Privacy-Preserving Collaborative Anomaly Detection

Haakon Andreas Ringberg

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Jennifer Rexford

September 2009

Abstract

Unwanted traffic is a major concern in the Internet today. Unwanted traffic includes Denial of Service attacks, worms, and spam. Identifying and mitigating unwanted traffic costs businesses many billions of USD every year. The process of identifying this traffic is called anomaly detection, and Intrusion Detection Systems (IDS'es) are among the most prevalent techniques. IDS'es, such as Snort, allow users to write "rules" that specify the properties of traffic that should be detected and the corrective action to be taken in response. Unfortunately, applying these rules in an online setting can be prohibitively expensive for large networks, such as Tier-1 ISPs, which may have tens of thousands of links and many Gbps of traffic. In the first chapter of this thesis we present a system that leverages machine learning algorithms to detect the same type of unwanted traffic as Snort, but on summarized data for faster processing. Our results demonstrate that this system can effectively learn to classify many Snort rules with a high degree of accuracy.

Unfortunately, distinguishing good traffic from unwanted traffic is challenging even in an offline setting because many types of unwanted traffic traffic, such as network attacks, deliberately mimic the behavior of normal traffic. We therefore propose that the targets of unwanted traffic should collaborate by correlating their attack data, under the assumption that a given malicious host is likely to affect more than one victim over time. That is, the senders of unwanted traffic will use individual computers (*i.e.*, malicious hosts) repeatedly for various nefarious purposes in order to maximize their profits, and this repeated use will leave traces across networks. In the second chapter of this thesis we present a measurement study that quantifies the potential gain from this collaborative anomaly detection. Specifically, using traces from operational networks, we calculate the fraction of detected network anomalies

(*viz.*, IP scans, port scans, and DoS attacks) that could have been mitigated if some subset of the victims collaborated by sharing information about past perpetrators.

One major challenge with the proposed collaborative anomaly detection is that the human owner/operators of participating networks are often hesitant to openly share information about the hosts (customers) that use their services. In the third chapter of the thesis we address this problem by proposing and evaluating the efficiency of a novel cryptographic protocol that allows victims to collaborate in a manner that protects their privacy. Our protocol allows participants to submit a set of IP addresses that they suspect might be engaging in unwanted activity, and it returns the set of IP addresses that existed in some fraction of all suspect sets (*i.e.*, threshold set-intersection). The protocol preserves privacy because it never reveals who suspected whom, and a submitted IP address is only revealed when more than n participating networks suspect it. Our implementation of said protocol is able to correlate millions of suspect IP addresses per hour when running on two quad-core machines.

Acknowledgements

I would not be where I am—or even who I am—without the help of a great many people. I will thank a few of them.

Professional Benny Applebaum, for explaining cryptography to me; Matthew Caesar, for excellent writing; Robert Calderbank, for serving on my pre-FPO committee; Mark Crovella & Anukool Lakhina, for generosity in time, energy, and software, without which I would not have gotten off the ground; Cabernet Group Members, for a motivating and collegial research group; Christophe Diot, for providing the preciseness and push I needed, and for a great deal of subsequent support; Nicholas Duffield, for being a great mentor at AT&T Labs and for serving as a thesis reader; William Edgar, for inspiration and opening doors; Michael Freedman, for collaborating on an interesting project and for serving as a thesis reader; Patrick Haffner, for being a very helpful collaborator; Brian Kernighan, for demonstrating that brilliant people can also be wonderful people, and for serving on my thesis committee; Balachander Krishnamurthy, for clear unfiltered insights; Craig Labovitz, for a rewarding summer in Ann Arbor; Kai Li, for being the primary reason I went to Princeton in the first place, and for consistently prioritizing what is best for me and my research; Larry Peterson, for serving on my thesis committee; Augustin Soule, for being my closest collaborator and a valued friend; Nicholas Sturgeon, for making me realize how rewarding I find philosophical discourse;

This dissertation is based on joint research with numerous brilliant collaborators. In particular: Chapter 2 is based on work with Nicholas Duffield, Patrick Haffner, and Balachander Krishnamurthy; Chapter 3 is based on work with Matthew Caesar,

Jennifer Rexford, and Augustin Soule; and Chapter 4 is based on work with Benny Applebaum, Matthew Caesar, Michael Freedman, and Jennifer Rexford.

I'd also like to acknowledge Thomson for generous financial support and the NSF for grant CNS-0626771.

Friends Brian Alexander, for watching baseball with me when we both had "important" work to do; Elaine Auyeung, for doing my algorithms homework; Tim Bavaro, for a shared appreciation of acerbic humor; Ramona Pousti Canan, for being as close to a soul mate as I can imagine; Tony Capra, for a sharp intellect; Melissa Carroll, for putting up with countless asinine arguments; Hubert Chao, for bringing me into the 312 fold; Min Li Chan, for being interesting; Jamie Consuegra, for smiling at me in English class; Amogh Dhamdhere, for being a fellow victim of a harsh job market; Ariful Gani, for making me feel grateful to be a Yankee fan; Ilya Ganusov, for being cool; Aleksey Golovinskiy, for being hilarious; Ji Gu, for shelter; Runa Hald, for being a challenge; Maya Haridasan, for having the second best smile in the world; Jiayue He, for four wonderful and delicious months in Paris and many more to come in NYC; Frode Henriksen, for being intellectually refreshing; Sigrid Holm, for never saying you didn't like FOX; Jialu Huang, for making me happy; Grunde Jomaas, for soccer; William Josephson, for more trips to NYC than I wish to enumerate; Ahreum Kang, for being perhaps the most wonderful person I know; Dhruva Karle, for being like a brother; Janek Klawe, for having the best smile ever; Kelly Li, for an abundance of spunk; Xiaojuan Ma, for being a great traveling companion; David Menestrina, for being a friend beyond reproach; Radha Narayan, for moral evolutionism; Linda Nguyen, for inspiring me to become

a better partner; Sarah Nguyen, for being a promising padawan; Kori Oliver, for making me feel cool; Frances Perry, for paving the way; Ann Raldow, for having the best memory of anyone I know; Aditya Rao, for being loyal to a fault; Sandeep Ravindran, for always being fun to hang around; Senad Rebac, for being both a coach and a friend; Michael Scullard, for being funny; Carl Christian Størmer, for being a kindred spirit; Jeffrey Vaughan, for letting me be a substitute rabbi; Nick Vlku, for intensely caffeinated conversations; Tracy Wang, for lots of shared moments; Lana Yarosh, for being fun; Harlan Yu, for fried chicken;

Family My parents, Tore Larsen and Unni Ringberg, whose impact is beyond words and numbers. The mere thought of matching your parenting efforts is overwhelming.
My brother, Helge Ringberg. I am a big fan of the Friedrich Nietzsche quote "[that which] doesn't kill us makes us stronger." I take some amount of personal pride in making you as strong as you are today. By the same token, however, I apologize for making you as strong as you are today. You're a wonderful brother!
My grandmother, Margareth Larsen, whose generosity toward those less fortunate is immensely inspirational. I hope to be equally intellectually sharp and active when I've passed 80. My grandfather, Lars Larsen, for lying on the floor under a table next to me drawing Santa Claus, ships, and cars.
My second set of parents, Bruce and Beverly Shriver. I cannot express how profoundly grateful I am that you accepted me into your house and family almost 10 years ago. Since it is absurd to think that I could ever repay you, I only hope that I can be equally generous to someone else in the future.

Jennifer Rexford A few years ago a fellow advisee asked whether I'd be willing to assist in an application to award Jen a faculty advisor award. I initially resisted because I felt many of these faculty awards are either (a) granted in a round-robin fashion, or (b) primarily a popularity contest: neither of which would do justice to how uniquely qualified and dedicated Jen is. Jen is not only a totally selfless advisor who cares only about her students, but she is also a great resource for lessons on the direction and presentation of research. It goes without saying that no one has been more important to my thesis and I could not have asked for a better advisor.

Too all: merci, thank you, & tusen takk.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet is integral for human communication: news, financial transactions, personal photographs, and video conferences, to name a few, are now frequently routed through the Internet. In spite of its ubiquitousness and importance, the Internet today is plagued by a great deal of unwanted traffic. Ironically, the same openness that has made the Internet wildly successful also makes it impossible to fully eradicate this unwanted traffic. In this dissertation we present an architecture that helps mitigate the problem of unwanted traffic. Our approach provides three essential properties: (1) scalable algorithms for the detection of unwanted traffic; (2) accuracy of detections by promoting collaboration between the victims of this traffic; and (3) protecting the privacy of collaborating networks via a novel cryptographic protocol.

1.1 Why Unwanted Traffic is Here to Stay

The Internet's precursors (NSFNet and ARPANET) were managed by a single entity and set up for communication between mostly trusted parties such as different branches of the US military or US universities. There was minimal concern that these parties would themselves inject unwanted traffic into the network since they had no motivation to do so. This "open access" policy was continued, and became further entrenched, with the NSF's efforts to commercialize the Internet in the 1990s.

Open access means that any network can join the Internet as long as it can connect to another network in the Internet. This openness allowed the Internet to grow from interconnecting thousands of computers to interconnecting billions of computers. However, this openness also means that the Internet itself is merely the cloud of interconnected networks: anyone can join and there is no central authority to manage the masses. Much like any other social club, therefore, the Internet's open membership policy has supported growth but admitted the occasional bad network apple. The bad networks, in turn, admit and tolerate bad end-hosts as their customers, which means that open access extends all the way to end-users.

Moreover, these occasional bad end-users have plenty of economic incentive to send various forms of traffic that is unwanted by destinations. Spam—unwanted pieces of email—for example, often contain unsolicited advertisements, which are profitable for those sending the spam. In order to maximize their profits, the nefarious parties try to send as much spam as possible, which requires many computers. Spam therefore indirectly leads to trolling for, hijacking, and commandeering vulnerable computers, possibly via Internet worms. Such compromised hosts are called

bots—because they are "robots" which can be controlled—and large collections of bots are referred to as botnets [16]. Bots can be used for spam [31, 68], DoS attacks [72], pump-and-dump schemes [1], click fraud [60, 51, 53], identity theft [8, 13], and a slew of other illicit actions [24, 35, 5].

There is strong reason to believe that the status quo provides the three necessary ingredients to ensure that the problem of unwanted traffic cannot be fully eradicated: means, motive, and opportunity. A large number of skilled computer programmers have the means to write software to perform the aforementioned illicit network acts; motive is in plentiful supply given that actions such as identity theft and pump-and-dump scams can be very profitable; and finally, the openness of the Internet provides numerous opportunities for malicious parties to send unwanted traffic. In this dissertation we therefore focus on algorithms that can help mitigate the problem of unwanted traffic in an efficient manner.

1.2 Rule-Based Anomaly Detection on IP Flows

Computer networks have limited or no control of the networks (or end-users) that send them traffic, and nor would it make business sense to block all traffic from a large swath of networks. Networks are therefore in need of algorithms that can detect unwanted traffic "on the fly"; we will refer to such detection algorithms as "anomaly detectors." One of the most prevalent set of anomaly detection techniques rely on signatures of unwanted traffic. That is, most unwanted traffic serves a very specific purpose for the originator, and can therefore contain telltale signs. Internet worms, for example, take advantage of vulnerabilities in computer software in order to take control of the end-hosts running this software; the IP packets spreading the

worm must therefore contain specific sequences of bytes in order to leverage the vulnerability.

Signature-based intrusion detection systems (IDS) often provide a simple language within which users can specify (A) the signature of some type of unwanted traffic that should be detected (often called a rule); and (B) the corrective action that should be taken in response (*e.g.*, alerting, off-ramping for further analysis, blocking). Their programmable and extensible nature have made such IDS'es very popular for protecting the edges of enterprise networks. Large communities of users have thus sprung up around these anomaly detectors; the members share their experiences with, and the rules that they write for, these IDS'es with one another, which make the IDS'es themselves even more appealing to new potential users.

One important drawback with the rule-based IDS'es, however, is that they do not effectively scale up to the speeds required by large networks. It may simply be infeasible to perform the deep-packet inspection (DPI) necessary to execute all types of rules, especially if the rules themselves can be written as arbitrary regular expressions. Furthermore, in order to protect the edge of large networks, all traffic traversing thousands of links will have to be compared against potentially hundreds of rules: a daunting task.

An intrusion detection system that could inspect every network packet would be ideal, but is impractical. Signature based detection systems such as Snort have been widely deployed by enterprises for network security, but are limited by the scaling factors described above. We have therefore developed an architecture that can translate many existing packet signatures to instead operate effectively on IP flows, which characterize a set of IP packets between two end-points within some fixed time interval. Flow statistics are compact and collected ubiquitously within

most ISPs' networks, often in the form of NetFlow.

We wish to construct rules at the flow level that accurately reproduce the action of packet-level rules. In other words, an alarm should ideally be raised for flows that are derived from packets that would trigger packet-level rules. Our methods are probabilistic in that the flow level rules do not reproduce packet level rules with complete accuracy; this is the price we pay to be scalable. More precisely, our architecture leverages Machine Learning (ML) algorithms in order to discover the flow-level classifier that most successfully approximates a packet signature. The essential advantage of ML algorithms is their ability to learn to characterize flows according to predicates that were not included in the original packet-level signature.

