash of an input value, $\mathrm{mod}\ n$, is i. A Bloom filter contains no false negatives, but may contain false positives if collisions occur; the false positive rate can be optimized by changing the size of the bit array, as well as using multiple hash functions (and requiring all of them to be set for an item to be verified as present in the Bloom filter). As with a hash table, a Bloom filter acts as a convenient one-way data structure that can contain many items, but generally is orders-of-magnitude smaller. Operations on a Bloom filter are also $O(1)$, keeping computational overhead low. The choice of hash functions used essentially acts as a symmetric key/shared secret encrypting the contents of the Bloom filter amongst all participants, obfuscating it to outsiders. (As I describe shortly, the model does *not* rely on keeping the choice of hash functions secret.)

- However, depending on the potential input size of produced events, **brute-force guessing** remains a potential vulnerability both with hash tables and Bloom filters—especially amongst curious insiders who know the hash functions used in transmitted Bloom filters. This is a significant problem with the implementation discussed in 5.2, where the space of input is as simple as IP addresses and ports, with an upper bound of 2^{34} possibilities, well within the range of a fast brute-force guessing algorithm (in fact, clever guessing techniques can restrict the number of addresses, e.g., ignoring network, multicast or private address spaces searched to something substantially smaller).

  [42] sketches the beginning of a fast, hashed-based solution, employing a combination of known *and* unknown hashes (e.g., private-keyed HMAC [39]) to add "noise" into a hash table to make it computationally infeasible to determine legitimate and illegitimate entries, and [7] suggests the probabilistic copying of bits into a Bloom filter to introduce noise in a similar fashion. I use a variant of these techniques to make it significantly more difficult to brute-force: given a set of inputs I and h public hash functions, select some subset I' of inputs with probability p, and insert them $h + 1$ times into a Bloom filter of size n bits. The first h are inserted using the standard set of hash functions, and the $h + 1$th is inserted using a privately-keyed hash. The private key is never released by the producer, so the bits set in the Bloom filter via the $h + 1$th hash appear as random and serve as noise, yet can be verified by the original producer if necessary. With appropriate choices for h, p, and n, this technique can result in significant obfuscation, making brute-forcing practically infeasible. Mitzenmacher [49] has developed metrics for the collision (false positive) rate; an evaluation of appropriate choices based on those metrics, and results corresponding to data collected in section 5.2, is amongst the proposed work in section 6.2.

A Bloom filter is not the only privacy-preserving data structure; a hash table or a Patricia trie are also possible ([27], [50], [42]). However, a Bloom filter's compactness and optimizability make it attractive for event communication.

3.3.3 Event Corroboration

Given the data privacy and anonymity mechanisms described above, we can trivially support set membership tests: given a set of inputs I and a Bloom filter B, it's straightforward to determine the intersection of I and B by enumerating through the items in I, hashing them, and testing the appropriate bits in B. False positives from 3.3.4 are ameliorated by scaling up the number of Bloom filters being exchanged: given $B_{1...n}$, i.e., Bloom filters from n peers, the likelihood that the *same* hash collisions occur at every peer decreases as n increases, i.e., the "global" view of corroborated events filters out the "local" noise from any individual Bloom filter.[1]

Event-driven corroboration. However, event corroboration does not imply a one-time matching against n Bloom filters; instead, the n peers will continuously transmit events composed of Bloom filters (and other formats, as well). Assuming sub-minute generation intervals (as with Worminator alerts in section 5.2), corroboration will have to be performed against orders-of-magnitude more Bloom filters than just n, which rapidly becomes very expensive in both memory and time.

A naive approach to solving this problem is to simply keep *merging* the Bloom filters as they arrive; Bloom filters support set union by continually ORing the corresponding bits together, assuming the filters are of the same size and use the same collection of hash functions. However, this will eventually cause the bit array to be set to all 1-values, resulting in many false positives. (In fact, the false positive rate will steadily increase much earlier.) Instead, an *aging* Bloom filter is needed.

[21] introduced the notion of *counting Bloom filters*, which stores an integer value in each array cell instead of a bit value, and thereby supports deletion via a mechanism extremely similar to *insert*. However, counting Bloom filters perform poorly here: if the values initially inserted into the Bloom filter are unknown, as when the filters are exchanged between peers, this form of expiration is infeasible. [27] and [11] described two mechanisms to support aging: a *cold cache*, whereby the Bloom filter is emptied when it crosses a certain threshold of fullness, or *double buffering*, which allows a staging empty operation. Double buffering employs a second Bloom filter that is initially empty, but is increasingly filled as the first becomes too full or when a predetermined time duration passes. Once the first one has crossed the appropriate threshold, the first one is emptied *and the role of the two filters are switched*, allowing for a graceful emptying operation.

These approaches, while an improvement, still do not allow for flexible aging mechanisms; for example, a specific rule in a temporal correlator may want to only examine those entries that were present at timestamp t or later (for example, to corroborate only recent events), or even to support range timestamp queries (find all common events between t_1 and t_2). This proposal introduces two new variations on Bloom filters that accomplish this purpose an order-of-magnitude faster than individual Bloom filter enumeration.

1. A **Most-Recently-Used (MRU) Bloom filter** is a simple variation on a Bloom filter that uses a similar concept to the counting Bloom filter, but instead of storing an integer representing the number of events, a timestamp is inserted. Typically, only one MRU Bloom filter needs to be instantiated per site and class of event being corroborated, mitigating increased memory requirements significantly.[2] This MRU Bloom filter then supports *union* as a $O(n)$ time operation by storing timestamps instead of 1-values, e.g., this is a multiplicative-time overhead over a standard union (which is also $O(n)$, although it's a small fraction of a union with a MRU Bloom filter). Additionally, *verify* remains $O(1)$, and two additional operations can be added: *expire*, which removes bits that are older than some time t in $O(n)$ time, and an operation called *generate* which generates a "regular Bloom filter as of time t" by scanning the timestamp values and extracting those bits who fit the threshold (again linear time). The latter operation may be

[1] Again, a precise evaluation of this with real data is part of the proposed work.

[2] A typical UNIX timestamp is 64 bits; offsetting from a more recent start time and using coarser granularity can reduce this requirement further, down to less than half.

useful if the *union*ed Bloom filter needs to be stored for historical purposes, or transmitted to others in a space-saving fashion.

2. The MRU Bloom filter still does not accommodate range queries, i.e., "was event e seen by entity E between t_1 and t_2?". Such range queries become significant when coordinating a large number of Bloom filter submissions, especially because sources may not publish at any coordinated time intervals. To accommodate this, a variation of the MRU Bloom filter can be designed: a **timestamp Bloom filter**. Instead of a single MRU value in place of each bit, a reference to a range-capable data structure (e.g., balanced binary search tree or variant thereof[3]) is stored, and each individual timestamp value is stored during a *union* or *insert*. As expected, this affects both memory and computation time requirements. Since a timestamp Bloom filter would still primarily be used for correlation, like the MRU filter, memory is less of a concern. However, the cost of most operations increase; given n bits and an average of m timestamps per bit: *insert* and *remove* become $O(\lg m)$, *union* (with incoming non-timestamp BFs) is $O(n \lg m)$, and *expire* and *generate* become $O(nm)$. Two *verify* operations exist—one with a range lookup and another without—and their computation times range from $O(1)$ to $O(\lg m)$. Most of these are mitigable because the average value of m is significantly smaller than that of n, and a reasonable expiration policy keeps the value of m manageable.

Timestamping. Given data structures to support efficient correlation of timestamps, there several different possible timestamping mechanisms, depending on the level of privacy preservation and the need to do reliable correlation.

- Have an event producer add a single timestamp to a Bloom filter before sending it out. The timestamp, in this case, is an upper bound on the event corroboration contained in the Bloom filter. (Alternatively, both upper and lower bounds could be included in the Bloom filter's metadata.)
- Utilize a timestamp from the publish/subscribe event infrastructure, if available. If the clocks are synchronized (either in an absolute or logical [41] sense), this alternative produces similar results to having the producer timestamp the clock.
- Have the recipient add a "received" timestamp. Given a synchronized, low-latency publish/subscribe network, this produces results almost as good as having the producer timestamp the event. In a more distributed/P2P environment, however, this may lead to ordering errors.
- Have the producer send an MRU Bloom filter instead of a regular Bloom filter. This is potentially risky from a privacy-preserving perspective, however; if multiple known hash functions are used, the problem space needed for brute-force guessing decreases dramatically, as a curious participant can cluster the bits with equal timestamps. This can be mitigated by introducing sufficient noise to minimize the number of clusters, rounding timestamps up or by using fewer (one) hash function, albeit with a potential increase in false positives. It is important to note that this does not guarantee accurate timestamps for every event, as events may collide when hashed into the Bloom filter. While transmitting timestamp Bloom filters would provide more data, it still does not authoritatively assign timestamps to any individual event, and the space overhead of a timestamp Bloom filter is significant.

[3] Other data structures, such as a skip list, may be more appropriate depending on the frequency distribution of individual operations and based on insert patterns, e.g., a tree may be expensive if inserts are only done in ascending order.

- Do not use timestamps at all. This reduces the corroboration problem to that of set intersection, and is not explored further in this thesis.