1.3 Evaluating the Potential for Collaborative Anomaly Detection

Our probabilistic methods to improve rule-based IDS'es trade off some accuracy for a great deal of scalability—but we still want that accuracy back. Moreover, even the "perfect" IDS'es are not perfect. That is, in many cases it may simply be impossible for any individual edge-network in isolation to distinguish good traffic from bad traffic. At a high level, the malicious parties that originate unwanted traffic will be most effective in their aim if they elude detection, which they can do by mimicking the behavior of normal traffic. The victim of a Distributed Denial of Service (DDoS) attack—an attack where the malicious party tries to disrupt the victim's network's services–for example, may not be able to distinguish requests from users that are genuinely interested in the service from those requests whose purpose is only to overwhelm the victim.

All is not lost, however. Recall that the malicious parties that control and originate unwanted traffic frequently have specific aims which revolve around maximizing their profit. It is therefore reasonable to expect that each individual compromised host—a.k.a. bot—will be used for multiple illicit actions over time. Therefore, if the victims (*e.g.*, networks or websites) of such attacks shared information about the hosts that were present during attacks then likely compromised hosts could be identified by being present in an unusually high number of attacks ("fool me once, shame on you, fool me twice, shame on me"). In other words, collaborating by combining their multiple vantage-points will allow the victims of unwanted traffic to become increasingly confident in their set of identified bots by the repeated appearance of certain hosts in attacks.

In order to understand the potential benefit of victim collaboration, it is important to study unwanted traffic from operational networks. Building this understanding requires studying what unwanted traffic looks like when viewed across several vantage points, and precisely how these vantage points may monitor traffic and exchange information to best isolate attacks. We accomplish this through a measurement study where we apply standard network anomaly detectors to IP flow traces from GEANT, a European ISP, to identify unwanted traffic, and analyze the ability of a representative set of collaboration schemes to assist the victims in isolating and mitigating these attacks. Our final results calculate the number and percentage of attacks that could have been mitigated by a set of collaborating victim end-hosts.

1.4 Privacy-Preserving Collaborative Anomaly Detection

The concern for privacy is a major obstacle for collaborative anomaly detection. That is, the suspicions shared between collaborating networks would necessarily include both legitimate customers and bona fide compromised malicious hosts (or otherwise the collaboration would be unnecessary). Divulging information about legitimate customers would be problematic for many potential participating networks. Furthermore, participants would probably be uneasy with any other network knowing which hosts they suspected, as this could reveal information about customer or traffic mix. Any collaboration scheme would therefore have to protect the privacy of both the submitted suspects and of the participating networks themselves.

We present a cryptographic protocol and implementation that provides two essential privacy properties: (1) suspect privacy ensures that no party should learn anything about IP addresses that are below some minimum threshold t; and (2) participant privacy ensures that no party should learn which suspect IP addresses were submitted by which collaborating network. Our system is "semi-centralized" in that the data analysis is split between two separate parties: a proxy and a database. The proxy plays the role of obliviously blinding suspect IP addresses, as well as transmitting blinded IP addresses to the database. The database, on the other hand, builds a table that is indexed by the blinded IP address. Each blinded IP address that ultimately is submitted by more than t networks is shared with the proxy, which unblinds the IP address.

The resulting semi-centralized system provides strong privacy guarantees under the assumptions that the proxy and the database do not collude. In practice, we

imagine that these two components will be managed either by the participants themselves that do not wish to see their own information leaked to others, perhaps even on a rotating basis, or third-party commercial or non-profit entities tasked with providing such functionality. For example, Google (which already plays a role in bot and malware detection [32]) or the EFF (which has funded anonymity tools such as Tor [17]), which themselves have no incentive to collude. It should be emphasized that the proxy and database are not treated as trusted parties: we only assume that they will not collude.

Using a semi-centralized architecture greatly reduces operational complexity and simplifies the liveness assumptions of the system. For example, clients can asynchronously provide their suspicions without our system requiring any complex scheduling. Despite these simplifications, the cryptographic protocols necessary to provide strong privacy guarantees are still non-trivial. Specifically, our solution makes use of oblivious pseudo-random functions [59, 25, 33], amortized oblivious transfer [58, 36], and homomorphic encryption with re-randomization. Our experiments show that the performance of our system scales linearly with computing resources, making it easy to improve performance by adding more cores or machines. For collaborative diagnosis of denial-of-service attacks, our system can handle millions of suspect IP addresses per hour when the proxy and the database each run on two quad-core machines.

1.5 Thesis Contribution

When combined, the systems and protocols presented in this dissertation provide a complete framework to help mitigate the problem of unwanted traffic: from

high-speed detection at individual networks to collaboration between networks in a privacy-preserving manner. Our architecture provides three essential properties:

1. **Scalability:** We make the classification power and extensibility of rule-based anomaly detectors available to very large networks, such as Tier-1 ISPs. We increase speed by analyzing summarized data structures (IP flows) instead of IP packets and retain a great deal of accuracy for many rules by leveraging machine learning (ML) algorithms to discover latent correlations between packet-level features and flow-level features.

2. **Accuracy:** In a measurement study we demonstrate that collaboration between victims of unwanted traffic can help improve detection accuracy because individual compromised hosts are used repeatedly for different attacks.

3. **Privacy** Finally, we present a novel privacy-preserving collaboration protocol, which will allow victims to share data to become more confident in their suspicions of potentially compromised hosts. Our protocol does this while protecting the privacy of (A) the participating networks, and (B) the submitted suspects.

Chapter 2

Rule-Based Anomaly Detection on IP Flows

2.1 Introduction

2.1.1 Motivation

Detecting unwanted traffic is a crucial task in managing data communications networks. Detecting network attack traffic, and non-attack traffic that violates network policy, are two key applications. Many types of unwanted traffic can be identified by rules that match known signatures. Rules may match on a packet's header, payload, or both. The 2003 Slammer Worm [55], which exploited a buffer overflow vulnerability in the Microsoft SQL server, was matchable to a signature comprising both packet header fields and payload patterns.

Packet inspection can be carried out directly in routers, or in ancillary devices observing network traffic, e.g., on an interface attached to the network through a passive optical splitter. Special purpose devices of this type are available from

vendors, often equipped with proprietary software and rules. Alternative software systems such as Snort [74] can run on a general purpose computer, with a language for specifying rules created by the user or borrowed from a community source.

In any of the above models, a major challenge for comprehensive deployment over a large network, such as a Tier-1 ISP, is the combination of network scale and high capacity network links. Packet inspection at the network edge involves deploying monitoring capability at a large number of network interfaces (access speeds from OC-3 to OC-48 are common) whereas monitoring in the network core is challenging since traffic is concentrated through higher speed interfaces (OC-768 links are increasingly being deployed). In either case, applying possibly many hundreds of rules to the traffic at line-rate can be infeasible. Whereas fixed-offset matching is cheap computationally and has known costs, execution of more complex queries may hit computational bandwidth constraints. Even in cases where detection is provided natively by the router, there may be large licensing costs associated with its widespread deployment.

2.1.2 Signature-based Detection on IP Flows

An intrusion detection system that could inspect every network packet would be ideal, but is impractical. Signature-based detection systems such as Snort have been widely deployed by enterprises for network security, but are limited by the scaling factors described above. We have therefore developed an architecture that can translate many existing packet signatures to instead operate effectively on IP flows. Flow statistics are compact and collected ubiquitously within most ISPs' networks, often in the form of NetFlow [14].

Our work does not supplant signature-based detection systems, but rather ex-

tends their usefulness into new environments where packet inspection is either infeasible or undesirable. We wish to construct rules at the flow level that accurately reproduce the action of packet-level rules. In other words, an alarm should ideally be raised for flows that are derived from packets that would trigger packet-level rules. Our methods are probabilistic in that the flow level rules do not reproduce packet level rules with complete accuracy; this is the price we pay to be scalable.

The idea of deriving flow-level rules from the header portion of a packet-level rule has been proposed in [49], but this technique only applies to rules that exclusively inspect a packet's header. What can be done for rules that contain predicates that match on a packet's payload? Ignoring the rule or removing the predicates are both unsatisfactory options, as they will lead to heavily degraded detection performance in general. Signatures that inspect a packet's payload can still be effectively learned if there is a strong association between features of the flow header produced by this packet and the packet's payload. For example, the Slammer Worm infects new host computers by exploiting a buffer overflow bug in Microsoft's SQL server; these attack packets contain known payload signatures in addition to targeting a specific UDP port on the victim host. The Snort signature to detect these packets utilizes both these pieces of information to improve detection. An exhaustive system for translating packet rules into flow rules must leverage these correlations between the packet payload and flow header in order to mitigate the impact of losing payload information.

Some signatures exhibit a strong association between payload and flow-header information even though no correlation is implied in the original packet signature. This can occur either because the human author of the signature was unaware of or disregarded this piece of information (*e.g.*, the unwanted traffic very frequently uses

a particular destination port, even though this was not specified in the packet signature), or because the association exists between the payload and flow-header features that have no packet-header counterpart (*e.g.*, flow duration). For this reason, our architecture leverages Machine Learning (ML) algorithms in order to discover the flow-level classifier that most successfully approximates a packet signature. The essential advantage of ML algorithms is their ability to learn to characterize flows according to predicates that were not included in the original packet-level signature.

2.1.3 Contribution

The primary contribution of this chapter is the ML-based architecture that can detect unwanted traffic using flow signatures. These flow signatures are learned from a reference set of packet signatures and joint packet/flow data. We evaluate our system on traces from and signatures used by a medium-sized enterprise. Our results show that ML algorithms can effectively learn many packet signatures including some that inspect the packet payload. We also demonstrate that our system is computationally feasible in that it: (1) can learn flow signatures fast enough to keep up with inherent data drift; and (2) the learned classifiers can operate at very high speeds. This is demonstrated both analytically and empirically.

We analyze our results with an emphasis on understanding why some signatures can be effectively learned whereas others cannot. To this end, we also present a taxonomy of packet signatures that *a priori* separates them into sets (A) that our system will be able to learn perfectly, (B) that our system is likely to learn very well, or (C) where the accuracy of our learned classifier varies based on the nature of the signature. For signatures that fall into classes (B) or (C), where there is *a priori* uncertainty regarding how well our system will perform, we detail the

properties of the signatures that are successfully learned using examples from our set of signatures.

The rest of the chapter is organized as follows. We discuss related work in Section 2.2. A taxonomy of packet signatures is presented in Section 2.3. In Section 2.4 we discuss the relevant aspects of how signature-based detection systems are used in practice, including some specifics on Snort rules and of flow level features that we employ. The operation of ML algorithms, and an algorithm that we find effective, namely, Adaboost, are reviewed in Section 2.5. Section 2.6 describes our dataset and experiment setup; our performance evaluation methodology is presented in Section 2.7, including detection accuracy metrics we have used. This prepares for our experimental evaluation results in Section 2.8, in addition to further analysis of the signatures whose detection performance our *a priori* taxonomy cannot predict. A proposal for how our system could fit into a distributed anomaly detection architecture is then presented in Section 2.9. The computational efficiency of our system, both in terms of learning and classifying flows according to given packet-level signatures, is discussed in Section 2.10. before we present our conclusions in Section 2.11.

2.2 Related Work

There is an extensive recent literature on automating the detection of unwanted traffic in communications networks, most importantly, detection of email spam, denial of service attacks and other network intrusions. Anomaly detection has been used to flag deviations from baseline behavior of network traffic learned through various unsupervised methods, including clustering, Bayesian networks, PCA anal-

ysis and spectral methods; see, e.g., [46, 73, 2, 76, 86, 6]. Our approach is different from these: rather than alarming unknown unusual events based on deviation from observed norms, we regard the set of events alerted by packet rules as representing the most complete available knowledge. The function of ML is to determine how best to reproduce the alerts at the flow level.

ML techniques have been used for traffic application classification. Approaches include unsupervised learning of application classes via clustering of flow features and derivation of heuristics for packet-based identification [9]; semi-supervised learning from marked flow data [22] and supervised learning from flow features [54, 37].

2.3 A Packet Signature Taxonomy

We adopt the following model and classification for packet rules. A packet rule is specified by a set of predicates that are combined through logical AND and OR operations. We classify three types of predicate: flow-header (FH), packet payload (PP), and meta-information (MI) predicates.

FH predicates involve only packet fields that are reported exactly in any flow record consistent with the packet key. This includes source and destination IP addresses and UDP/TCP ports, but exclude packet header fields such as IP identification (not reported in a flow record) and packet length (only reported exactly in single packet flows).

PP predicates involve the packet payload, i.e. excluding network and transport layer headers present.

MI predicates involve only packet header information that is reported either inexactly or not at all in the flow record (e.g., the IP ID field). From the above

discussion, packet length is MI, as are TCP flags, because being cumulative over flows of packets, they are reported exactly only for single-packet flows.

Packet rules may contain multiple predicates, each of which may have different types of (FH, PP, MI) associated with it. We give a single type to the rule itself based on the types of predicates from which it is composed. We partition the set of possible packet rules into disjoint classes based on the types of predicates present. The classification sits well with the performance of our ML-method, in the sense that rule class is a qualitative predictor of accuracy of learned flow-level classifiers. Our packet rule classification is as follows

Header-Only Rules: comprise *only* FH predicates.

Payload-Dependent Rules: include at least one PP predicate.

Meta-Information Rules: include no PP predicates, do include MI predicates, and may include FH predicates.

The relationship between the classification of packet rules and the classification of the underlying predicates is illustrated in Figure 2.1; each circle illustrates the set of rules with attributes corresponding to the predicate classification FH, PP, and MI. The packet rule classification is indicated by colors.

2.4 Packet and Flow Rules in Practice

Snort [74] is an open-source IDS that monitors networks by matching each packet it observes against a set of rules. Snort can perform real-time traffic and protocol analysis to help detect various attacks and alert users in real time. Snort employs a pattern matching model for detecting network attack packets using identifiers such as IP addresses, TCP/UDP port numbers, ICMP type/code, and strings obtained

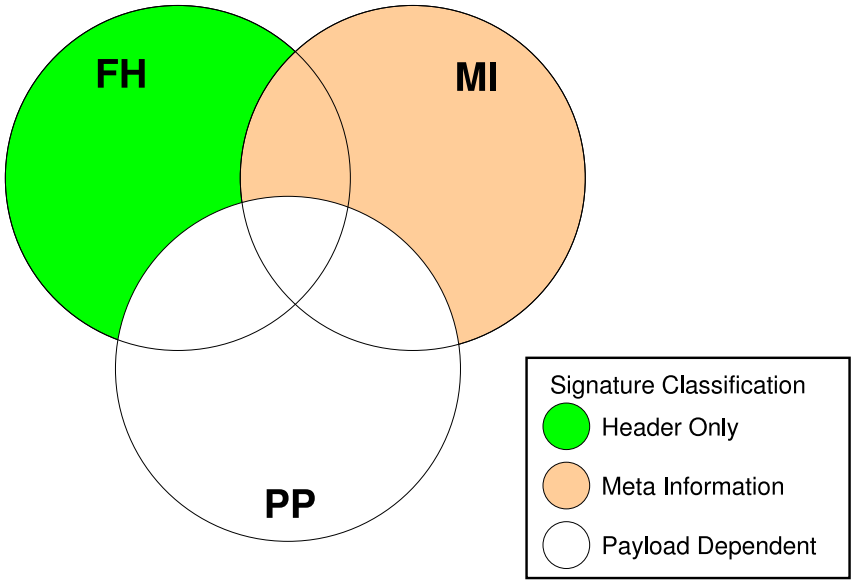

Figure 2.1: Packet Rule Classification: FH (flow header), PP (packet payload), MI (meta-information) indicate rule attributes according to predicate classes; disjoint packet rule classification illustrated by different colors.

in the packet payload. Snort's rules are classified into priority classes, based on a global notion of the potential impact of alerts that match each rule. Each Snort rule is documented along with the potential for false positives and negatives, together with corrective measures to be taken when an alert is raised. The simplicity of Snort's rules has made it a popular IDS. Users contribute rules when new types of anomalous or malicious traffic are observed. A Snort rule is a boolean formula composed of predicates that check for specific values of various fields present in the IP header, transport header, and payload.

Our flow-level rules were constructed from the following features of flow records: source port, destination port, #packets, #bytes, duration, mean packet size, mean packet inter-arrival time, TCP flags, protocol, ToS, "source IP address is part of Snort home net", "destination IP address is part of Snort home net", "source IP

address is an AIM server", "destination IP address is an AIM server". The Snort home net is commonly configured to whatever local domain the operator desires to protect from attacks originating externally.

We construct *flow level predicates* in the following ways:

(1) For categorical features like protocol or TCP flags, we use as many binary predicates as there are categories. For example, if the protocol feature could only take on the values {ICMP, UDP, TCP} then an ICMP packet would be encoded as the predicate ICMP=1, UDP=0, and TCP=0.

(2) For numerical features such as #packets, we want to be able to finely threshold them, so that a rule with a predicate specifying, e.g. an exact number of packets, can be properly captured. Our predicates take the form "feature > threshold".

Our system seeks to leverage ML algorithms in order to raise Snort alerts on flow records. To train our ML algorithms we require concurrent flow and packet traces so that the alerts that Snort raises on packets can be associated with the corresponding flow record. "Correspondence" here means that the packets and flow originate from the same underlying connection. In other words, if Snort has raised an alert on a packet at time t then we locate the flow with the same IP 5-tuple, start time t_s, and end time t_e such that $t_s \leq t \leq t_e$. We then associate the packet alert with the flow. A single packet may raise multiple Snort alerts, and a single flow will often correspond to a sequence of packets, which means that individual flows can be associated with many Snort alerts.

2.5 Machine Learning Algorithms

Formally our task is as follows. For each Snort rule our training data takes the form of a pair (x_i, y_i) where flow i has flow features x_i, and $y_i \in \{-1, 1\}$ indicates whether flow i triggered this Snort rule. Our aim is to attribute to each Snort rule a score in the form of a weighted sum $\sum_k w_k p_k(x)$ over the flow level predicates $p_k(x)$ described in Section 2.4. When this score exceeds an *operating threshold* θ, we have an *ML Alarm*. Since ML alarms should closely mimic the original Snort alarms y_i, the weights w_k are chosen to minimize the classification error $\sum_i I(y_i \neq \mathsf{sign}(\sum_k w_k p_k(x) - \theta)$. However, deployment considerations will determine the best operating threshold for a given *operating point*.

Supervised linear classifiers such as Support Vector Machines (SVMs) [79], Adaboost [69] and Maximum Entropy [19] have been successfully applied to many such problems. There are two primary reasons for this. First, the convex optimization problem is guaranteed to converge and optimization algorithms based either on coordinate or gradient descent can learn millions of examples in minutes (down from weeks ten years ago). Second, these algorithms are regularized and seldom overfit the training data. This is what our fully automated training process requires: scalable algorithms that are guaranteed to converge with predictable performance.

Preliminary experiments established that, on average, Adaboost accuracy was significantly better than SVMs. In the remainder of this section we therefore highlight the properties of Adaboost that make it well-suited for our application. A linear algorithm like Adaboost works well here because the actual number of features is large. In theory, each numerical feature (*e.g.*, source port or duration) may generate as many flow level predicates of the form "feature > threshold" (such

Flow type	wk 1	wk 2	wk 3	wk 4
Neg:no alerts	202.9	221.8	235.9	251.6
Unique neg.	41.8	48.3	42.7	48.7
Pos:some alert	6.7	7.2	6.5	6.9
Unique pos	0.1	0.1	0.1	0.1

Table 2.1: Number of flows in millions per week

predicates are called *stump classifiers*) as there are training examples. In practice, this potentially large set of predicates does not need to be explicitly represented. Adaboost has an incremental greedy training procedure that only adds predicates needed for finer discrimination [69].

Good generalization is achieved from classifiers that represent the "simplest" linear combination of flow-level predicates. Adaboost uses an L_1 measure of simplicity that encourages sparsity, a property that is well matched to our aim of finding a small number of predicates that are closely related to the packet level rules. This contrasts with the more relaxed L_2 measure used by SVM's, which typically produces more complex classifiers. Finally, while Adaboost is known for poor behavior on noisy data, the low level of noise in our data makes the learning conditions ideal. In preliminary experiments, we observe a similar behavior with L_1-regularized Maximum Entropy[19], an algorithm that is much more robust to noise.

2.6 Data Description and Evaluation Setup

The data was gathered at a gateway serving hundreds of users during August–September 2005. We examined all traffic traversing an OC-3 link attached to a border router, gathered via an optical splitter. A standard Linux box performed the role of a monitor reading packets via a DAG card. Simultaneously, unsampled

Protocol	Flag value	Alerts	No alert
ICMP	1	.383	88.5
TCP	6	.348	55.3
UDP	17	6.79	77.1

Table 2.2: Number of flows in millions per protocol for Week 2

netflow records were also collected from the router. Snort rules in place at the site were used for the evaluation. The traffic represented 5 Terabytes distributed over 1 Billion flows over 29 days, i.e., an average rate of about 2MBytes/second. The average number of packets per flow was 14.5, and 55% of flows comprised 1 packet.

We split the data into 4 weeks. Week 1 is used for training only, week 2 for both training and testing and weeks 3-4 for testing only. Table 2.1 reports the number of flows each week. The 200–250 million examples collected each week would represent a major challenge to current training algorithms. Fortunately, the number of unique examples is usually 40–50 million per week, and of these only about 100,000 contain an alert. These can train optimized implementations of Adaboost or SVMs in a span of hours. Removing purely deterministic features greatly simplifies the training problem by reducing the number of examples; it also slightly improves performance. The two main deterministic features are:

source IP is part of local network: Snort rules usually assume that alerts can only be caused by external flows, which means that they require this feature to be 0. After computing unique flow statistics, there were 54 million local and 167 million external flows which are not alerts, zero local and 7 million external flows which are alerts. Making a boolean decision that all local flows are safe, prior to running the classifier, reduces the training data by 54 million examples.

protocol: Snort rules only apply to a single protocol, so splitting the flows into

ICMP, TCP and UDP defines 3 smaller learning problems, minimizing confusion. Table 2.2 shows how week 2 can be split into 3 subproblems, where the most complex one (UDP) only has 6.79 million alert flows and 77.1 million no-alert flows.

Alerts of 75 different rules were triggered over the 4 week trace. We retained the 21 rules with the largest number of flows over weeks 1 and 2; the resulting rules are listed in Table 2.3. The second column reports the total number of flows associated with the rule over week 1 and 2, which range from 13 million to 1360 (note that most rules are evenly distributed over the 4 weeks). The third column reports the number of unique flows, which is representative of the complexity of a rule, being the number of positive examples used in training. (The remaining columns are discussed in Section 2.8).

2.7 Detection Performance Criteria

Each rule is associated with a binary classifier that outputs the confidence with which the rule is detected on a given flow. A detection is a boolean action, however, and therefore requires that an operating threshold is associated with each classifier. Whenever the classifier outputs a confidence above the operating threshold, an alarm is raised. It is customary in the machine learning literature to choose the operating threshold that minimizes the classification error, but this is not necessarily appropriate in our setting. For example, a network operator may choose to accept a higher overall classification error in order to minimize the *False Negative* rate. More generally, the network operators is best equipped to determine the appropriate trade-off between the *False Positive* (FP) and *True Positive* TP rates. The **Receiver Operating Characteristics** (ROC) curve presents the full trade-off for

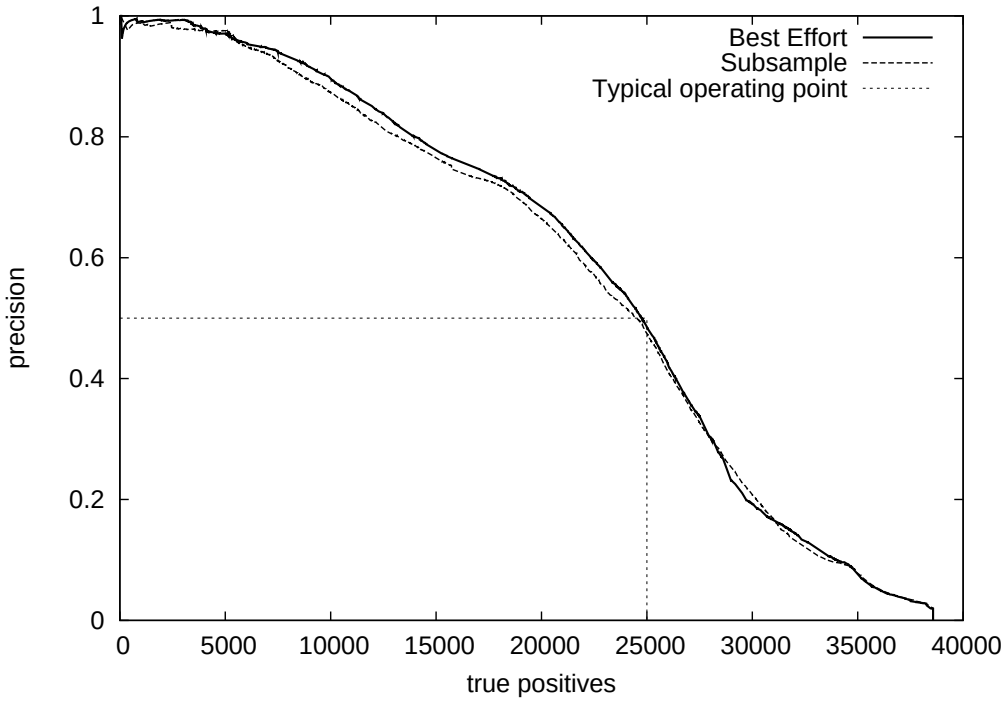

Figure 2.2: Precision vs. number of true positives for the EXPLOIT ISAKMP rule training on week1 and testing on week2

binary classification problems by plotting the TP rate as a function of the FP rate. Each point on the ROC curve is the FP and TP values for a specific confidence (*i.e.*, operating threshold) between 0 and 1.

The ROC curve is useful for network operators because it provides the full trade-off between the FP and TP rates, but this also makes it a poor metric for us when evaluating a number of rules in a number of different settings. For our purposes we require a single threshold-independent number that must account for a range of thresholds. The most studied such measure is the **Area Under the ROC Curve** (AUC), but all our experiments return AUC values better than 0.9999. Besides the fact that such values make comparisons problematic, they are often meaningless. The **Average Precision** (AP), defined in (2.1) below, provides a

pessimistic counterpart to the optimistic AUC. When setting the threshold at the value of positive example x_k, the numbers of total and false positives are $TP_k = \sum_{i=1}^{n_+} 1_{x_k \leq x_i}$ and and $FP_k = \sum_{j=1}^{n_-} 1_{x_k \leq z_j}$, where i and j label the n_+ positive examples and n_- negative examples z_j respectively. The precision at threshold x_k is the fraction of correctly detected examples $\frac{TP_k}{TP_k + FP_k}$ and we compute its average over all positive examples

$$AP = \frac{1}{n_+} \sum_{k=1}^{n_+} \frac{TP_k}{TP_k + FP_k} \tag{2.1}$$

The AP reflects the negative examples which score above the positive examples, and, unlike the AUC, ignores the vast majority of negative examples whose scores are very low. A benefit of the AP metric is that it is more interpretable. Suppose we run Snort until it detects a single alert, and then set up the detection threshold at the classifier output for this alert. Assuming the alerts are I.I.D., an AP of p means that, for each true positive, one can expect $\frac{1-p}{p}$ false negatives.