The goal with timestamping is to establish, as a minimum, an *ordering* on the arrival of Bloom filters, but ideally, as close to original event timestamps as possible. Having accurate timestamp data enables accurate temporal constraints on correlation, which reduces state maintenance in a correlator (as described in 5.1).

Raw corroborated events. One potentially appropriate mechanism to extend limited corroboration to more generalized correlation is to establish policies releasing underlying event data *if* it has been corroborated by a sufficient number of peers; the fundamental idea is that this event is no longer a secret, so obfuscation is no longer necessary. This technique proves useful if only a fraction k of the participants can corroborate the event; the remaining $1 - k$ fraction of participants remain in the dark, as they cannot test the appropriate bits of the Bloom filter. By sending out the raw event itself, these remaining participants can include the event's information in their correlation scenarios. This model proves especially useful in section 5.2, where we define the notion of a privacy-preserving "watchlist" vs. a more explicit "warnlist" of correlated suspicious sources.

Multiple entry types and aggregate matching. Despite the fact that Bloom filters and other hash structures only support individual set membership tests, the "generic container" aspect of Bloom filters make it natural to hash different datatypes into the same Bloom filter. Not only does this make it more difficult to brute-force, it allows a primitive form of aggregate matching.

For instance, in the case of hashing IP addresses, an alternative algorithm would be to hash not only the IP address (1.2.3.4), but the corresponding class C network address (1.2.3.0), the class B network address (1.2.0.0), and so on. This increases the density of items in the Bloom filter; a strategy of increasing the Bloom filter size and then employing compression [49] can be employed where appropriate to alleviate this. While this may require *a priori* knowledge of the ideal aggregates to hash, versioning (see section 3.4) can be used to gracefully deploy such aggregates after-the-fact.

3.3.4 Privacy Policies

As previously discussed, different producers may have different privacy policies dictating required levels of anonymity and data privacy. Based on the models proposed above, combined with the notion of *typed* events, a correlator can seamlessly support heterogeneous types of events.

Varying anonymity. Section 3.3.1 discussed four different levels of anonymity. To support these heterogeneous levels of anonymity, a uniform source identifier model is used, but different values are filled in based on the level of anonymity required.

- For **completely anonymous** sources, the source identifier is left blank. This is a degenerate condition and can be considered a special case of the other scenarios, and is not explored further in this thesis.

- **Differentiably anonymous** sources use a randomized-but-consistent string when producing events—either a universally unique identifier (such as UUID [28]), or an authority-assigned unique identifier. This identifier is created at the beginning of a correlation task and is not changed for the duration of the task. Optionally, producers may choose to change the identifier between each correlation task in a long collaboration to further obscure their identity.

- **Categorizably anonymous** sources use a combination of a *category identifier* (e.g., SIC codes for corporate sources [53]) along with a unique identifier.
- Finally, **non-anonymous** sources would simply use a descriptive string in the source identifier, based on a pre-agreed standard naming scheme (e.g., hostname of the sensor).

Varying privacy. While Bloom filters are the main encoding structure in this thesis, the framework by no means should be limited to only the use of Bloom filters. If a lower level of data privacy is required, less obfuscating solutions can be used. For example, some sources are only concerned with obfuscating *internal* information, while external data can be transmitted verbatim. The inclusion of raw events alongside hashed events does not pose a significant challenge for corroboration; polymorphic definitions for *verify* can be adopted to make the actual process near-identical. The correlator itself, however, does need to do type-checking and reject more complex correlation operations when the data structures do not support it, e.g., verifying a single attribute of an event is not explicitly supported by Bloom filters (unless individual attributes are encoded). I discuss this and other typing issues in 3.4.

Privacy specification language. Currently, there exists no language for collaborating peers to specify event privacy policies. The closest standard is the W3C's P3P language [72], but this only specifies privacy to end-users, and does not completely address more generalized data interchange. This thesis does not completely answer the question; the implementation in section 5.2 does utilize an XML-based configuration file to configure event processing, including privacy-preserving and anonymity transforms, but does not present a general approach and solution to the exchange of privacy policies. The work in this thesis motivates further research in the subject; see section 8.1 for proposed future work in developing such a language.

3.4 Model Details

Given the model overview and specific solutions to be used, I now develop the model implemented in section 5.

The model features three main components: a typed event model, type-driven event processing and correlation middleware, and a publish-subscribe anonymizing event infrastructure.

3.4.1 Event Model

Type-driven event models are not new; [46] suggests that anything from XML to Java objects can be used for typing. Sun's Java Message Specification (JMS; see [67]) is a event message broker that supports the transport of arbitrary Java types. In fact, the model specified here builds on top of standard OO type structures, and the implementations in section 5 are based on Java type declarations.

However, a distinction is made as the type hierarchy in this approach directly contains privacy-preserving mechanisms, and metadata associated with such events supports heterogeneous correlation.

Hash-based data structures. As implied by the solutions above, the model natively can support the *insert*/*verify* mechanisms as implemented by various hash structures (e.g., hash tables, Bloom filters, and optionally Patricia tries). Single event hashes are also supported to create simpler hash tables, if desired.

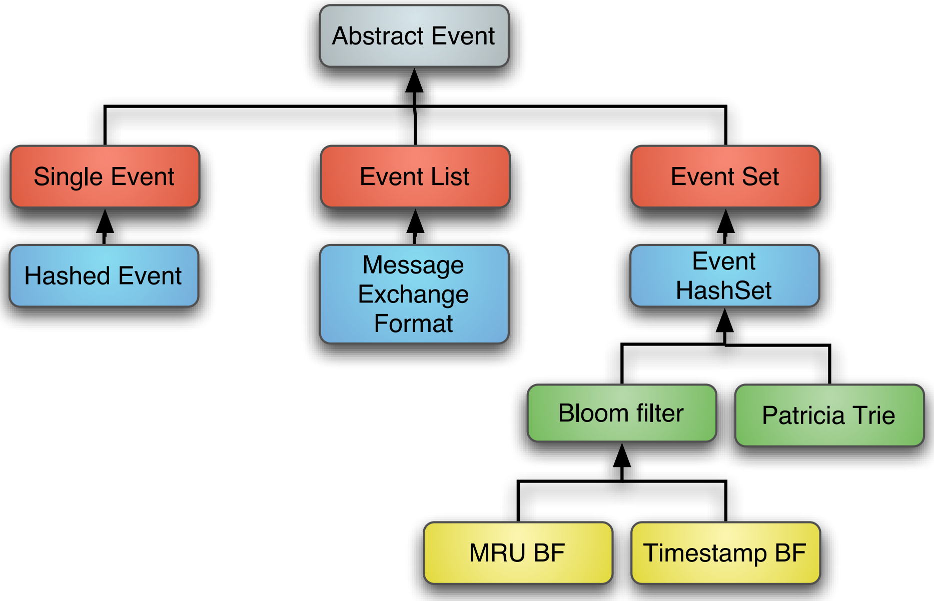

Figure 2: Base event type hierarchy.

This typology maps nicely to standard OO mechanisms. Given the ability to do type introspection, a correlator can differentiate between a hash verification-based corroboration as opposed to a more generic predicate-matched rich event correlation operation, and allow or deny appropriate correlation mechanisms as appropriate.

Encapsulation and message exchange. Since hash structures are themselves events, several encapsulation possibilities exist, such as a list of Bloom filters. Message exchange is often defined as a subtype of event lists, as a number of exchange formats, e.g., [14], [16], act as generic containers. For the standards that do not allow for multiple messages per standardized event, a degenerate 1-element list can be adopted without having to violate the type hierarchy.

Type metadata. The abstract event has some important metadata, needed for correlation. The more important among these include the following:

- A **unique identifier**. Techniques like UUID [28] can again be used here, or the event distribution infrastructure may itself introduce semantics for identifiers. It is critical that such identifiers not be tied to the data or the source of the data.
- The **base type** of the event(s). This is needed to help the correlator determine if it can support the heterogeneous corroboration of a given hash-based event against a given raw event. (For a non-hashed event, this will contain the type name itself; non-hashed lists and sets typically leave this blank and let the constituent events report their own base type.
- The **source** of the event. As discussed in 3.3.1, this is typically a single identifier, but may contain different information based on privacy policies.
- The **timestamp(s)** of the event. As a minimum, this refers to the generation time of the event itself. If possible, there should be at least three timestamps: the creation time of the event, the timestamp of the earliest embedded event, and the timestamp of the latest embedded event. The

latter two are significant in supporting temporal constraints for a (hashed) list or set.

- **Preferred encoding**. This is primarily for implementation purposes, especially when standardized exchange formats are used—most contain an "other" field which is necessary to transport hash-based data structures, as most of the data remains opaque to the format itself.

- The **generator** of the event. Depending on privacy concerns, this may not be filled, but in general this refers to the class of event producer. This is *not* equivalent to the source of the event; the generator typically refers to the piece of software (e.g., sensor) whereas the source refers to the organizational identity of the producer.

- The **version number** of this event type. The model supports a primitive versioning scheme, allowing for type evolution and run-time type changing. The primary intention of this versioning is to ensure participants keep types synchronized; this is especially significant when evolving privacy policies and the corresponding updates of data structures to match those policies, in which a event distributor or correlator may reject versions deemed too old to use for safe exchange. This thesis does *not* generally address type versioning, although the model does allow for more flexible schemes; possible future versioning extensions are discussed in section 8.1.