Let us illustrate what AP means with an example drawn from the results detailed in the next section. Figure 2.2 plots the precision as a function of the number of TP for the EXPLOIT ISAKMP rule: the AP corresponds to the area under this curve. We can see what a comparatively low AP of 0.58 for this rule means in terms of the operating curve. We are able to alert on say 25,000 of the Snort events (about 2/3), while suffering the same number of false negatives, *i.e.* a precision of 0.5. We will see in the next section that for many other rules we do far better, with AP close to 1, leading to very small false positive rates. Moreover, we will explain how a classifier with an AP of 0.5 can still be very useful to a network operator.

2.8 Experimental Results

2.8.1 Baseline Behavior

The average precisions of our learned flow detectors are reported in Table 2.3. We have grouped alerts according to the taxonomy presented in Section 2.3. For each category we perform a simple *macro-average*, where the AP for each rule is given equal weight, which is reported in the *average* row beneath each rule group. The *baseline* column in Table 2.3 reports the AP from training on one full week of data and testing on the subsequent week. We perform two such experiments: the *wk1-2* column uses week 1 for training and week 2 for testing whereas *wk2-3* uses week 2 for training and week 3 for testing. For header and meta-information rules, the baseline results give an AP of at least 0.99 in all cases. Payload rules exhibit greater variability, ranging from about 0.4 up to over 0.99. Our analysis will illuminate the different properties of rules that lead to this variation in ML performance.

There were two payload rules that exhibited dramatically lower AP than the others; these are listed at the end of Table 2.3 and not included in the macro-average. A detailed examination of the underlying Snort rules showed these to be relatively complex and designed to alarm on a mixed variety of predicates. We posit that the complexity of the Snort rules contributes to the difficulty in accurately characterizing them based on flow features.

2.8.2 Data Drift

The main information provided in Table 2.3 concerns the dependence of the AP as a function of the temporal separation between the training data and the test data. Measuring how performance drifts over time is critical, as it determines how often

retraining should be applied. While our baseline corresponds to a 1-week drift, *wk1-3* indicates a 2 week drift: it can either be compared to *wk1-2* (same training data, drifted test data) or *wk2-3* (drifted training data, same test data). In both cases, the difference from a 1-week drift to a 2-week drift is often lower than the difference between *wk1-2* and *wk2-3*: this suggests that the impact of a 2-week drift is too low to be measurable. On the other hand, the loss in performance after a 3 week drift (*wk1-4*) is often significant, in particular in the case of Payload and Meta-Information rules.

2.8.3 Sampling of Negative Examples

Because the number of negative examples far exceeds the number of positive training examples, (*i.e.*, the vast majority of packets—and flows—do not raise any Snort alarms), we expect that sampling to reduce the number of negative examples will have minimal impact on detection accuracy, but will drastically reduce the training time. We wish to preferentially sample examples whose features are more common, or conversely, avoid the impact of noise from infrequently manifested features. For this reason we group the negative examples into sets with identical features, then apply Threshold Sampling [21] to each group as a whole. This involves selecting the group comprising c examples with probability $\min\{1, c/z\}$ where z is chosen so as to sample a target proportion of the examples.

The results for a sampling rate of 1 in 100 negative examples are shown in the two columns labeled *Sampling*, rightmost in Table 2.3. When comparing either the *wk1-2* or the *wk2-3* columns in the baseline and in the sampled case, there is a measurable loss in performance. This loss is small relative to fluctuations in performance from one week to another, however, which suggests that sampling

negative training examples is an effective technique. In this example, sampling speeds up training by about a factor of 6. Without sampling, training a single rule takes on average 1 hour on a single Xeon 3.4GHz processor, but can be reduced to 10 minutes with sampling.

2.8.4 Choosing an operating point

Choosing an appropriate operating threshold can be challenging. That is, above which confidence do we trigger an ML alarm? We have introduced precision, which is the proportion of ML alarms which are also Snort alarms. Another useful concept is the *recall*, which is the proportion of Snort alarms which are also ML alarms. A detector is perfect when both the precision and recall are 1, which in our case often happens for header and meta-information rules.

The story is more complicated for payload rules. The first 2 columns in Table 2.4 report the precision for thresholds chosen to obtain a recall of 1 and 0.99, respectively. We see that we can get both high precision and recall only for the "MS-SQL version overflow attempt" and "ICMP PING CyberKit 2.2 Windows" rules. For all the rules whose average precision is below 0.7, the precision falls to near 0 for high recall values.

In cases where human post-processing is possible, high recall/low precision operating points can still be very useful, especially when the number of alarms is much lower than the total number of examples. As we can see in the last 2 columns in Table 2.4, even rules with comparatively low AP scores only raise alarms for a small percentage of flows to guarantee a recall of 1 or 0.9. For instance, the "APACHE WHITESPACE" rule, with an average precision below 0.6, can deliver a 0.99 recall while alerting on only 0.1% of the flows. While human examination of false positives

is not possible in our case, one can imagine a scenario where Snort itself is run on traffic that generates ML alarms. That is, the ML system is able to process data efficiently but occasionally produces too many false positives; however, the slower Snort could then process traffic for all alarms raised by the ML system (which would still be a very small fraction of total traffic), which would remove the false positives. This represents an interesting direction for future study but is beyond the scope of this dissertation.

2.8.5 Detailed Analysis of ML Operation

In Section 2.3 we presented a taxonomy of Snort rules that distinguishes them according to the types of packet fields they access. "Payload rules" contain at least one predicate that inspects a packet's payload, "header rules" contain only predicates that can be exactly reproduced in a flow setting, and "meta rules" encompass all other Snort rules. Given enough training examples, a ML algorithm will be able to learn to perfectly classify flows according to header rules, whereas payload rules are generally much more challenging. As our results indicate, however, there are many meta rules that can be learned perfectly, and some payload rules as well.

We must delve deeper into the classifiers in order to understand the variability of detection accuracy within the payload and meta groups. Recall from Section 2.5 that a trained classifier is a *weighted* sum over each predicate. Since each predicate operates on a single feature (*e.g.*, TCP port, packet duration), this weight can provide intuition into the relative importance of this predicate to the classifier. For example, which of the destination port number or the flow duration is most important in order to correctly classify Slammer traffic? The standard way to measure the relative importance of each feature for a classifier is to measure the detection

accuracy when the feature is removed. Thus, we train the classifier using all features, but then remove the given feature from consideration during classification: if detection accuracy goes down then clearly this feature was important. Table 2.5 reports the results of doing precisely this: each column reports the AP when the feature for that column is ignored during classification.

Table 2.5 demonstrates that Adaboost is able to correctly *interpret* (as opposed to merely mimic) many header rules by prioritizing the proper fields: the destination port, which encodes the ICMP code and type fields, is essential to each of the ICMP rules. Moreover, the meta rules that are learned well tend to inspect packet-header fields that are reported inexactly in flows, *e.g.*, packet payload size or TCP flags. The "SCAN FIN" rule is raised by Snort when only the FIN flag is set in a TCP packet. When we inspected the exact classifier generated by Adaboost (*i.e.*, this includes the chosen thresholds) for this rule, we found that Adaboost learns to raise this alarm whenever the aggregated TCP flags field in the flow header has a set FIN flag either by itself, combined with SYN, or combined with SYN and RST. As expected, no alarm is raised if the flow TCP flag field has FIN and ACK set.

Predicates that require access to packet payload information, on the other hand, cannot be reproduced in a flow setting whatsoever. For payload rules to be learned in a flow setting, therefore, the corresponding flow classifier must rely on some combination of (A) other predicates of the original Snort rule, and (B) entirely new predicates constructed by the ML algorithm to describe the packets/flows matching these rules. Table 2.5 contains several instances of each, and we will further investigate two (*viz.* "ICMP PING **CyberKit** 2.2 Windows" and "**MS-SQL** version overflow attempt") by inspecting the precise classifier generated by Adaboost.

The MS-SQL rule has several predicates, including one that matches a specific

destination port number, one that inspects the size of the packet payload, and one that looks for a string pattern in the payload itself. Adaboost learns the first predicate exactly, but learns a mean packet size predicate that is more precise than the Snort equivalent. That is, whereas Snort requires that the packet payload size must be greater than 100 bytes, Adaboost requires that the mean packet size should be 404 bytes, which in fact is the exact length of a SQL Slammer packet. (Indeed, the corresponding rule has been used in some cases to help identify Slammer traffic [55].) Combining this predicate and the destination port number, Adaboost learns this rule with high accuracy.

CyberKit is another payload rule that is learned by Adaboost with a high degree of accuracy. Table 2.5 shows that the important features for this classifier are (a) the destination port number, (b) the mean packet size, and (c) whether or not the target host is part of the configured local domain ("dest IP local"). The first and last of these features are a part of the Snort specification, but the mean packet size predicate is not. Adaboost tells us that flows that trigger this Snort alarm have a mean packet size between 92 and 100 bytes per packet.

The ability of ML algorithms to generate predicates independent of the original Snort specification is why ML algorithms are necessary over more rudimentary techniques. For example, a technique that identifies and translates only the flow and meta predicates from Snort rules (*i.e.*, those predicates that can be translated either exactly or approximately) would perform worse in the case of MS-SQL. While such simpler techniques would perform equally well for header rules, they would be ineffective for the majority of payload rules where only a ML approach has a chance to perform well.

2.9 Implementation Scenarios and Architecture

We now describe how our approach could be exploited for flow alerting at network scale, in an architecture illustrated in Figure 2.3, whose components we now describe.

Flow Records: are collected from a cut set of interfaces across the network topology (edge and/or core) so that all traffic traverses at least one interface at which flow records are generated. The flow records are exported to a collector.

Packet Monitor / Alerter: a small number of packet monitors are located at sites chosen so as to see a representative mix of traffic. Each is equipped with a set of packet level rules which are applied to the observed packet stream. Alerts produced by the packet rules are forwarded to the ML trainer.

Machine Learning Trainer: correlates packet alerts with flows generated from the same traffic, and generates the set of flow level alerting rules. The rules are updated periodically, or in response to observed changes in traffic characteristics.

Runtime Flow Classifier: applies flow-level rules to all flow records, producing flow-level alerts.

Future Work. Section 2.10 investigates the scaling properties of computation required in this architecture. Now we consider the ML aspects that require further study. Our evaluation used a single dataset for learning and testing. But the architecture requires that flow-level rules generated by ML on data gathered at a small number of sites can accurately alarm on flows measured at other sites. We propose to extend the study to multiple datasets gathered from different locations, training and testing on different datasets. One question is whether differences in the distribution of flow features such as duration, due, e.g., to different TCP dynamics across

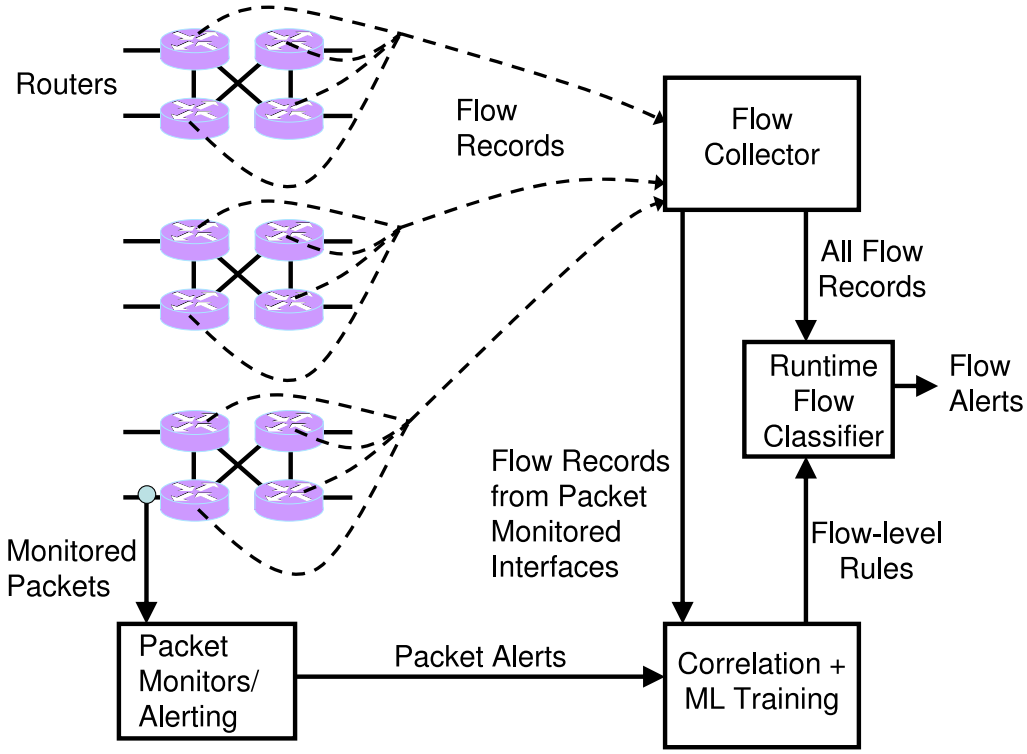

Figure 2.3: Machine Learning Based Flow Alerting System

links of different speeds, could impair the accuracy of cross-site alarming. A further matter is to investigate the effect on detection accuracy if using packet sampled flow records for learning and classification.

2.10 Computational Efficiency

We analyze the computational speed of the three phases of our scheme: (i) correlation of flow records with Snort alarms prior to training; (ii) the ML phase; (iii) run-time classification of flows based on the learned flow rules. We combine analysis with experimental results to estimate the resources required for the architecture of Section 2.9. We consider two scenarios.

A: Scaling the interface rate: what resources are needed to perform correlation and ML at a higher data rate? We consider traffic equivalent to a full OC48 link (corresponding to a large ISP customer or data center). At 2.5Gbits/sec this is a scale factor 150 larger than our test dataset; we assume the numbers of positive and negative examples scale by the same factor.

B: Scaling classification across sites: Consider a set of network interfaces presenting traffic at rate of our data set; at 2MB/sec this represents medium sized ISP customers. The flow rules are learned from traffic on one of the interfaces. What resources are required to classify flows on the others?

2.10.1 Costs, Implementations, and Parallel Computation

We believe parallelization of correlation and learning steps is reasonable, since the cost is borne only once per learning site, compared with the cost deploying Snort to monitor at multiple locations at line rate. Parallelism for the classification step is

far more costly, since its scale the resources required for at monitoring point. The implementations used here are not optimized, so the numerical values obtained are conservative.

2.10.2 Initial Correlation of Flow and Snort Data

Our prototype system can correlate flow records with Snort alarms at a rate if 275k flows per second on a 1.5 GHz Itanium 2 machine: about 15 minutes to correlate one week's data. Under our scaling scenario A, the hypothetical OC48 would require about 33 hours of computation on the same single processor to correlate one week's data. This task is readily parallelized, the cost borne once prior to the learning stage.

2.10.3 Learning Step

The time taken for Adaboost or the Maxent algorithm we considered [19] to learn a given rule is proportional to the product of three quantities:

- the number of iterations N_i, which is fixed to the conservatively large number of 200 in our problem.
- the total number of +ve and -ve examples $N_e = n_- + n_+$
- the number of candidate weak classifiers N_c that Adaboost must consider

For numerical features, the number of weak classifiers is the number of boundaries that separate runs of feature values from positive and negative examples when laid out on the real line. This is bounded above by the twice the number n_+ of positive examples. We computed the dependence of N_c on data size for sampled

subsets of the dataset; per rule, N_c scaled as n_+^α for some $\alpha < 1$. These behaviors suggest the following strategy to control computation costs for processing traffic while maintaining learning accuracy:

- Use all positive examples;
- Use at most fixed number n_-^0 of negative examples.

Limiting the number of negative examples does not impair accuracy since we still have more positive examples. Computation time is proportional to $N_i N_e N_c \leq 2N_i n_+(n_+ + n_-^0)$. While n_+ is much less than n_-^0—see Table 2.1— computation time scales roughly linearly with the underlying data rate.

To see how this plays out in our hypothetical example, we take our dataset with 1 in 10 sampling of positive examples as representing the reference operating threshold. Hence, from Table 2.1, there are roughly $n_-^0 = 4M$ unique negative examples. For n_+ we take the average number of unique positive examples per rule per week, namely 8861, the average of the second numerical column in Table 2.3. Scaling to OC48 scales $n_+ \to 150n_+$ and hence $n_+(n_+ + n_-^0) \to 150n_+(150n_+ + n_-^0)$. Learning time increases by roughly a factor 200, lengthening the average time per rule from 10 minutes to 33 hours. Although this may seem large, it is conservative and likely unproblematic, since (i) it is far shorter that the data drift timescale of two weeks which should not depend on link speed, and can be reduced by (ii) optimized implementation; (iii) parallelization, once per learning site; and (iv) sampling the positive examples. Sampling may be desirable to control training time for rules with many positive examples, being precisely the rules for which sampling has the least impact on accuracy.

2.10.4 Classification Step

The number of predicates selected by Adaboost is typically around 100: the number of feature lookups and multiply-adds needed to test a rule. The same machine as above is able to apply these predicates, *i.e.* perform flow classification, at a rate of 57k flows/second. Our original dataset presented flows at a rate of about 530 flows/second, so this could nearly accommodate the 150 fold increase in flow rate in Scenario A, or classify flows from 100 interfaces in Scenario B.

2.11 Conclusions

We proposed an ML approach to reproducing packet level alerts for anomaly detection at the flow level; Applying Snort rules to a single 4 week packet header trace, we found:

- Classification of flow-level rules according to whether they act on packet header, payload or meta-information is a good qualitative predictor of average precision.
- The ML approach is effective at discovering associations between flow and packet level features of anomalies and exploiting them for flow level alerting.
- Drift was largely absent at a timescale of two weeks, far longer than the few minutes required for learning.

We proposed and implemented a proof-of-concept architecture to exploit this at network scale. We analyzed the computation complexity of our approach and argued that computation remains feasible at network scale. Although our study focused on single packet alarms produced by Snort, our approach could in principle we applied

to learn from flow records alone, alarms generated by multipacket/flow events of the type monitored by Bro [62].

Alert message	Number of flows over weeks 1-2		Average Precision for wkA-B (week A=train, B=test) Baseline		Drift		Sampling	
	total	unique	wk1-2	wk2-3	wk1-3	wk1-4	wk1-2	wk2-3
Header								
ICMP Dest. Unreachable Comm. Administratively Prohib.	154570	12616	1.00	1.00	1.00	1.00	1.00	1.00
ICMP Destination Unreachable Communication with Destination Host is Administratively Prohibited	9404	3136	0.99	0.99	0.98	0.99	0.99	0.98
ICMP Source Quench	1367	496	1.00	1.00	1.00	1.00	1.00	1.00
average			1.00	0.99	0.99	0.99	1.00	0.99
Meta-information								
ICMP webtrends scanner	1746	5	1.00	0.99	0.99	0.99	0.90	0.99
BAD-TRAFFIC data in TCP SYN packet	2185	2145	1.00	1.00	1.00	0.99	1.00	1.00
ICMP Large ICMP Packet	24838	1428	1.00	1.00	1.00	1.00	1.00	1.00
ICMP PING NMAP	197862	794	1.00	1.00	1.00	1.00	0.61	1.00
SCAN FIN	9169	7155	0.99	1.00	1.00	0.86	0.99	1.00
(spp stream4) STEALTH ACTIVITY (FIN scan) detection	9183	7169	1.00	1.00	1.00	0.87	1.00	1.00
average			1.00	1.00	1.00	0.95	0.92	1.00
Payload								
MS-SQL version overflow attempt	13M	28809	1.00	1.00	1.00	1.00	1.00	1.00
CHAT AIM receive message	1581	1581	0.66	0.57	0.60	0.65	0.56	0.30
EXPLOIT ISAKMP 1st payload length overflow attempt	76155	65181	0.59	0.58	0.57	0.57	0.58	0.56
ICMP PING CyberKit 2.2 Windows	332263	299	1.00	1.00	1.00	1.00	1.00	1.00
ICMP PING speedera	46302	100	0.83	0.81	0.81	0.83	0.83	0.81
(http inspect) NON-RFC HTTP DELIMITER	13683	13653	0.41	0.54	0.57	0.30	0.37	0.50
(http inspect) OVERSIZE REQUEST-URI DIRECTORY	8811	8802	0.96	0.96	0.96	0.96	0.96	0.96
(http inspect) BARE BYTE UNICODE ENCODING	2426	2425	0.41	0.59	0.44	0.40	0.36	0.59
(http inspect) DOUBLE DECODING ATTACK	1447	1447	0.69	0.53	0.66	0.75	0.55	0.36
(http inspect) APACHE WHITESPACE (TAB)	1410	1409	0.47	0.60	0.53	0.59	0.40	0.59
average			0.70	0.72	0.71	0.70	0.66	0.67
(spp stream4) STEALTH ACTIVITY (unknown) detection	1800	1800	0.00	0.01	0.01	0.00	0.00	0.01
(snort decoder) Truncated Tcp Options	26495	25629	0.05	0.06	0.05	0.05	0.05	0.05

Table 2.3: Number of flows and average precision per rule: baseline, drift, and sampling

Alert message	Precision for recall of		Alert % for recall of	
	1.00	0.99	1.00	0.99
MS-SQL version overflow	1.00	1.00	3.0	2.9
CHAT AIM receive message	0.02	0.11	0.0	0.0
EXPLOIT ISAKMP first payload	0.02	0.03	0.9	0.6
ICMP PING				
CyberKit 2.2 Windows	1.00	1.00	0.1	0.0
ICMP PING speedera	0.02	0.83	0.5	0.0
(http inspect)				
NON-RFC HTTP DELIMITER	0.00	0.01	1.3	0.6
OVERSIZE REQUEST-URI DIR.	0.01	0.20	0.1	0.0
BARE BYTE UNICODE ENC.	0.00	0.00	1.1	0.4
DOUBLE DECODING ATTACK	0.00	0.00	1.8	0.4
APACHE WHITESPACE (TAB)	0.00	0.00	1.1	0.1

Table 2.4: Precision and Alarm rate at high recall for payload rules

Rule	base line	dest port	src port	num byte	num pack	dura tion	mean pack size	mean pack intval	TCP flag	IP serv type	dest IP local
Header											
ICMP Dest. Unreachable Comm. Admin. Prohib.	1.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ICMP Destination Unreachable Comm. with Dest. Host Administratively Prohib.	0.99	0.00	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
ICMP Source Quench	1.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.61
average	1.00	0.00	1.00	1.00	1.00	0.99	0.99	1.00	1.00	1.00	0.87
Meta-information											
ICMP webtrends scanner	0.99	0.89	0.99	0.00	0.99	0.99	0.75	0.99	0.99	0.99	0.59
BAD-TRAFFIC data in TCP SYN packet	1.00	0.74	1.00	1.00	1.00	1.00	0.99	1.00	0.50	1.00	1.00
ICMP Large ICMP Packet	1.00	1.00	1.00	1.00	1.00	1.00	0.43	1.00	1.00	1.00	0.96
ICMP PING NMAP	1.00	1.00	1.00	1.00	1.00	1.00	0.02	1.00	1.00	1.00	0.50
SCAN FIN	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.24	0.99	0.99
average	1.00	0.92	1.00	0.80	1.00	1.00	0.64	1.00	0.75	1.00	0.81
Payload											
MS-SQL version overflow attempt	1.00	0.99	1.00	1.00	1.00	1.00	0.83	1.00	1.00	1.00	0.48
CHAT AIM receive message	0.61	0.64	0.61	0.49	0.55	0.42	0.33	0.54	0.29	0.51	0.51
EXPLOIT ISAKMP 1st payload length overflow	0.58	0.15	0.58	0.49	0.26	0.55	0.58	0.57	0.57	0.56	0.57
ICMP PING CyberKit 2.2 Windows	1.00	0.52	1.00	0.95	1.00	1.00	0.77	0.99	1.00	1.00	0.39
ICMP PING speedera	0.82	0.79	0.82	0.07	0.82	0.82	0.06	0.82	0.82	0.81	0.72
(http inspect) NON-RFC HTTP DELIM	0.48	0.02	0.34	0.15	0.47	0.24	0.22	0.32	0.22	0.42	0.42
average	0.75	0.52	0.72	0.52	0.68	0.67	0.46	0.71	0.65	0.72	0.52

Table 2.5: The importance of each feature to a classifier as measured by the AP if the feature is removed during detection

Chapter 3

Evaluating the Potential of Collaborative Anomaly Detection

3.1 Introduction

Unwanted traffic from malicious hosts is a serious problem in the Internet today. DDoS attacks, exploit scanning, email and instant-message spam, click fraud, and other forms of malicious behavior are a common occurrence [24, 35, 68]. Vulnerabilities in network software have led to the rapid proliferation of automated attack methods (worms, botnets, viruses). It is estimated that 25% of all personal computers may be infected by malware [80], and organizations are estimated to lose billions of dollars per year as a result [63].

Defending against attack traffic can be extremely challenging. The stealthy nature of many attacks, where malicious hosts emulate the characteristics of well-behaved traffic, limits the ability of any one host or network to detect or filter malicious activity in isolation. In order to counter this emerging threat, previous

work has proposed that victim sites *collaborate* to build a shared defense against attacks [39, 56, 75, 4].

While the notion of victim collaboration has been previously proposed in the literature, the extent to which it improves our ability to detect and isolate malicious traffic has not been rigorously evaluated. In order to design the most effective mitigation techniques, and to determine how existing collaborative architectures would perform in practice, we need to build an understanding of the sorts of workloads malicious hosts generate across different host sites. Building this understanding requires studying malicious activity across several vantage points, and precisely how these vantage points may monitor traffic and exchange information to best isolate attacks. To the best of our knowledge, our work is the first to directly measure the benefits of victim cooperation on ISP-level traffic traces. In particular, we apply standard network anomaly detectors to identify unwanted traffic, and analyze the ability of a representative set of collaboration schemes to assist the victims in isolating and mitigating these attacks.

Our measurement study is based on IP flow traces from GEANT, a European ISP operated by a consortium of research and educational institutions. We studied all traffic traversing the twenty routers that make up GEANT's European backbone network, during August 2008. We applied standard anomaly detectors [5, 43] to these traces in order to identify unwanted traffic that collaborating hosts and networks may wish to remove, including DoS, port scanning, and IP scanning events. Our final results calculate the fraction of attacks that could have been mitigated by a set of collaborating victim end-hosts.