3.4.2 Pluggable, Event Type-Driven Middleware

Given a standardized, type-driven event model, this proposal now discusses the key pieces of the software framework.

- **Type modules.** Each of the above types are translated into first-class objects. The solutions in section 5 implement these types as a collection of Java interfaces and classes. In addition, *translation facilities* are written to support conversion from one event format to another when possible. For example, during the transmission phase, events are converted into an instance of a standardized messaging event, which then has code to serialize itself into a compatible wire format when being sent. This translation facility is defined at the abstract level and made concrete at lower levels where appropriate.

- **Transform modules.** In the cases where types cannot be directly converted to each other, transform modules are custom-written. In particular, a number of *aggregators* are used when converting single events into lists or sets. Configuration directives dictate appropriate aggregation intervals (e.g., based on time or volume of event flow).

- **Correlation modules.** This is the heart of the framework: type-driven temporal correlation. Depending on the scenario, a custom correlator can be developed as its own standalone module—or a general rule-based correlator can be adopted. Examples of general correlators include our Event Distiller (see section 5.1) or a third-party solution, e.g., [70].

  Third-party solutions, however, are unaware of how to handle privacy-preserving corroboration, so employing them would require a two-phase correlation mechanism. The first phase employs a dedicated corroborator (see section 5.2 for an example), whose job is to perform temporally constrained corroboration against incoming privacy-preserved events, and to update *local* event data with corroboration results. For example, a field may be added to the local event schema specifying the number of peers who have validated a particular event. At this point, the third-party correlator can execute rules based solely on the local data. (Optionally, a transform could then be written to take generated resulting events and make them privacy-preserving before the results are transmitted.)

- **Other modules.** Finally, other modules are present to enable communication in and out of the framework, including mechanisms to connect the framework to the event publish/subscribe architecture, to potential backend databases, and even to reporting/frontend modules. These are essentially middleware "glue" and are not elaborated upon further here.

These modules are tied together by a runtime middleware engine that provides module lookup, event routing and implicitly calls conversion facilities as needed. Ideally, this engine should be written on a platform that supports late binding and/or dynamic libraries, to support the installation of new type versions and correlation modules during runtime. The implementations discussed in section 5—the Event Packager in 5.1, and the Worminator engine in 5.2—are written in Java, which support both.

3.4.3 Publish/Subscribe Event Infrastructure

As stated in section 3.3.1, this thesis is not concerned with implementing a new publish/subscribe event infrastructure from scratch. However, the notion of an event-driven publish/subscribe infrastructure remains intrinsic to the proposed thesis model, and in addition to the anonymity and authentication issues as previously discussed, other considerations must be taken into account.

- **Channel vs. CBR subscriptions.** In theory, content-based routing enables the ability to reduce the amount of data communicated: peers can submit filters indicating interest in certain classes or instances of events. This holds true for privacy-preserving transformations: peers can submit Bloom filters, indicating only interest in those filters that overlap their own submission. This enables pushing the filtering problem deeper into the network infrastructure and minimizing replication to only those nodes where appropriate; [27] describes this mechanism in greater detail.

  Absent a flexible CBR, channel-based subscriptions can be used: in essence, each subscription channel becomes a peer group. In theory, more degenerate "channels" can be leveraged, e.g., multicast; these possibilities are discussed in section 8.1.

- **Event type compatibility.** Ideally, the infrastructure is type-compatible with the definitions in section 3.4.1, as that minimizes the effort to marshal data to and from different formats, and simplifies communication workflow. JMS and MEET[4] are two that have more generic type facilities.

  However, most pub/sub infrastructures do not offer generic typing facilities. For instance, Siena [10] supports primitive-typed attribute-value pairs. As a worst-case scenario, the proposed middleware should support primitive String datatypes and serialize more complex types into a publishable form.

- **Event latency and ordering.** Again, an ideal pub/sub infrastructure minimizes latency and guarantees ordering. However, anonymity and distributed mechanisms increase latency and may, in a P2P deployment, disrupt ordering. In order to effectively correlate, the proposed framework must support a *reordering* mechanism, which would employ a sliding window and

[4] Multiply Extensible Event Transport, a pending thesis proposal of my colleague, Phil Gross.

reorder incoming events by their generation timestamps. The implementation of Event Distiller, in section 5.1, features such a mechanism. If these timestamps are absent, reordering becomes infeasible and correlation must be performed in a non-order-specific manner, such as set intersections.

- **Encryption** is one last optional service to be offered by a pub/sub infrastructure (optional, as privacy-preserving hashes act as a shared secret, but potentially important if privacy-preserving transforms are not used). Ideally, a pub/sub infrastructure supports distinct per-channel permission and encryption models, enabling distinct simultaneous correlation groups and enabling scenarios where certain classes of peers may not be able to interoperate with each other. However, many CBR-based infrastructures do not support encryption due to the significant cost of decrypting/encrypting data at each node to make routing decisions.

Very few publish/subscribe systems offer all of the above features, plus a robust authentication, anonymization, and distribution infrastructure. Nevertheless, by layering the proposed framework on top of a third-party infrastructure, different infrastructures can be adopted based on the requirements of the collaboration environment.

4 Related Work

This proposal and the implementations discussed in section 5 relate to four main subclasses of work in the software engineering and network security/intrusion detection fields.

4.1 Event Correlation and Event Systems

Event correlation is a difficult field to summarize, as nearly every specialization in Computer Science has its own event correlation techniques. I confine my focus here to some of the seminal event correlators in the software engineering and networking communities; some intrusion detection-specific event correlation papers are discussed in section 4.2.

- Luckham et al. describe *Rapide*, an "event-based, concurrent, object oriented language" for describing event patterns to determine event causalities and Root Cause Analysis (RCA). Rapide has been employed for a broad variety of applications; [47] discusses the most important context, which is the enabling of specification and analysis of system architecture. It enables declarative programming of system and communication architecture for system monitoring and simulation. [46] is a book by the same author describing Rapide as an event pattern language, and supporting a broader array of event-driven "Complex Event Processing".

- Yemini et al.'s work on *DECS* [78], or Distributed Event Correlation System, describes techniques for very high-speed event correlation for network and distributed application management. The "codebook approach" essentially decomposed events into "codes" in bit vectors, enabling very high-speed lookup and near real-time performance. DECS is paired with the Netmate network modeling tool [19]; they used this architecture to model a distributed database application for decentralized network problem detection (e.g., communication failure).

There are many other event correlation approaches, but virtually all of them (with a few notable exceptions, discussed in section 4.3) assume access to raw event data, and the associated rule languages assume the ability to perform predicate tests on whole events. Rapide provides a very general, flexible approach to event correlation, including time constraints, but does not include any privacy-preservation; in theory, an implementation of this thesis model could be built as a framework on top of Rapide (as it provides a general programming language), but the project is currently unmaintained [5]. Yemini et al. use a more compact and fast representation, and their transformation to bit vectors shares some similarities with the data structures used in section 5.2, but do not embed privacy-preserving transforms.

In addition, recent work on publish-subscribe systems—including, in particular, content-based routing (CBR)—has motivated the inclusion of basic event correlation techniques in the publish-subscribe architecture itself as a means of reducing (or summarizing) information, thereby reducing communication overhead, making routing decisions simpler, and supporting more flexible subscription models.

- *Elvin* [64] is perhaps the best known CBR publish-subscribe system; it supports single-message predicate matching, but does not correlate between events.
- Carzaniga et al. [10] built *Siena*, a Scalable Internet Event Notification Architecture, with a flexible subscription language featuring predicate matching. In addition, support for “sequence subscriptions” was added to the language where predicates can be declared on a sequence of events; the subscriber then receives the sequence of notifications that matched the subscription.
- IBM’s *Gryphon* [6] has the most extensive correlation facilities available for a CBR publish-subscribe solution; “event stream interpretation” [79] allows for both the semantic reduction of data, i.e., Gryphon will rewrite a stream of events as a single event, and optimistic delivery to counter the slowness of stream processing.

By definition, most CBR publish-subscribe systems need access to the full event data, as they are concerned with efficient routing decisions at every node in the network, and perform predicate matches as the event is routed. Privacy-preserving mechanisms frequently obscure data, breaking predicate-matching functionality. We have done some preliminary research in supporting privacy-preserving CBR publish-subscribe; see [27].

4.2 Distributed Intrusion Detection

A Distributed Intrusion Detection/Collaborative Intrusion Detection (DIDS/CIDS) system is one that employs *multiple* Network Intrusion Detection and/or Host Intrusion Detection sensors (NIDS/HIDS), often across multiple local area networks, and correlates resulting alerts to get a broader picture of Internet-based threats.

- *GrIDS* by Staniford et al. [65] is perhaps the earliest well-known DIDS; its name stands for a Graph-based Intrusion Detection System. The graph forms a hierarchy (tree); data is aggregated at lower-level nodes and summaries are passed up the hierarchy. A rule engine is used to determine appropriate aggregation.
- SRI’s *EMERALD* (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [59] is not formally a DIDS—it’s technically a NIDS—but the authors point to its distributed component model as a “composable surveillance” that scales better than a centralized monitor. Components include service monitors that act as sensors, and domain/enterprise monitors

that aggregate and interpret sensor information. EMERALD uses event communication mechanisms, but does not dictate a format to be used. It is primarily geared towards signature analysis.

- *Quicksand*, by Kruegel et al. [40], is close to a hybrid of the first two; it is a completely decentralized architecture, like EMERALD, but uses a rule engine to implement a distributed pattern language (Attack Specification Language, or ASL). Quicksand is particularly designed for a large network with lots of sensors, although their experiments focused on a single departmental network.
- Janakiraman et al.'s *Indra* [32] is one of the first DIDS platforms to explicitly support a P2P-based communication platform. They also stress that most existing DIDS platforms only collect data for humans, and that an autonomic system is more appropriate. Their architecture leverages the Scribe publish-subscribe protocol [63] running on top of Pastry, a generic P2P routing substrate. However, at the time of publication this architecture was theoretical—the implementation was at an early prototype stage.
- *DShield* [69], built by Ullrich, is the most active DIDS project on the Internet. It is volunteer-based; anyone can download a client and submit alerts to DShield. DShield then regularly produces "top 10"-style reports and blacklists that enable people to update their filters to common threats.
- *DOMINO* by Yegneswaran et al. [77] organizes a decentralized, heterogeneous collection of NIDS sensors into "satellite" and "axis" nodes, with satellites talking only to axis nodes and the axis nodes forming a overlay graph between which encrypted traffic is exchanged hourly. The architecture is mostly theoretical; a small experimental subset has been implemented, featuring an "active sink" honeypot to collect potentially malicious payloads. More importantly, the paper measured, using DShield alert logs, the notion of information gain, and concluded that 40-60 sites enables building summaries and/or blacklists with high degrees of confidence, and that very few IPs dominate the alerts.

There are significant differences between the work presented in this proposal and the works cited above. First, none of the above approaches are privacy-preserving, although some of them use end-to-end encryption to prevent sniffers and man-in-the-middle attacks. Second, with the exception of DShield (and the corresponding analysis in [77]), none of the projects above offer a significant implementation or deployment to verify the hypothesis with collected data; section 5.2 details our approach—Worminator—and preliminary results collected from deployment. Finally, Worminator's modular framework allows for the separation of event type, application, correlation engine, and the underlying communication substrate, allowing for a broader set of solutions.

4.3 Privacy-Preserving Collaboration

The works referenced here are most similar in nature to the proposed work. In particular, they make privacy preservation one of the key requirements, and support it to varying degrees.

- Lincoln et al. [42] describe a privacy-preserving mechanism for sharing security alerts, and addresses several techniques to sanitize alert data, including scrubbing and hashing. They propose the use of multiple hash functions, some keyed, to build solutions that avoid dictionary attacks. They also employ multiple repositories that randomly forward alerts to each other to obfuscate event sources. There appears to be no implementation of the above model; however, they conducted some small performance tests of hashing and correlation overhead (which is minor).

As opposed to Lincoln, et al., this thesis is application-agnostic; while section 5.2 discusses a similar, privacy-preserving CIDS, the infrastructure behind Worminator can be applied to other forms of intrusion detection or software fault correlation. Indeed, the future work section (8.1) suggests approaches to support privacy-preserving collaborative payload anomaly detection. Additionally, this thesis introduces the notion of a framework to enable scalable, heterogeneous privacy-preserving mechanisms, while [42] focuses on a fixed basket of techniques. Finally, Worminator has several significant differences to enable practical deployment, including the use of Bloom filters, fast Bloom filter correlation techniques, and publish-subscribe infrastructures. To the best of my knowledge, the work proposed in [42] remains unimplemented, and in fact postdates much of the Worminator work.

- Kissner just completed a thesis proposal titled "Privacy-Preserving Distributed Information Sharing" [37], and some of the results are published in [38]. In the thesis proposal, she outlines two different privacy-preserving mechanisms: a polynomial set representation that supports not only privacy-preserving intersection, but also union and element reduction (i.e., set count difference), and a pair of *hot item* algorithms, one defining an identification mechanism and the other defining a publication mechanism. The latter two algorithms are closest to the use of Bloom filters [9] in this thesis; in fact, the HOTITEM algorithms use bit vectors that closely resemble Bloom filters. Her notions of data and owner privacy also closely correspond to the definition of source anonymity and data privacy in section 2.1. However, there are significant differences.

  1. The proposed algorithms focus primarily on sets and associated membership tests. The work proposed here takes an event-first approach, where data have timestamps closely associated with them and used in correlation. The use of time-based correlation changes the algorithmic approach significantly.

  2. The model proposed here (see section 3) assumes the *differentiability* of sources as being privacy-preserving. The work proposed by Kissner enables either owner non-privacy or complete owner privacy, but does not directly correspond to the indirectly differentiable-but-anonymous model. The correlation techniques leveraged here require the use of differentiability, and even classification in special cases.

  Most significantly, her thesis proposal focuses on a theoretical, cryptographic approach to set operations and hot item identification; this thesis uses a set of algorithms very close to her HOTITEM work, but focuses on the correlation itself. An alternative version of the framework described here could be developed with her algorithms, and in fact may serve as interesting future work. As a result, I view the theses as more complementary than competing.

- Huang et al. describe *Privacy-Preserving Friends Troubleshooting Network* [29], which extends Wang et al.'s PeerPressure research [73]—a collaborative model for software configuration diagnosis—with a privacy-preserving architecture utilizing a "friend"-based neighbor approach to collaboration. The key relevant aspects of the paper include a variation of secure multi-party computation problem to "vote" on the popularity of a configuration to determine the configuration outlier, and the use of hash functions to enable secure multiparty computation

(SMC) to support an unknown set of values; the relation of this proposal to SMC is discussed further in the next section. Finally, as with the previous work, they do not address temporal constraints in their correlation mechanisms.

4.4 Other Privacy-Preserving Computation

Statistical transformation. If the data to be correlated is of significant size, a statistical transformation of the data (such as a frequency distribution) is a convenient privacy-preserving mechanism; not only is it extremely difficult to reconstitute the original content given the distribution, but the distribution can also be transformed in various ways to make the distribution itself difficult to reconstruct.

- Wang's *PAYLoad Anomaly Detector* (PAYL [74], [75]) uses the combination of a 1-gram statistical model with a learning anomaly detector to build extremely accurate "normal" models of network packet payloads, and can effectively label anomalous traffic. Additionally, PAYL can generate a signature of ingress/egress correlated traffic, which can be represented as a substring of the original payload *or* as a privacy-preserving Z-string (Zipf distribution) that cannot be used to reconstruct the original packet(s).

Such techniques are complementary to this thesis; in fact, section 8.1 details a hypothetical PAYL-Worminator integration scenario. Other statistical modelers could be applied in a similar fashion. The main body of this proposal does not address them further, as the feasibility discussion in section 5 focuses on small event payloads, to which cryptographic hashing techniques apply better than statistical transformation.

Privacy-preserving databases and data mining. There is a tremendous volume of work on various aspects of data mining and databases, including secure query mechanisms, statistical databases, secure indices, etc. I mention here some of the relevant classes of work.

- **Statistical databases** refer to query engines that may return individually restricted or perturbed results as privacy-preserving measures; Agrawal et al. [4] demonstrates that accurate aggregations can still be reconstructed via decision-tree learning. Lindell and Pinkas [43] expand on this notion and discuss mechanisms to **securely build decision trees** over the union of two otherwise private databases, enabling aggregate data mining operations. However, statistical databases and aggregate data-mining do not directly apply to precise matching applications as discussed in this proposal.

- Agrawal et al. [2] also presents mechanisms for supporting **privacy-preserving information sharing across databases**; in particular, they develop secure two-party intersection, equijoin, and cardinality algorithms. Strong privacy guarantees are accomplished by the use of commutative encryption.

  This approach—and many others—fundamentally assume two-party interaction, either based on the querier-database model or set operations on two information-sharing databases. In general, these models do not generally scale to multiparty event correlation.

- Instead of obscuring data, the **k-anonymity model** (Sweeney [68] and others) argues disclosure is not an issue as long as the data has been scrubbed of source information *and* individual sources cannot be correlated via any subquery of the raw, anonymized data. The *k* refers to the minimum number of records that must have the same value for a given column on any subquery.

While this property can be maintained for statistical aggregation models, this restriction would be difficult to enforce on a generalized event correlation framework.

- In certain scenarios, the **search queries and/or indices may need to be privacy-preserving**; Bloom filters [9] play a significant role in this requirement. Bellovin and Cheswick [8] support privacy-preserving queries through a "semi-trusted" third party to accomplish this goal; Bloom filters are used as search keys. For indices, the goal is to prevent the index from revealing content that was indexed within it. Bawa et al.'s privacy-preserving index [7] uses *content vectors* constructed of Bloom filters. Finally, Goh [24] uses Bloom filters as a *per-document index* to track words until they are integrated into a *secure index*. The index itself is secured by requiring queriers to obtain key-based trapdoors for words. In all of these cases, as with information sharing, the model is two-party interaction, and the database is typically trusted; while this proposal uses Bloom filters, it relies on a multi-party, untrusted model.