To mitigate the possibility of false positives, we correlate our detected anomalous events with a DNS blacklist [15]. That is, only anomalies that originated from source

IP addresses listed in the DNS blacklist were considered. We refreshed our local copy of the DNS blacklist every 12 hours throughout our one-month measurement window in order to minimize the probability of DHCP changes affecting our results. Finally, we evaluated collaboration schemes where detected attackers are blocked for a variable period of time, as contrasted with permanent blacklisting.

Our experiments show that the end-hosts and networks that are affected by the unwanted traffic in the Internet have a great deal to gain by collaborating to address their common problem. The potential benefit is phenomenally high for anomaly types that are naturally one-to-many, such as IP scans that can affect hundreds of thousands of victims within minutes. Across all our studied anomalies, even a *random* selection of three thousand victims could potentially mitigate nearly 30% of detected anomalies. Moreover, our experiments also show that malicious hosts often target a small number of victim hosts to attack repeatedly within a short period of time. This means that much of the benefit of victim collaboration can be had by blacklisting malicious hosts for a short period of time after the initial attack. A blacklist duration of one hour captures over 90% of detected anomalies if all victims collaborate.

Roadmap: We start by presenting our data sources, correlation methodology, and anomaly detectors in in Section 3.2. In Section 3.3 we measure the potential benefit that different collaboration schemes would have provided to the victims of the attacks in our traces. We then briefly discuss related work in Section 3.4, before concluding in Section 3.5.

3.2 Methodology

To evaluate the potential of victim collaboration for anomaly detection, we require a large, representative set of unwanted traffic. We accomplish this by analyzing the traffic of IP addresses listed in an amalgamation of DNS blacklists [15]. The two types of traces and our methodology for correlating them are described in Section 3.2.1. Moreover, we wish to characterize the benefit of victim collaboration across multiple types of unwanted traffic, such as port scans and DoS attacks. This necessitates classifying attack patterns from our IP flow traces. We leveraged anomaly detection techniques presented in previous work [5, 43] in order to accomplish this, which is described in Section 3.2.2.

3.2.1 Data Sources

Our traffic traces are from the GEANT ISP network backbone [29]. The GEANT network interconnects 30 National Research and Education Networks representing 34 countries across Europe. GEANT maintains multiple redundant connections to the Internet and provides transit service to its customers. The network operation center routinely collects flow and routing information and makes them available to the research community. Each of the 20 GEANT routers samples 1-in-1000 packets and exports the flow headers to a central collector using a NetFlow-like [14] format. In this study we analyzed traces from all of August 2008. This data set represents 57 gigabytes of flow header information.

We wish to analyze the potential benefit of using victim collaboration to remove *unwanted* traffic. We therefore limit our study to traffic originating from IP addresses that have been identified as malicious in a DNS blacklist (DNSBL). We use

the Composite Blocking List (CBL) [15], a widely-used DNSBL. The CBL is used for the Spamhaus blocking list, which is maintained using several large spam traps. The CBL currently receives over 85% of spam hitting large spamtraps [15].

As with any study measuring characteristics of per-host traffic, associating a flow with a particular host represents a challenge. For example, a dynamically assigned IP address may be used by a spam bot and a legitimate host at different periods of time. To reduce the potential for false positives, we download snapshots of the CBL every 12 hours, and apply the snapshot only to traffic traces for the 12 hours following its download. The list we download at time t is therefore only used to extract flows from our GEANT trace for 12 hours following its download, which means that if a host's IP disappears from the CBL, our methodology will quickly reflect that fact. In other words, our list of hosts originating unwanted traffic is actually a sequence of 62 lists for each of the 31 days in August 2008 (one for each 12 hour period), and the blacklist downloaded at time t is used during the period $[t, t + 12\text{h}\rangle$.

In Section 3.3 we will also investigate the effectiveness of graylisting individual malicious hosts that perform an attack. Specifically, instead of permanently blocking an IP address that has performed an attack at time t, the IP will be blocked for a period of Δ hours, *i.e.* during the interval $\langle t, t + \Delta]$. We refer to Δ as the "blacklist duration" parameter. Such a blacklisting scheme is consistent with other DNSBLs [77] and the associated analysis will allow us to (1) understand how rapidly malicious hosts attack victims, and (2) set Δ to a value smaller than the vast majority of DHCP lease times [83] while still providing most of the benefits of victim collaboration.

3.2.2 Anomaly Detectors

We wish to evaluate the effectiveness of collaborative anomaly detection across a range of specific types of unwanted traffic. In particular, we have chosen to study the effectiveness of collaboration to mitigate DoS attacks, port scans, and IP scans. In order to identify these attacks, we leverage detection techniques developed and presented in previous work [5, 43]. The detectors we have chosen each incorporate a well-defined specification of what constitutes an attack, which make our results easier to understand and reproduce. Recall further that we are only analyzing traffic from source IP addresses that were listed in the CBL DNS blacklist within the past 12 hours, which means that only these addresses are detected as originating unwanted traffic. This fact means that the chance of a falsely detected attack is vastly reduced.

For example, a port scan is defined to occur whenever a given source IP address contacts more than α different destination port numbers on a single destination IP address within a δ-second time window. This model was presented in [5]. Our DoS and IP scan detectors are designed in the same way, namely, a DoS attack is defined as occurring whenever a source IP address opens more than β different connections to a single $\langle$ destination IP address-destination port number $\rangle$-pair, and an IP scan occurs when a source IP address contacts more than γ different destination IP address, also within a given δ-second time window.

Each of our detectors usees a threshold (namely, α for portscans, β for DoS attacks, and γ for IP scans) that defines the separation between an attack and normal traffic. Like any anomaly-detection-based approach, there is no universally appropriate threshold. A network operator employing these detectors would choose a threshold depending on the relative importance placed on avoiding false negatives

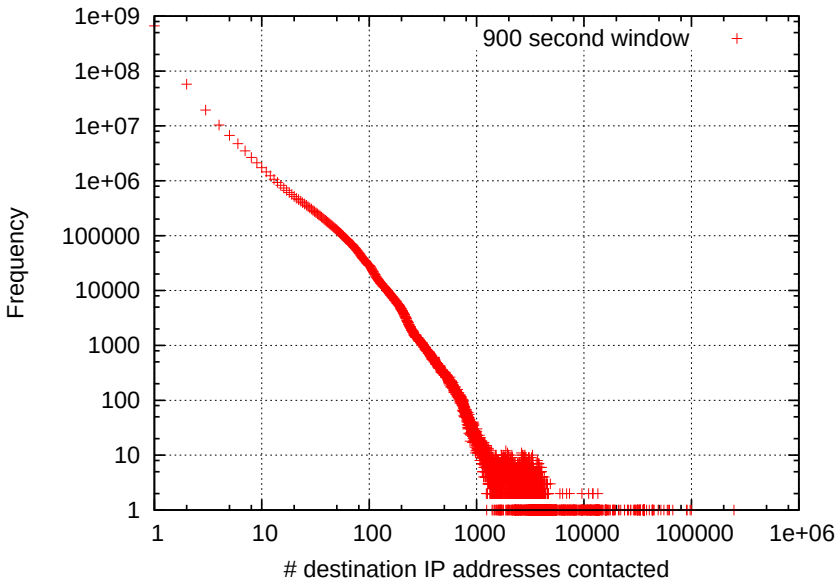

Figure 3.1: Distribution of "number of destination IP addresses contacted" within 15 minutes

Type	Threshold	% Flows
DoS	1000	2.77E-4%
Port Scan	500	1.31E-3%
IP Scan	500	8.21E-3%

Table 3.1: Anomaly Detectors

versus the overhead of manually weeding out false positives. As we aim to study the benefit of collaboration to mitigate *unwanted* traffic, we wish to ensure that the traffic we define as unwanted is indeed so. We therefore prioritize having a low false-positive rate, and use a relatively high threshold for each of our detectors.

To determine an appropriate threshold for our purposes, we study the overall traffic distribution relative to a given detector. For example, recall that an IP scan is defined to occur whenever a given source IP address contacts more than γ destination IP addresses within a δ-second time window. By setting δ to be 15 minutes, we can study the distribution of "number of IP addresses contacted" by a

single source IP address within a 15 minute window. This distribution can be seen in Figure 3.1 for our flow traces. We set the threshold γ to 500, which corresponds to 8.21E-3% of *all* flows. Recall again that we chose to study only anomalies originating from IP addresses listed in the CBL, which means the fraction of studied flows is even smaller. We tuned our other detectors in the same manner by studying their distributions, and the chosen thresholds are specified in Table 3.1.

3.3 Victim Collaboration

The methodology presented in Section 3.2 ultimately gives us a timeseries of anomalies. That is, once we apply the anomaly detectors described in 3.2.2 to the traffic originating from IP addresses listed in the CBL DNS blacklist as described in 3.2.1, we get a sequence of anomalies $\langle \chi, \nu \rangle_t^\tau$, where χ is the attacking IP address (henceforth referred to as the attacker), ν is the victim IP address (victim), t is the time of the attack, and τ is the type of attack (either DoS, port scan, or IP scan).

We wish to evaluate the potential effectiveness of a victim collaboration scheme that would block attacking IP addresses after such an attack. That is, we will consider collaboration schemes where an attack of type τ by attacker χ on victim ν at time t will be blocked for Δ hours, *i.e.*, the interval $\langle t, t+\Delta]$, by the set of victims V if $\nu \in V$. In other words, attacks on any one participant in the collaboration scheme means that the attacking IP address will be blocked by all participants in the scheme for some period of time.

The central parameters in the above scheme are (1) the types of attacks τ covered by the collaboration scheme, (2) the set of victims V that participate in the scheme, and (3) the duration Δ hours that they block attacking IP addresses. In

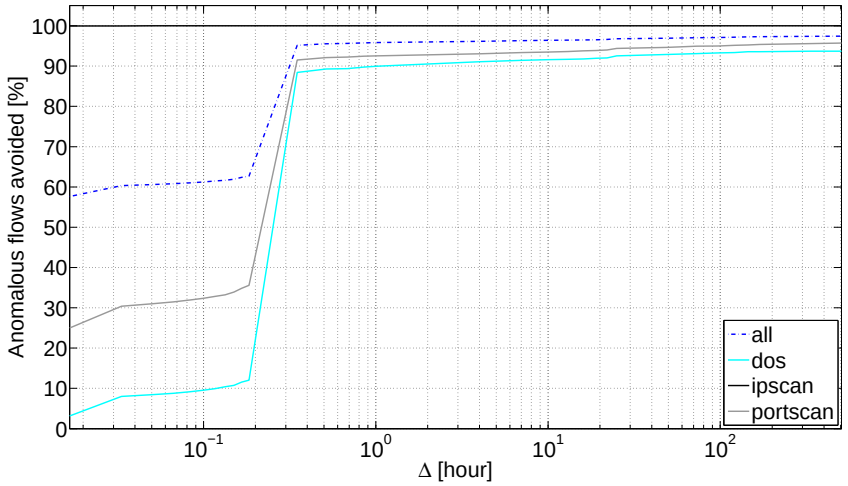

Figure 3.2: Effectiveness of victim collaboration as a function of Δ

the following section we will evaluate the importance of each of these parameters to an effective collaboration scheme to mitigate unwanted traffic. The metric we will use is the fraction of *all* attacks that could have been mitigated if the given collaboration scheme was in place. In other words, any subsequent attack by χ during the interval $\langle t, t + \Delta]$ on any $\nu_i \in V$ will be rendered mitigated, and we will measure the fraction of mitigated attacks as a function of all attacks, regardless of whether the attack targets a victim participating in the scheme.

3.3.1 Blacklist Duration

The "blacklist duration" parameter Δ specifies how long an IP address is blocked after being the source of an attack. More precisely, after an attack $\langle \chi, \nu \rangle_t^\tau$, if the victim ν is a participant in the collaboration scheme (*i.e.*, $\nu \in V$) then attacker χ will be blocked by all $\nu_i \in V$ during the interval $\langle t, t + \Delta]$. We wish to measure the significance of the Δ parameter. Figure 3.2 plots the fraction of attacks that could have been mitigated as a function of Δ. In this experiment we have chosen V to

be the universal set, *i.e.*, all victims collaborate to block source IP addresses for Δ hours after an attack.

Figure 3.2 shows this result. The figure has a separate line for each of our studied anomalies, and one curve titled "all" corresponding to the fraction of attacks that can be blocked across all attack types. The line for DoS attacks, for example, calculates the fraction of all DoS attacks that could have been rendered ineffective if victims collaborated to blacklist attackers for Δ hours. We found attacks that are inherently one-to-many, such as IP scans where a single attacker probes a large number of victim hosts, benefit the most from victim collaboration. This makes sense because the first victim is able to warn all other victims, and there will necessarily be more victims due to the nature of the attack. Also, the figure demonstrates that a vast fraction of the anomalies we detected could have been mitigated with victim collaboration. The figure is an unrealistic upper bound due to the fact that *all* victims are assumed to participate (we investigate sensitivity to this assumption shortly), but it demonstrates that attackers have a very high degree of *fan-out*. That is, each malicious host engages in a great deal of malicious activity. A substantial amount of that activity can potentially be mitigated by victim collaboration.