Additionally, most of the research in this field is more concerned with offline analysis, as opposed to near real-time event correlation, and the adopted algorithms and mechanisms differ correspondingly. [3] is more open-ended, and presents a strawman for a "Hippocratic database" design that respects privacy, regulatory, retention and other policies. Many of the approaches covered are complementary, and could be applied as an extension to the proposed framework.

Secure multiparty computation (SMC) [76] is a theoretically attractive way to accomplish privacy-preservation; certain forms of correlation can be fashioned as such a computation problem. For example, a SMC approach to intrusion detection would formulate intersection as a function to be executed over every participant's input. Du et al. [18] discuss a model to transform standard computation problems to secure multiparty computations, and review, amongst other problems, the possibility of sharing intrusion detection information. However, existing SMC algorithms are considered too expensive [42] to handle large event streams such as the ones presented in this proposal.

5 Feasibility

I now discuss two different implementations of the model: XUES (**X**ML **U**niversal **E**vent **S**ervice), an event processor used as part of an autonomic software monitoring platform, and Worminator, a Collaborative Intrusion Detection System. XUES was a first (incomplete) version of the model, but laid the groundwork for the complete implementation in Worminator. The motivation of each application is discussed, followed by a brief overview of the architecture and a look at the results produced so far.

5.1 XUES

XUES was part of a research effort called KX (**K**inesthetics e**X**treme), a work funded by DARPA to produce lightweight, run-time legacy application monitoring solutions based on *sensors*[5], *gauges*, *controllers* and *effectors*.

[5] The actual proposal termed them *probes*, but this term is not used to avoid confusion with the Intrusion Detection use of the word, which refers to potentially malicious scanner entities.

Sensors were inserted into a legacy system (via code hooks, network traffic sensors or the like), collected raw event streams, and sent them to the monitoring layer for processing. Gauges then correlated the received events to determine if abnormal behavior was occurring. Any such abnormal behavior would result in meta-events being sent to the controller, which used workflow to orchestrate a set of effectors to fix the symptom. An effector was an (optionally mobile) code module that applied a repair mechanism to the legacy system, ranging from a simple reboot or service restart to a more complex reconfiguration to fix the problem permanently. A *behavioral model*, such as a software architectural model, e.g., [23], specified the sensor instrumentation, gauge model, and appropriate repair workflow.

The main accomplishment of this work was to build a solution that would enable the retrofitting of autonomic capabilities—self-configuration, self-healing, and self-management (see [22] for a formal definition)—onto legacy systems. [33] and [58] describe our implementation in greater detail; for brevity, I focus here on the event processing and correlation modules known, respectively, as the *Event Packager* and *Event Distiller*, which formed the main components of the XUES package.

Event Packager (EP). EP was developed to fulfill a need to support incoming event format transformation; common services, including timestamping; and flexible output mechanisms, including replication onto other publish/subscribe architectures or to databases for offline analysis. To accommodate both current and future formats, EP was built using pluggable modules for types, inputs, outputs, transforms, and stores, and supported the definition of multiple routing pipelines, so that incoming events would be appropriately processed and acted upon.

Written in about 9,000 lines of Java code, EP was designed to interface with several publish/subscribe infrastructures, including Siena and Elvin. Java's runtime model was leveraged to support on-the-fly module insertion and removal; an XML configuration file would specify initial configuration. Processing was primarily limited to single events or limited sequences of events; for more complex temporally-bound sequences, the Event Distiller was used.

Event Distiller (ED). ED was responsible for detecting problematic or anomalous system activities by matching (*gauging*) sequences of events (a nondeterministic ordering of events ultimately indicating correct vs. incorrect behavior) emitted by one or more sensors. Such event sequences' transitions (i.e., between subsequent events) would almost always have temporal constraints, due to the near real-time nature of the application domain and to act as a check on the state overhead needed to support nondeterministic matching.

Each ED rule was partitioned into "states" and "actions", where matches amongst the former were mapped to (meta-)events that were emitted corresponding to the latter. This state/transition representation closely corresponds to a nondeterministic finite automaton; the idea is that one event may lead to many different possible subsidiary events and match multiple rules. Other features included temporal transitions and an associated garbage collector to manage the state space; rule chaining; flexible looping; success/failure actions (optionally, at each state); absorption rules to prevent events from propagating to multiple rulesets; and variable binding to support conditional matching. Internally, ED used a collection of nondeterministic state engines for temporal complex event pattern matching.

Population of event patterns was accomplished via: a) a static, file-based rulebase; b) dynamic rule generation via an event-driven API; and c) support as a "reporting gauge" on CMU's Acme Gauge Bus [1], enabling integration with architecturally-oriented modeling tools. The Event Distiller implementation was about 7,000 lines of Java code; in experiments, a typical rulebase was a few hundred lines of XML.

Results. KX has been applied in several software environments. The two most significant ones included a DARPA integration experiment with a GIS-enabled intelligence gathering platform known as GeoWorlds [13], where KX was deployed to intelligently monitor distributed service status and restart services in case of failures or timeouts due to bugs; and as a QoS monitor/service deployer for a commercial J2EE-based multi-channel Instant Messaging (IM) service used daily by thousands of real-world end-users, where KX would monitor existing servers' loads and deploy new servers and/or migrate users in case of overload conditions. For further details, see [58] and [71].

Relation of XUES to the proposed model. From an implementation standpoint, XUES did *not* explicitly support privacy-preserving transforms. However, EP's architecture, with pluggable inputs, outputs, and transforms, and most significantly a pluggable event typing mechanism, supports most features of the stated model and lends itself extremely well to integrating privacy mechanisms. Use of publish/subscribe as a fundamental communications model enabled interest-based collaboration and eliminated dependence on source identity in event distribution. In fact, XUES served as a template for the Worminator implementation, which is discussed in the next section.

From a theoretical standpoint, KX/XUES is *privacy-compatible*, as applications can be developed on top of the platform that require application trace corroboration and decision-making based on consensus. In fact, the future work (section 8.1) references the notion of *application communities*, which requires a software monitoring platform much like KX to secure large-scale homogeneous software deployments; privacy preservation is likely to play a significant role.

5.2 Worminator

The Worminator project is the first full implementation of the proposed model for the purposes of Collaborative Intrusion Detection. A ground-up rewrite of the XUES platform, Worminator adds support for privacy-enabled data structures and transforms to take intrusion alerts generated by underlying sensors and embed them in Bloom filters or other hash-based structures. Like EP, Worminator is fully modular and supports heterogeneous data types, sensors and communication networks, and supports near real-time event processing and correlation/corroboration.

It is hypothesized that worm and other exploit attacks are often preceded by difficult-to-detect *stealthy scans* to determine hosts vulnerable to specific exploit vectors. Such stealthy scans are generally accomplished by using many probes, and distributing them across the Internet so that any given target is hit by a particular source at very low rates. Additionally, such scans can often target critical infrastructure elements with little chance of detection, because individual COTS IDS sensors—both misuse and anomaly detectors—either miss broad, slow scans (as they require too much state), or if their sensitivity is turned up, produce far too many alerts, making it infeasible for a system administrator or an automated response mechanism to respond in an appropriate fashion. Such scans can be used to build hitlists used in rapid "Warhol" worms, slow, surreptitious worms [66], and targeted attacks.

With sufficient scan data, it should be possible to create a *profile* of scanner behavior to help differentiate benign-and-curious scanners, worm-infested machines, and actively malicious entities planning their next attack. However, potential contributors, both academic and commercial, are unwilling to release alerts, for fear of revealing their network topology, open services, or even the distribution of source addresses that regularly communicate with them. This leads to the main hypothesis of the Worminator project.

Hypothesis. A privacy-preserving architecture enables the participation of a broad group of contributors to detect the slow, stealthy scans that act as precursors of attacks. The anonymous-but-categorizable identification model is used in order to glean data on suspect sources, while distinguishing between scan behavior seen at critical versus non-critical infrastructures.

Anonymity. We are in communication with various ISAC (Information Sharing Alert Coordination) organizations, set up to enable exchange of Internet threats, and are in the process of deployment to the core institution involved with the REN-ISAC (Research and Education Network ISAC [61]). An ISAC forms a natural authenticating and anonymization authority, and we hope to leverage the REN-ISAC's participation in encouraging more participants to join the Worminator effort.[6]

Implementation and alert exchange. Worminator is written in about 10,000 lines of Java code, and currently uses JBoss's [31] implementation of the Java Message Service (JMS) [67] publish/subscribe infrastructure to communicate events.[7] The main underlying sensor used is Counterstorm's AntiWorm-1, a COTS IDS sensor that is better at long-term scan detection than most others on the market [62]. Worminator reads alerts from the underlying Counterstorm sensor and exchanges them with peers, using sub-minute latencies, to rapidly determine sources of interest, targeted services, and potentially exploit vectors.

For alert exchange, *watchlists* consisting of encoded Bloom filters can be created and exchanged between peers' sites to corroborate local data.[8] Currently, (source IP, destination port) tuples are hashed and exchanged. A *watchlist corroborator* runs as a Worminator module on each site, and can perform decentralized privacy-preserving corroboration using timestamp Bloom filters as discussed in 3.3.3. Corroborated alerts can then be reported via a website, as feedback back into the Counterstorm system as additional metadata to determine threat levels, or as a *warnlist* published back to the community; the latter may optionally be published *without* privacy-preserving mechanisms to enable peers that have not corroborated to take proactive measures against a correlated threat. Watchlists and warnlists are communicated using the IETF IDMEF draft standard (Intrusion Detection Message Exchange Format [16]). The configuration to determine alert communication and workflow is written in an XML rule file and is site-specific.

Current deployment. Worminator has been successfully deployed at three academic and two commercial sites, and is currently undergoing deployment at two more academic sites. We are in discussions with other organizations to place Worminator sensors; however, the results of this thesis are not dependent on additional deployment, as regulatory compliance and legal issues make deployment an extremely slow, unpredictable process.

Early results. Worminator is very effective at corroborating sources at multiple sites, processing thousands of alerts per hour (and ready to scale up as the number of deployments increase). Figure 3

[6] Prof. Stolfo has also been instrumental in contacting other ISACs, including the Financial Services ISAC and the Multi-State ISAC, with whom discussions are ongoing.

[7] We have done some work in evaluating P2P protocols and building *network scheduling* mechanisms for decentralized exchange; see [44]. Work on decentralized exchange is ongoing, but is not directly a part of this thesis work, and is not addressed further in this proposal.

[8] As previously implied, raw alert data can be collected if participants are willing.

shows the number of sources seen at 1–4 sites. Of the roughly 32,000 sources seen in a one-month period, only 1,924 of the sources had scanned at least two sites, 659 had scanned at least 3, and 232 had scanned 4. This does not directly mean that the roughly other 30,000 sources are invalid, but if we are interested in confining our view to broad scans and probes, the amount of work needed to analyze and develop appropriate response strategies decreases by several orders of magnitude. Figure 4 demonstrates that there is a huge difference in scan lengths of sources seen at multiple sites; while some sources do "one-shot" scans, a significant number are actually repeatedly seen over the course of a full month or longer.

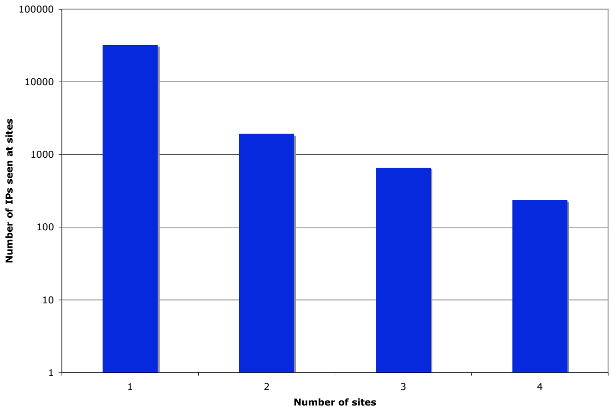

Figure 3: Logarithmic graph of the number of source IPs seen at 1–4 sites for September 12–October 10.

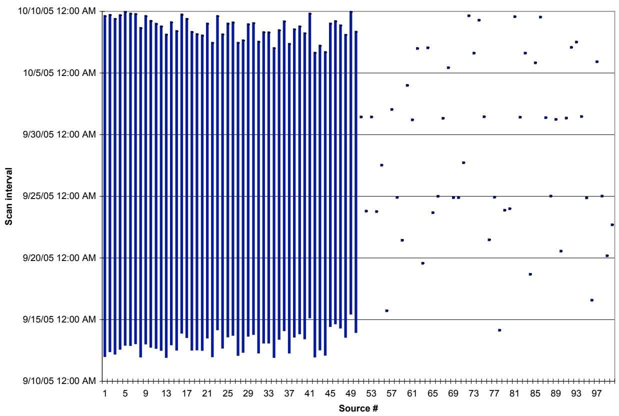

Figure 4: Average top 50 and bottom 50 scan lengths of sources scanning at least two sites for September 12–October 10.

We are continuing to collect and compile results, which will form the core of the longitudinal study proposed in section 6.1.

6 Research Plan and Schedule

Most of the implementation in 5 is complete; the system is deployed and is actively collecting data. The main remaining tasks are to formally evaluate the effectiveness of our techniques—both the main hypothesis (i.e., an evaluation of the effectiveness of our privacy-preserving mechanisms), and the "Worminator hypothesis", namely that such collaboration enables forms of scan detection that are otherwise difficult or infeasible. I discuss the planned evaluation method and expected results for each of these here, followed by a timeline scheduling the tasks to be done.

6.1 Evaluation of Worminator Collaboration: Longitudinal Study

The longitudinal study of stealthy scan behavior is designed to demonstrate the proposed Worminator hypothesis, that collaborative intrusion detection not only enables detection of *worm spread* but also scanning behavior as precursors to an attack. There are three key *longitudes* for analysis:

1. **Over time**: as it is difficult to determine ahead of time when a widespread worm attack will occur, the goal here is not necessarily to correlate certain scan behavior with particular attack

behavior[9], but to identify long-term scanners who try to "fly under the radar" by throttling their scanning rates at any individual site to as little as a few scans per day per IP. One can define a measure of *stealthiness* by looking at scan time windows, both on a local (single-site) and on a global (many-site) basis.

2. **Over geographical and network space**: a clever attacker is unlikely to focus all their scanning efforts from a single source; instead, the current trend is to spread scan efforts over a broad range of sources, and to leverage those sources as a proxy to mask scanning behavior. Botnets ([48], [60]) are also becoming an increasingly common tool for scans. By leveraging collaboration, the goal is to observe a wider destination space to ease detection of broader networks of coordinated scanners.

3. **By target**: one key form of anonymity cited in this proposal is that of *anonymous but categorizable*. This allows for the exploration of *targeted scans*, e.g., sources that scan particular categories of networks but skip others. If time permits and enough sites are deployed on Worminator (i.e., this is outside of the proposed thesis scope, but will be explored if time permits), we can compute aggregate statistics on the popularity of commercial vs. academic targets, etc. As of the writing of this thesis, only two commercial sites have been deployed, so the results of this experiment may vary.

As of the writing of this proposal, a preliminary study has been completed and presented; we aim to publish the more salient results early next year.

6.2 Evaluation of Privacy-Preserving Methods

As mentioned in sections 3.3.2 and 3.3.3, there are a number of optimizable aspects of Bloom filters.

- **Bloom filter size** and the number of hash functions used is typically dictated by the application and the amount and distribution of data inserted into it. At this time, the values are estimated to demonstrate feasibility, but ideally the size should be chosen (either empirically or automatically) so that the false positive rate is reasonably low while sparseness is minimized.
- **Practical "brute-forceability"**. This proposal asserts that the techniques in 3.3.2 result in a data structure that is infeasible to brute-force. However, this *infeasibility* must be more precisely characterized: what percentage of addresses obtained by brute-forcing are real, and what is the resulting information gain? Ideally, this value should be low enough that any honest-but-curious *or* malicious entity cannot make (in)appropriate use of the data.

Additionally, I aim to develop and evaluate metrics for corroboration effectiveness when using privacy-preserving data structures, such as Bloom filters:

- What is the **practical information loss** from using Bloom filters for Worminator? This is closely tied to the discussion on Bloom filter size, but should be compared against collected original, unobfuscated alerts.
- What **temporal constraint latencies** do Bloom filters induce? As discussed, Bloom filters: a) act as containers, and as a result induce *aggregation overhead*; and b) typically do not transmit

[9] If a widespread worm attack *does* indeed happen during our analysis window, then we will aim to correlate the attack with prior scanning behavior.

detailed timestamps of all events contained in a single BF. As a result, this closely ties in with the problem of aggregation policy. This thesis does not aim to generally solve aggregation policy, as it is highly application-specific (e.g., a real-time alert coordination system may require far more frequent communication, and less aggregation, than a long-running distributed application debugging task). However, an accurate characterization of the ability to accurately temporally correlate data with Bloom filter aggregation and communication is desirable.

- **Effectiveness of MRU and timestamp Bloom filters**: what is the performance benefit, and what is the added memory overhead? While the code in section 5.2 has an implementation of timestamp Bloom filters, it has not yet been extensively measured.

Many of these are data-driven modeling problems; I aim to build several metrics and theoretical estimates for each of these questions, and verify, if possible, with the actual Worminator deployment platform. The collection of raw data from several sites to date enables straightforward comparison testing, as the compilation of raw data into Bloom filters can be done after-the-fact as necessary.

To date, work on privacy-preservation has primarily focused on the ability to build data structures and techniques; the intent of this evaluation is to begin discussion of the effectiveness and practical limitations of such techniques.

6.3 Plan and Schedule for Completion of the Research

Table 1 shows my plan for the completion of this research.

7 Expected Contributions

7.1 Thesis Contributions

The contributions of this thesis are anticipated to include the following.

1. A **generalized approach** to supporting privacy-preserving event corroboration. As discussed in section 4, most work in this field has been extremely specialized. By building a more general framework, we expect to enable the application of privacy-preserving mechanisms and collaboration to a broader variety of fields. The techniques developed in this work are geared towards such a general application.

2. Additionally, this approach has been applied towards the development of a practical **collaborative intrusion detection** system. The methods in this thesis have been specifically geared for real-world solutions that establish privacy requirements, and Worminator's current and ongoing deployment is a key example of the feasibility of the proposed approach.

3. Characterization of the **effectiveness of privacy-preserving mechanisms**: to date, people have focused on proving the feasibility of privacy-preserving computation, but less so the advantages/disadvantages of picking such mechanisms. The proposed evaluation of the models in this thesis is a first significant step towards developing and using metrics to determine appropriate solutions based on the required application.