Finally, Figure 3.2 also tells us that much of the benefits of victim collaboration can be had with a relatively short blacklist duration parameter Δ. In other words, malicious hosts are typically highly active, but in brief bursts. The figure shows that a blacklist duration parameter Δ of approximately one hour achieves the majority of the benefit of collaboration[1]. Note, however, that the exact point of increase for the all, port scan, and DoS curves in Figure 3.2 is an artifact of our methodology since we detect anomalies within a 15 minute time interval. It is likely that the

[1]A one hour blacklist duration is also smaller than the vast majority of DHCP lease times presented in [83]

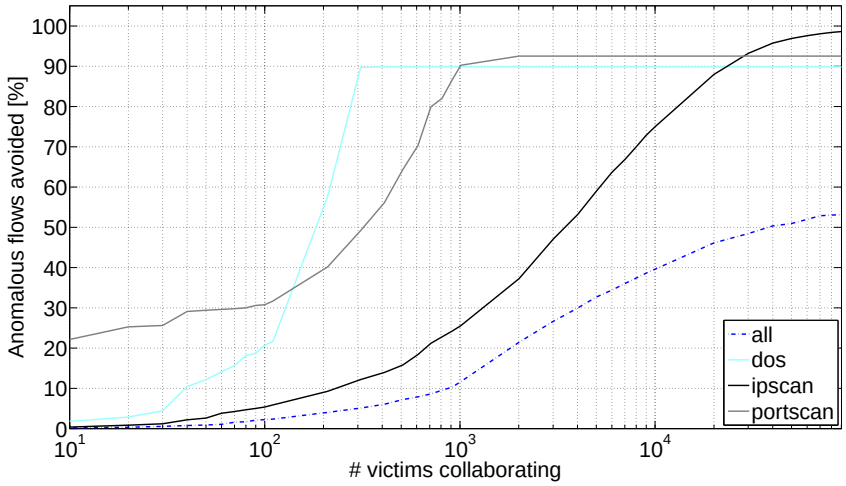

Figure 3.3: Effectiveness of victim collaboration with random victim selection

"true" curves would have a smooth increase in the interval $\Delta \in [0, 1]$ hours.

3.3.2 Set of Collaborators

In Section 3.3.1 we assumed that all hosts would participate in the collaboration scheme while we altered the length of the blacklist duration. In this section we will do the reverse in order to evaluate the impact that the composition of participating hosts has on the effectiveness of a collaboration scheme. For these experiments we have set the blacklist duration parameter Δ to one hour.

Whereas the blacklist duration parameter Δ is one-dimensional (*i.e.*, it can take on values $[0, \infty]$) the composition of the victim collaborator set is inherently multi-dimensional. This difference means that evaluating the importance of the latter parameter is fundamentally more challenging. From an analytical perspective, determining the composition of the set of collaborators can be separated into two distinct parameters: (a) the *number* of hosts that participate in the scheme, and (b) the *distribution* from which these hosts are chosen. This simplifies the process

somewhat, as (a) is one-dimensional, but (b) can still take on a very large number of states. We have chosen to consider three different schemes in determining the manner in which hosts are chosen, which we will discuss below.

Random: First, we randomly select the set of participating hosts, and plot results in Figure 3.3. That is, for any given number of collaborating hosts on the x-axis, the selection of participating hosts is randomly determined. Each data point is the mean benefit of collaboration as specified in Section 3.3.1, averaged over ten runs (each time selecting a random set of participants). One important difference between this Figure and Figure 3.2 is that here the curve representing the benefit of collaborating against IP scans is not always above the curves representing other attack types. Rather, the fraction of mitigated IP scans is smaller when fewer victims collaborate, but is greater than other types of attacks when more than 300 thousand victims collaborate. Recall that, because IP scans are inherently one-to-many, any single IP scan event affects a great deal more victims than a DoS event. For example, the one thousand victims that collaborate to mitigate 26% of IP scans represent less than 1% of all IP scan victims whereas the one thousand victims that collaborate against port scans encompass nearly all victims of port scans in our one month trace. In other words, while IP scans need more *total* number of victims to collaborate in order to achieve the same benefit as other attack types, the *fraction* of IP scan victims that need to collaborate is smaller.

The second key observable difference between Figures 3.2 and 3.3 is that the curve for mitigating "all" attacks is at the bottom in the latter figure. This show that, for a random victim ν of a *specific* type of attack τ, ν is more likely to share attackers with other victims of τ than ν is to share attackers with victims of *any* type of attack. In other words, while malicious hosts engage in a wide variety of

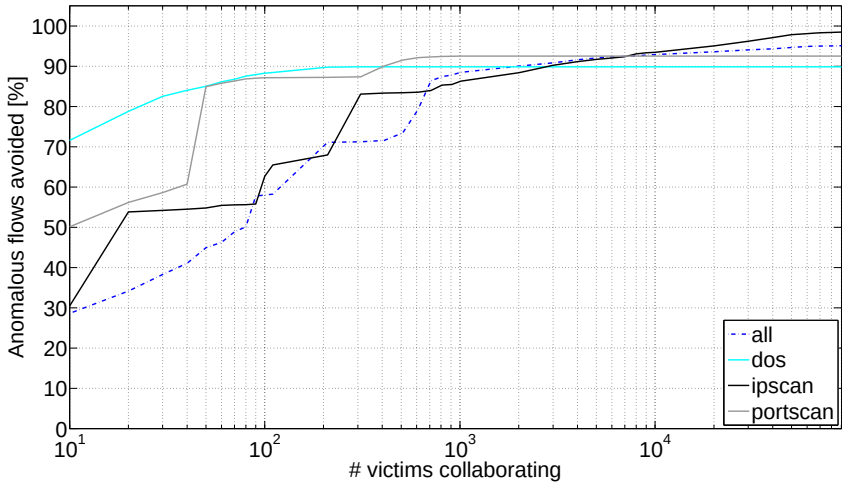

Figure 3.4: Effectiveness of victim collaboration when victims are chosen based on how often they are attacked

different types of attacks over long time scales, they tend to perform only a single type of attack within shorter time scales (*i.e.*, one hour in our experiments).

Most victimized hosts: In a real deployment, it is unlikely that the distribution of victims that choose to participate in the collaboration scheme is entirely random. Rather, end-hosts that are attacked often may have a stronger incentive to participate, as they have the most to gain from doing so. Hence, here we consider the case where the most-victimized hosts (the ones that are attacked most often) are the first to join the collaboration scheme. Figure 3.4 plots the fraction of attacks that could have been mitigated for each of our studied anomaly types as a function of the number of participating victims, where the victims are chosen based on how often they are attacked. The benefits of collaboration are understandably much higher for this scenario compared to the random selection shown in Figure 3.3. Figure 3.4 demonstrates that, in addition to being able to affect a large number of victims in a small amount of time (*i.e.*, a high degree of fan-out), malicious hosts are also intense *repeat-offenders*. That is, malicious hosts are able to do a great deal of damage very

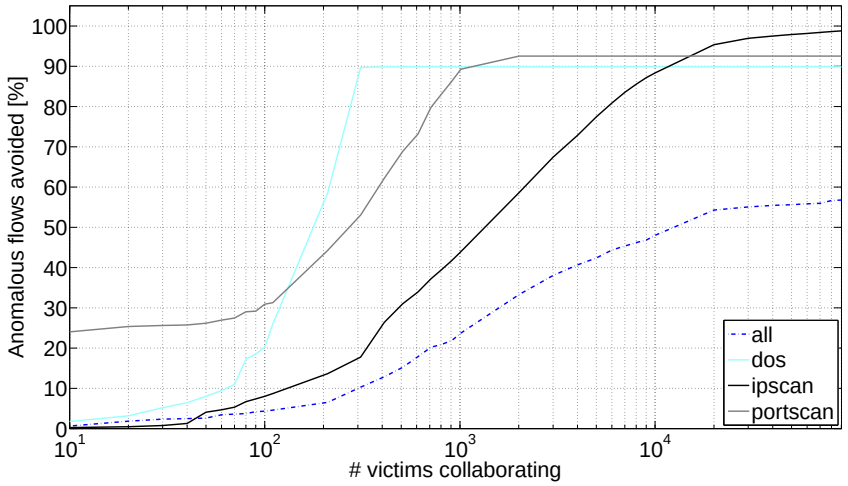

Figure 3.5: Effectiveness of victim collaboration with a weighted random victim selection

quickly to their chosen targets. These victims bear a large fraction of the burden for some of the attack types, and therefore a large fraction of attacks could potentially be mitigated with only a small number of well-chosen participating victims.

Weighted random: For completeness, we explore the space between the random and most-victimized approaches to host selection. We do this by considering a *weighted random selection* strategy, where the probability of selecting a host is a function both of a random variable, and the rate at which the host is attacked. In particular, for Figure 3.5 the benefit of collaboration is reported as a function of the number of victims collaborating, where a victim that is attacked twice as often has twice the probability to be included in the scheme. Whereas Figure 3.3 tells us that a random 1000 victims can expect to mitigate 12% of attacks, Figure 3.5 shows that a weighted random selection of 1000 victims could mitigate 25% of the attacks in our trace.

3.4 Related Work

A great deal of research work has been done to allow enterprises and networks to correlate observations from various vantage points in order to improve anomaly detection. The majority of this work has analyzed traffic traces and leveraged general statistical techniques, *e.g.*, [45]. While these techniques have shown promise for intranetwork anomaly detection, they have not been extended to cross-organizational settings where there will be many more vantage points and thus the computational expense of the correlation is much greater. The most well-known technique for such victim collaboration is the sharing of spam blacklists, *e.g.*, [77]. Recently researchers have also proposed leveraging victim collaboration in response to other threats such as worms [56] and self-propagating code [39]. See, for example, [4]. Our work estimates the expected benefit of such *proposed* schemes, thereby providing strong impetus for their development and deployment. The only other work we are aware of that investigates the degree of correlation in Internet attack traffic is [41], but they focus on IDS traces as the underlying data whereas we correlate detections by standard anomaly detectors applied to flow traces from an operational network.

3.5 Conclusions

Given the extreme challenges in identifying and filtering unwanted traffic, some form of victim collaboration seems necessary. This chapter characterizes the ability of victims to establish a shared defense against attacks, over a variety of attack types. We show that malicious hosts often have a high degree of *fan-out*, affecting a large number of victims. These victims therefore have a great deal to gain by collaborating.

In Section 3.2 we explain that we are only interested in originators of *unwanted* traffic and therefore limit our study to those IP addresses that engage in anomalous activity *and* are listed in the CBL DNS blacklist. For our future work we wish to investigate whether this biases our correlation results due to preexisting correlation among IP addresses listed in the CBL.

Chapter 4

Privacy-Preserving Collaborative Anomaly Detection

4.1 Introduction

Many important data-analysis applications must combine and analyze data collected by multiple parties. Such distributed data analysis is particularly important in the context of security. For example, victims of denial-of-service (DoS) attacks know they have been attacked but cannot easily distinguish the malicious source IP addresses from the good users who happened to send legitimate requests at the same time. As demonstrated in Chapter 3, victims of attacks can improve their individual detection accuracy by collaborating because compromised hosts often participate in multiple attacks. Cooperation is also useful for Web clients to recognize they have received a bogus DNS response or a forged self-signed certificate, by checking that the information they received agrees with that seen by other clients accessing the same Web site [64, 81]. Collaboration is also useful to identify popular Web content

by having Web users—or proxies monitoring traffic for an entire organization—combine their access logs to determine the most frequently accessed URLs [3]. In this chapter, we present the design, implementation, and evaluation of an efficient, privacy-preserving system that supports these kinds of data-analysis operations.

Today, these kinds of distributed data-analysis applications lack privacy protections. Existing solutions often rely on a trusted (typically centralized) aggregation node that collects and analyzes the raw data, thereby learning both the identity and inputs of participants. There is good reason to believe this inhibits participation. ISPs and Web sites are notoriously unwilling to share operational data with one another, because they are business competitors and are concerned about compromising the privacy of their customers. Many users are understandably unwilling to install software from Web analytics services such as Alexa [3], as such software would otherwise track and report every Web site they visit. Unfortunately, even good intentions do not necessarily translate to good security and privacy protections, only too-well demonstrated by the fact that large-scale data breaches have become commonplace [65]. As such, we believe that many useful distributed data-analysis applications will not gain serious traction unless privacy can be ensured.

Fortunately, many of these collaborative data-analysis applications have a common pattern, such as computing set intersection, finding so-called *icebergs* (items with a frequency count above a certain threshold), or identifying items that in aggregate satisfy some other statistical property. We refer to this problem as *privacy-preserving data aggregation* (PDA). Namely, each participant p_j has an input set of key-value tuples, $\langle k_i, v_{i,j} \rangle$, and the protocol outputs a key k_i if and only if some evaluation function $f(\forall j | v_{i,j})$ is satisfied. For example, the botnet anomaly-detection application is an instance of the iceberg problem where the goal is to detect keys that

occur more than some threshold τ times across the n parties. In this scenario, the keys k_i refer to IP addresses, each value $v_{i,j}$ is 1, and f is defined to be $\sum_{j=1}^{n} v_{i,j} \geq \tau$ (implemented, in fact, as simply keeping a running sum per key). In other words, such a protocol performs the equivalent of a database join (union) across each participant's input (multi)set, and outputs those IP addresses that appear more than τ times. In our system, keys can either be arbitrary-length bitstrings or can also be drawn from a limited domain (*e.g.*, the set of valid IP addresses). However, we restrict our consideration of values to those drawn from a more restricted domain—such as an alphanumeric score from 1 to 10 or A to F—a limitation for privacy reasons we explain later. This f could as easily perform other types of frequency analysis on keys, such as median, mode, or dynamically setting the threshold τ based on the set of inputs—for example, if there exists some appropriate "gap" between popular and unpopular inputs—as opposed to requiring τ be set *a priori* and independent of the inputs.