Table 1: Research plan and schedule for completion.
(✓ indicates a complete task, ◗ indicates an ongoing task, and ❏ indicates an upcoming task.)

Status	Task	Completion Date
✓	**XUES**: Development and Demonstration	*Completed*
✓	Development of Event Packager	September, 2002
✓	Development of Event Distiller	April, 2002
✓	Integration of XUES with other KX components	May, 2002
✓	Integration of KX with CMU ACME, GeoWorlds for DARPA demo	June, 2002
✓	Successful demonstration of KX	July, 2002
✓	Public release of XUES	September, 2002
✓	Completion, final code submission to DARPA/AFRL	March, 2004
◗	**Worminator**: Development and Deployment	*Ongoing*
✓	Prototype development and demonstration of Bloom filter-based Worminator platform at 2 sites	September, 2003
✓	Compilation of preliminary results, work on D-NAD report	April, 2004
✓	Worminator patent application	July, 2004
✓	Worminator grant proposal writing	September, 2004
✓	Development of real-time Worminator alert-sharing platform, based on earlier work on XUES	January, 2005
✓	Development of Worminator website, frontend	February, 2005
✓	Successful presentation of 3-site deployment, demo to DHS	March, 2005
✓	Presentation of 5-site deployment, demo to ARO	November, 2005
◗	Writeup of longitudinal study of scans over space and time	January, 2006
◗	**Thesis milestones**	*Ongoing*
✓	Write thesis proposal	Nov. 7, 2005
❏	Defend thesis proposal	Nov. 21, 2005
❏	Evaluation of privacy-preserving mechanisms	March, 2006
❏	Thesis distribution	July, 2006
❏	Thesis defense	August, 2006

4. Development of **fast correlation data structures for Bloom filters**, including the MRU and timestamp Bloom filters.
5. Finally, the Worminator results compiled into a **longitudinal study** will serve as a useful baseline for evaluating collaborative intrusion detection (and, hopefully, a helpful step forward for intrusion detection in general).

7.2 Research Accomplishments

In addition to the contributions listed above, the following practical accomplishments have already been completed to date:

- Published papers, including [58], [44], [27], [36], [34], [33], [26], and a poster presentation [57].

- Grant support by several DARPA, NSA, DHS, and ARO. The author was involved in the writing of two phases of DHS grants, as well as the ARO grant. Finally, successful demonstrations have been conducted for DARPA, DHS and ARO as partial fulfillment of grants' requirements.
- Successful application of the XUES platform to a DARPA challenge platform, demonstrated during Demo Days in June, 2002.
- Successful deployment of Worminator to 5+ sites [56], demonstration of Worminator to program managers at NSF, DHS, ARO and others.
- As part of the DHS contract, licensing and commercialization of the Worminator platform by Counterstorm, Inc. [30].
- Patent application filed on aspects of Worminator work.

8 Future Work and Conclusion

8.1 Future Work

As the focus of this thesis has been a generalized framework for privacy-preserving event corroboration, there are numerous interesting future work possibilities. I segment this list into a list of "if time permits" future work, which I would like to explore, and "future directions", which are long-term future explorations.

8.1.1 Immediate Future Applications

- Support for **other correlated data types** in Worminator. We have begun exploring the idea of integrating PAYL packet content alerts [74] and distributing/correlating them using Worminator. If time permits, we will explore an integration and report the results obtained by doing so.
- Develop **aggregation/exchange policies**. Currently, Bloom filters are exchanged when they reach a particular "fill" threshold or when a certain amount of time has elapsed since the previous filter was sent. To minimize communication overhead *and* improve latency, one of several optimization techniques could be adopted, e.g., a *posture*-like approach to communication: if nodes are deemed as being under threat, they may choose to communicate more frequently (e.g., 1 second latency) than when nodes are idle (once or twice a minute).
- Development of a **privacy-preserving policy language** into Worminator. The language would most likely be designed in XML, as an extension of the W3C P3P standard. Each privacy-preserving data structure would then have associated metadata, enabling matching between policies and implementations.
- **Building an "Application Communities"** [15] peer-to-peer application monitoring framework. The fundamental idea behind Application Communities is to accept software monoculture and to *leverage* it. Early work [45] has demonstrated the possibility of distributing the task of detecting faults in instrumented code, which typically runs orders-of-magnitude slower than native code, thereby reducing the effect on any one individual instance. Worminator could be adapted to serve as a privacy-preserving infrastructure; in fact, several of the committee

members have submitted a grant proposal to DARPA to, amongst other things, explore this possibility.

8.1.2 Future Directions

- Solving **other privacy challenges**; the most difficult one is probably the malicious insider problem, where a participant can pierce the veil of anonymous-but-differentiable by triggering events at peers' sites in a coordinated fashion. For example, a malicious insider could execute portscans from a known source to a specific peer known to be participating in a collaboration network, and can then wait for corroborating evidence to come from that peer. While the event would be tagged with a unique identifier, the insider could scan the event, and if the known source is found, draw an association. No truly general solution may be found other than eliminating differentiability, but various workarounds may be possible.

- Evaluation of current **event distribution strategies** and integration of other distribution strategies underlying publish/subscribe infrastructures. We have done some exploratory work in this field [44], and work is ongoing on developing a complete decentralized-yet-secure exchange model.

- Development of a solution for **automatically profiling** scanners and attackers. Preliminary work done in our longitudinal study demonstrates there are unique scanner profiles—the worm-infested home computer connected to a cable modem, the targeted attacker that has gained access to full class C (/24) subnets to distribute his behavior, the very long-term stealthy scanner, etc. A logical extension of this platform, to improve autonomic response mechanisms, would be to cluster this data and assign threat levels based on the resulting profile.

- A more **general event typing and versioning** framework: our work in [58] discusses mechanisms to support more flexible type discovery and versioning via the use of *tag processors*; such work could also apply for privacy-preserving types and corresponding transforms.

- Popularize the term **collaroborelate**, a combination of collaborate, correlate and corroborate, to simplify the vocabulary choice of any work extending this thesis. Or, invent a better acronym first. :)

8.2 Conclusion

In this proposal, I introduced the notion of *privacy-preserving decentralized event correlation*, with the primary goal of enabling collaboration (and, in particular, corroboration) between peers whose privacy policies typically prevent data disclosure. A number of architectural decisions, techniques, and two implementations were outlined in order to accomplish this goal. Preliminary results and plans for future results were also presented. Ultimately, I hope the work done for my thesis will improve and extend the domain of collaborating applications and peers, and spur the development of new languages for expressing such policies.

References

[1] CMU ABLE. DASADA Gauge Infrastructure, 2002. http://www.cs.cmu.edu/~able/rainbow/gaugeinf.html.

[2] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information Sharing Across Private Databases. In *SIGMOD*, 2003.

[3] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic Databases. In *VLDB*, 2002.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-Preserving Data Mining. In *ACM SIGMOD*, 2000.

[5] Stanford Program Analysis and Verification Group. Rapide website, 1999. http://pavg.stanford.edu.

[6] Guruduth Banavar, Marc Kaplan, Kelly Shaw, Robert E. Strom, Daniel C. Sturman, and Wei Tao. Information Flow Based Event Distribution Middleware. In *Middleware Workshop at the International Conference on Distributed Computing Systems*, 1999.

[7] Mayank Bawa, Roberto J. Bayardo Jr., and Rakesh Agrawal. Privacy-Preserving Indexing of Documents on the Network. In *VLDB*, 2003.

[8] Steven M. Bellovin and William R. Cheswick. Privacy-Enhanced Searches Using Encrypted Bloom Filters, 2004.

[9] Burton H. Bloom. Space/time trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[10] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service. In *Principles of Distributed Computing*, 2000.

[11] Francis Chang, Wu-chang Feng, and Kang Li. Approximate Caches for Packet Classification. In *IEEE INFOCOM*, 2004.

[12] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *International Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[13] Murilo Coutinho, Robert Neches, Alejandro Bugacov, Ke-Thia Yao, Vished Kumar, In-Young Ko, and Ragy Eleish. GeoWorlds: A Geographically Based Information System for Situation Understanding and Management. In *International Workshop on TeleGeoProcessing (TeleGeo 99)*, Lyon, France, 1999.

[14] R. Danyliw, J. Meijer, and Y. Demchenko. The Incident Object Description Exchange Format Data Model and XML Implementation, 2005. http://tools.ietf.org/wg/inch/draft-ietf-inch-iodef/draft-ietf-inch-iodef-04.txt.

[15] DARPA. Application Communities Proposer Information Pamphlet/BAA, 2005. http://www.darpa.mil/ipto/Solicitations/open/05-51_PIP.htm.

[16] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format, 2005. http://tools.ietf.org/wg/idwg/draft-ietf-idwg-idmef-xml/draft-ietf-idwg-idmef-xml-14.txt.

[17] John R. Douceur. The Sybil Attack. In *International Workshop on Peer-to-Peer Systems*, 2002.

[18] Wenliang Du and Mikhail Atallah. Secure Multi-Party Computation Problems and Their Applications: A Review and Open Problems. In *New Security Paradigms Workshop*, 2001.

[19] Alexander Dupuy, Soumitra Sengupta, Ouri Wolfson, and Yechiam Yemini. NETMATE: A Network Management Environment. *IEEE Network*, 5(2):35–43, 1991.

[20] EPIC. Privacy, 2005. http://www.epic.org/privacy/.

[21] Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In *ACM SIGCOMM*, 1998.

[22] A.G. Ganek and T.A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 2003.

[23] David Garlan, Robert T. Monroe, and David Wile. Acme: Architectural Description of Component-Based Systems. In *Foundations of Component-Based Systems*, pages 47–67. Cambridge University Press, 2000.

[24] Eu-Jin Goh. Secure Indexes. Technical report, 2004.

[25] David Goldschlag, Michael Reed, and Paul Syverson. Onion Routing for Anonymous and Private Internet Connections. *Communications of the ACM*, 43(2):39–41, 1999.

[26] Philip Gross, Suhit Gupta, Gail Kaiser, Gaurav S. Kc, and Janak J. Parekh. An Active Events Model for Systems Monitoring. In *Working Conference on Complex and Dynamic Systems Architectures*, 2001.

[27] Philip Gross, Janak J. Parekh, and Gail Kaiser. Secure ”Selecticast” for Collaborative Intrusion Detection Systems. In *Workshop on Distributed Event-Based Systems*, 2004.

[28] The Open Group. Universal Unique Identifier, 1997. `http://www.opengroup.org/onlinepubs/9629399/apdxa.htm`.

[29] Qiang Huang, Helen J. Wang, and Nikita Borisov. Privacy-Preserving Friends Troubleshooting Network. In *NDSS*, San Diego, CA, 2005.

[30] Counterstorm Inc. Counterstorm homepage. `http://www.counterstorm.com`.

[31] JBoss Inc. JBoss Application Server, 2005. `http://www.jboss.com/products/jbossas`.

[32] Ramaprabhu Janakiraman, Marcel Waldvogel, and Qi Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *WETICE*, 2003.

[33] Gail Kaiser, Philip Gross, Gaurav S. Kc, Janak J. Parekh, and Giuseppe Valetto. An Approach to Autonomizing Legacy Systems. In *Workshop on Self-Healing, Adaptive and Self-MANaged Systems*, New York, NY, 2002.

[34] Gail Kaiser, Janak J. Parekh, Philip Gross, and Giuseppe Valetto. Kinesthetics eXtreme: An External Infrastructure for Monitoring Distributed Legacy Systems. In *Autonomic Computing Workshop*, 2003.

[35] Russell Kay. Event Correlation. *ComputerWorld*, 2003. `http://www.computerworld.com/networkingtopics/networking/management/story/0,10801,83396,00.html`.

[36] Angelos D. Keromytis, Janak J. Parekh, Philip Gross, Gail Kaiser, Vishal Misra, Jason Nieh, Dan Rubenstein, and Salvatore J. Stolfo. A Holistic Approach to Service Survivability. In *ACM Workshop on Survivable and Self-Regenerative Systems*, 2003.

[37] Lea Kissner. *Thesis Proposal: Privacy Preserving Distributed Information Sharing*. PhD thesis, CMU, 2005.

[38] Lea Kissner and Dawn Song. Privacy-Preserving Set Operations. In *CRYPTO*, 2005.

[39] H. Krawczyk, M. Bellare, and R. Canetti. RFC 2104 - HMAC: Keyed-Hashing for Message Authentication, 1997. `http://www.faqs.org/rfcs/rfc2104.html`.

[40] Christopher Kruegel, Thomas Toth, and Clemens Kerer. Decentralized Event Correlation for Intrusion Detection. In *International Conference on Information Security and Cryptology*, 2002.

[41] Leslie Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

[42] Patrick Lincoln, Phillip Porras, and Vitaly Shmatikov. Privacy-Preserving Sharing and Correlation of Security Alerts. In *USENIX Security*, 2004.

[43] Yehuda Lindell and Benny Pinkas. Privacy Preserving Data Mining. *Cryptology*, 15(3):177–206, 2002.

[44] Michael E. Locasto, Janak J. Parekh, Angelos D. Keromytis, and Salvatore J. Stolfo. Towards Collaborative Security and P2P Intrusion Detection. In *IAW*, West Point, 2005.

[45] Michael E. Locasto, Stelios Sidiroglou, and Angelos D. Keromytis. Application Communities: Using Monoculture for Dependability. In *HotDep*, 2005.

[46] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley/Pearson Education, Indianapolis, 1st edition, 2002.

[47] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann. Specification and Analysis of System Architecture Using Rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355, 1995.

[48] Bill McCarty. Botnets: Big and Bigger. *IEEE Security and Privacy*, 1(4):87–90, 2003.

[49] Michael Mitzenmacher. Compressed Bloom Filters. *IEEE Transactions on Networking*, 10(5):604–612, 2002.

[50] Donald H. Morrison. PATRICIA–Practical Algorithm To Retrieve Information Coded in Alphanumeric. *Journal of the ACM*, 15(4):514–534, 1968.

[51] NIST. FIPS 180-1: Secure Hash Standard (SHA-1), 1995. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.

[52] US Department of Health and Human Services. HIPAA, 2005. `http://www.hhs.gov/ocr/hipaa/`.

[53] US Department of Labor/OSHA. SIC Division Structure, 2005. `http://www.osha.gov/pls/imis/sic_manual.html`.

[54] Ruoming Pang and Vern Paxson. A High-Level Programming Environment for Packet Trace Anonymization and Transformation. In *ACM SIGCOMM*, 2003.

[55] Janak J. Parekh. Candidacy Exam, 2003.

[56] Janak J. Parekh. Worminator homepage, 2005. `http://worminator.cs.columbia.edu`.

[57] Janak J. Parekh. Worminator (poster). In *Symposium on Recent Advances in Intrusion Detection*, 2005.

[58] Janak J. Parekh, Gail Kaiser, Philip Gross, and Giuseppe Valetto. Retrofitting Autonomic Capabilities onto Legacy Systems. *Cluster Computing*, To appear in, 2005.

[59] Phillip Porras and Peter G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *National Information Systems Security Conference*, 1997.

[60] Honeynet Project and Research Alliance. Know your Enemy: Tracking Botnets, 3/13/05 2005. `http://www.honeynet.org/papers/bots/`.

[61] Research, Education Networking Information Sharing, and Analysis Center. REN-ISAC homepage, 2005. `http://www.ren-isac.net/`.

[62] Seth Robertson, Eric V. Siegel, Matt Miller, and Salvatore J. Stolfo. Surveillance Detection in High Bandwidth Environments. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, 2003.

[63] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *COST264 Workshop on Networked Group Communication*, 2001.

[64] Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content-Based Routing with Elvin4. In *AUUG2K*, 2000.

[65] Stuart Staniford-Chen, Steven Cheung, R. Crawford, and M. Dilger. GrIDS - A Graph Based Intrusion Detection System for Large Networks. In *National Information Computer Security Conference*, Baltimore, MD, 1996.

[66] Stuart Staniford-Chen, Vern Paxson, and Nicholas Weaver. How to 0wn the Internet in Your Spare Time. In *USENIX Security*, 2002.

[67] Sun. Java Message Specification, 2002. `http://java.sun.com/products/jms/`.

[68] Latanya Sweene. k-Anonymity: A Model for Protecting Privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[69] Johannes Ullrich. DShield home page, 2005. `http://www.dshield.org`.

[70] Risto Vaarandi. SEC - Simple Event Correlator, 2005. `http://simple-evcorr.sourceforge.net/`.

[71] Giuseppe Valetto and Gail Kaiser. Using Process Technology to Control and Coordinate Software Adaptation. In *International Conference on Software Engineering*, 2003.

[72] W3C. Platform for Privacy Preferences (P3P) Project, 2005. `http://www.w3.org/P3P/`.

[73] Helen J. Wang, John C. Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang. Automatic Misconfiguration Troubleshooting with PeerPressure. In *OSDI*, San Francisco, 2004.

[74] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *RAID*, Seattle, WA, 2005.

[75] Ke Wang and Salvatore J. Stolfo. Anomalous Payload-based Network Intrusion Detection, 2004.

[76] Andrew C. Yao. Protocols for Secure Computations. In *IEEE Symposium on Foundations of Computer Science*, 1982.

[77] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global Intrusion Detection in the DOMINO Overlay System. In *NDSS*, 2004.

[78] Shaula Alexander Yemini, Shmuel Kliger, Eyal Mozes, Yechiam Yemini, and David Ohsie. High Speed and Robust Event Correlation. *IEEE Communications*, 1996.

[79] Yuanyuan Zhao and Robert E. Strom. Exploiting Event Stream Interpretation in Publish-Subscribe Systems. In *Principles of Distributed Computing*, 2001.

Acknowledgements

In addition to acknowledging my committee (Gail Kaiser, Sal Stolfo and Al Aho) for their timely and helpful feedback, I would like to acknowledge my colleagues in this work, including Gabriela Cretu and Ke Wang for their help on Worminator; Phil Gross, Gaurav Kc, Peppo Valetto and Enrico Buonnano for their work on KX; and Suhit Gupta for help in brainstorming ideas for this proposal. I also thank other members of PSL and IDS, and my friend Katherine Compton for her tireless encouragement.