Informally, PDA should provide two privacy properties: (1) *Keyword privacy* requires that no party should learn anything about k_i if its corresponding values do not satisfy f. (2) *Participant privacy* requires that no party should learn which key inputs (whether or not the key remains somehow blinded prior to satisfying f) belongs to which participant. In our example of collaborating DoS victims, keyword privacy means nobody learns the identity of good IP addresses or which Web sites they frequent, and participant privacy means a Web site need not worry that its mix of clients would be revealed. In our example of collaborating Web clients, the privacy guarantees mean that a Web user need not worry that other users know what Web sites he accesses, or whether he received a bogus DNS response or a forged certificate. We believe these privacy properties would be sufficient to

encourage participants to collaborate to their mutual benefit, without concern that their privacy (or the privacy of their clients) would be compromised. Our goal, then, is to design a system that provably guarantees these properties, yet is efficient enough to be used in practice.

Ideally, we would like a system that can handle hundreds or thousands of participants generating thousands of key-value tuples. Unfortunately, fully-distributed solutions do not scale well enough, and fully-centralized solutions do not meet our privacy requirements. Simple techniques like hashing input keys [27, 4], while efficient, cannot ensure keyword and participant privacy. In contrast, the secure multi-party computation protocols from the cryptographic literature [85, 23, 57, 47, 26, 25, 42, 50, 7] would allow us to achieve our security goals, but are not practical at the scale we have in mind. [82] has a similar intent to our work, but provides much weaker privacy properties (*e.g.*, keys are known by the system) and was not evaluated in a distributed setting. Finally, few of these systems have ever been implemented [50, 28, 7], let alone operate in the real world [10] and at scale. So, a meta-goal of our work is to help bring multi-party computation to life.

In this chapter, we *design, implement, and evaluate* a viable alternative: a "semi-centralized" system architecture, and associated cryptographic protocols, that provides privacy-preserving data aggregation without sacrificing efficiency. Rather than having a single aggregator node, the data analysis is split between two separate parties—a *proxy* and a *database*. The proxy plays the role of obliviously blinding client inputs, as well as transmitting blinded inputs to the database. The database, on the other hand, builds a table that is indexed by the blinded key. For each row of this table whose values satisfy f, the database shares this row with the proxy, who unblinds the key. The database subsequently publishes its non-blinded data

for that key.

The resulting semi-centralized system provides strong privacy guarantees *provided that the proxy and the database do not collude*. In practice, we imagine that these two components will be managed either by the participants themselves that do not wish to see their own information leaked to others, perhaps even on a rotating basis, or even third-party commercial or non-profit entities tasked with providing such functionality. For example, in the case of cooperative DoS detection, ISPs like AT&T and Sprint could jointly provide the service. Or, perhaps even better, it could be offered by third-party entities like Google (which already plays a role in bot and malware detection [32]) or the EFF (which has funded anonymity tools such as Tor [17]), who have no incentive to collude. Such a separation of trust appears in several cryptographic protocols [12], and even in some natural real-world scenarios, such as Democrats and Republicans jointly comprising election boards in the U.S. political system. It should be emphasized that the proxy and database are not treated as *trusted parties*: we only assume that they will not collude. Indeed, jumping ahead, our protocol does not reveal sensitive information to either party.

Using a semi-centralized architecture greatly reduces operational complexity and simplifies the liveness assumptions of the system. For example, clients can asynchronously provide their key-value tuples without our system requiring any complex scheduling. Despite these simplifications, the cryptographic protocols necessary to provide strong privacy guarantees are still non-trivial. Specifically, our solution makes use of oblivious pseudorandom functions [59, 25, 33], amortized oblivious transfer [58, 36], and homomorphic encryption with re-randomization.

We formally prove that our system guarantees keyword and participant privacy. We first show a protocol that is robust in the *honest-but-curious* model (where,

informally, each party can perform local computation on its own view in an attempt to break privacy, but still faithfully follows the protocol). Then, we show how, with a few modifications to our original protocol, to defend against *any coalition of malicious participants*. In addition, the protocols are robust in the face of collusion between either proxy/database and any number of participants.

The remainder of the chapter is organized as follows. Section 4.2 defines our system goals and discusses why prior techniques are not sufficient. Section 4.3 describes our PDA protocols and sketches the proofs of their privacy guarantees. Section 4.4 describes our implementation, and Section 4.5 evaluates its performance. We conclude the chapter in Section 4.7.

4.2 Design Goals and Status Quo

This section defines our goals for practical, large-scale privacy-preserving data aggregation (PDA), and we discuss how prior proposals failed to meet these requirements. We then expand on our security assumptions and privacy definitions.

4.2.1 Design Goals

In the privacy-preserving data aggregation (PDA) problem, a collection of participants (or *clients*) may autonomously make observations about *values* (v_i) associated with *keys* (k_i). These observations may be, for example, the fact that an IP address is suspected to have performed some type of attack (through DoS, spam, phishing, and so forth), or the number of participants that associate a particular credential with a server. The system jointly computes a two-column input table T. The first column of T is a set comprised of all unique keys belonging to all participants (the

key column). The second column is comprised of a value $\mathsf{T}[k_i]$ that is the aggregation or union of all participant's values for k_i (the *value column*). The system then defines a particular function f to be evaluated over each row's value(s). For simplicity, we focus our discussion on the simple problem of over-threshold set intersection for f: If clients' inputs of the form $\langle k_i, 1\rangle$ are aggregated as $\mathsf{T}[k_i] \leftarrow \mathsf{T}[k_i] + 1$, is $\mathsf{T}[k_i] \geq \tau$?

A practical PDA system should provide the following:

- ***Keyword privacy:*** We say a system satisfies *keyword privacy* if, given the above aggregated table T, at the conclusion of the protocol all involved parties learn only the keys k_i whose corresponding aggregate value $\mathsf{T}[k_i] \geq \tau$. In addition, we might also have parties learn the values $\mathsf{T}[k_i]$, *i.e.*, the entire value column of T, even if the corresponding keys remain unknown. We discuss later why we may reveal the keyless value column (a histogram of frequencies in the over-threshold set intersection example) in addition to those over-threshold keys.

- ***Participant privacy:*** We say a system satisfies *participant privacy* if, at the conclusion of the protocol, nobody can learn the inputs $\{\langle k_i, v_{i,j}\rangle\}$ of participant p_j other than p_j himself (except for information which is trivially deduced from the output of the function). This is formally captured by showing that the protocol leaks no more information than an ideal implementation that uses a trusted third party. This convention is standard in secure multi-party computation; further details can be found in [30].

- ***Efficiency:*** The system should scale to large numbers of participants, each generating and inputting large numbers of observations (key-value tuples).

The system should be scalable both in terms of the network bandwidth consumed (communication complexity) and the computational resources needed to execute the PDA (computational complexity).

- ***Flexibility:*** There are a variety of computations one might wish to perform over each key's values $\mathsf{T}[k_i]$, other than the sum-over-threshold test. These may include finding the maximum value for a given key, or checking if the median of a row exceeds a threshold. Rather than design a new protocol for each function f, we prefer to have a single protocol that works for a wide range of functions.

- ***Lack of coordination:*** Finally, the system should operate without requiring that all participants coordinate their efforts to jointly execute some protocol at the same time, or even all be online around the same time. Furthermore, no set of participants should be able to prevent others from executing the protocol and computing their own results (*i.e.*, a liveness property).

As we discuss next, existing approaches fail to satisfy one or more of these goals.

4.2.2 Limitations of Existing Approaches

Having defined these five goals for PDA, we next consider several possible solutions from the literature. We see that prior secure multi-party computation protocols achieve strong privacy at the cost of efficiency, flexibility, or ease of coordination. On the other hand, simple hashing or network-layer anonymization approaches fail to satisfy our privacy requirements. Our protocol, which leverages insights from both approaches, combines the best of both worlds. Table 4.1 summarizes the discussion in this section.

Approach	Keyword Privacy	Participant Privacy	Efficiency	Flexibility	Lack of Coordination
Private Set Intersection	**Yes**	**Yes**	Poor	No	No
Garbled-Circuit Evaluation	**Yes**	**Yes**	Very Poor	**Yes**	No
Hashing Inputs	No	No	**Very Good**	**Yes**	**Yes**
Network Anonymization	No	**Yes**	**Very Good**	**Yes**	**Yes**
This paper	**Yes**	**Yes**	**Good**	**Yes**	**Yes**

Table 4.1: Comparison of proposed schemes for privacy-preserving data aggregation

Set-Intersection Protocols. Freedman *et al.* [26] proposed a specially-designed secure multi-party computation protocol to compute set intersection between the input lists of two parties. It represented each party's inputs as the roots of an encrypted polynomial, and then had the other party evaluate this encrypted polynomial on each of its own inputs. While asymptotically optimized for this setting, a careful protocol implementation found two sets of 100 items each took 213 seconds to execute (on a 3 GHz Intel machine) [28]. Kissner and Song [42] extended and further improved this polynomial-based protocol for a multi-party decentralized setting, yet their computational complexity remains $O(n\ell^2)$ and communication complexity is $O(n^2\ell)$, where n is the number of participants and ℓ is the number of input elements per party. Furthermore, after a number of pairwise interactions between participants, the system needed to coordinate a group decryption protocol between all parties. Hence, this prior work on set-intersection faces scaling challenges on large sets of inputs or participants, and it also requires new protocol design for each small variant of the set-intersection or threshold set-intersection protocol.

Secure Multi-Party Computations using Garbled Circuits. In 1982, Yao [85] proposed a general technique for computing any two-party computation privately, by building a "garbled circuit" in which one party encodes the function to be executed and his own input, and the other party obliviously evaluates her inputs on this

circuit. Very recently, the Fairplay system [50, 7] provided a high-level programming language for automatically compiling specified functions down into garbled circuits and generating network protocol handlers to execute them. While such a system would provide the privacy properties we require and offer the flexibility that hand-crafted set-intersection protocols lack, this comes at a cost. These protocols are even more expensive in both computation and communication, requiring careful coordination as well.

Hashing Inputs. Rather than building fully decentralized protocols—with the coordination complexity and quadratic overhead (in n) this entails—we could aggregate data and compute results using a centralized server. One approach is to simply have clients first hash their keys before submitting them to the server (*e.g.*, using SHA-256), so that a server only sees $H(k_i)$, not k_i itself [4]. While it may be difficult to find a pre-image of a hash function, brute force attacks are still always possible: In our collaborating intrusion detection application, for instance, a server can simply compute the hash values of all four billion IP addresses and build a simple lookup table. Thus, while certainly efficient, this approach fails to achieve either of our privacy properties. An alternative that prevents such a brute-force attack would be for all participants (clients) to coordinate and jointly agree on some secret key s, then use instead a *keyed* pseudorandom function on the input key, *i.e.*, $F_s(k_i)$. This would satisfy keyword privacy, until a single client decides to share s with the server, a brittle condition for sure.

Network Anonymization through Proxying. In the previous proposal, the server received inputs directly from clients. Thus, the server was always able to associate a row of the database with a particular client, whether or not its key is known.

One solution would be to simply proxy a client's request through one or more intermediate proxies that hides the client's identity (*e.g.*, its own IP address), as done in onion routing systems such as Tor [17]. Of course, this solution still does not achieve keyword privacy.

Although the prior approaches have their limitations, they also offer important insights that inform our design. First, a more centralized aggregation architecture avoids distributed coordination and communication overhead. Second, proxying can add participant privacy when interacting with a server. And third, a keyed pseudorandom function (PRF) can provide keyword privacy. Now, the final insight to our design is, *rather than have all participants jointly agree on the PRF secret s, let it be chosen by and remain known only to the proxy.* After all, the proxy is already trusted not to expose a client's identity to the server (database), so let's trust it not to expose this secret s to the database as well. Thus, prior to proxying (roughly) the tuple $\langle F_s(k_i), v_i \rangle$, the proxy executes a protocol with a client to *blind* its input key k_i with F_s. This blinding occurs in such a way that the client does not learn s and the proxy does not learn k_i.[1] This completes the loop, having a proxy play a role in providing both keyword and participant privacy, while the database offers flexibility in any computation over a key's values $\mathsf{T}[k_i]$ and scalability through traditional replication and data-partitioning techniques (*e.g.*, consistent hashing [40]).

4.2.3 Security Assumptions and Definitions

We now motivate and clarify some design decisions related to our security assumptions and privacy definitions. Roughly speaking, our final protocol defends against

[1] We note that oblivious pseudorandom function evaluation had been previously used in the set intersection context in [25] and [33].

malicious participants and non-colluding *honest-but-curious* databases and proxies.

Honest-but-curious parties. In our model, both proxy and database are expected to act as *honest-but-curious* (also called *semi-honest*) participants. That is, each party can perform local computation on its own view in an attempt to break privacy, but is assumed to still faithfully follow the protocol when interacting with other parties. We believe this model is very appropriate for our semi-centralized system architecture. In many deployments, the database and proxy may be well-known and trusted to act on their good intentions to the best of their abilities, as opposed to simply another participant amongst a set of mutually distrustful parties. Thus, other than fully compromising a server-side component and secretly replacing it with an actively malicious instance, data breaches are not possible in this model, as participants never see privacy-comprising data in the first place. In addition, the honest-but-curious model is one of the two standard security models in multi-party computation protocols—the other being the (obviously stronger) assumption of full malicious behavior. Unfortunately, security against fully malicious behavior comes at a great cost, as each party needs to prove at each step of the protocol that it is faithfully obeying it. For example, the proxy would need to prove that it does not omit any submitted inputs while proxying, nor falsely open blinded keys at the end of the protocol; the database would need to prove that it faithfully aggregates submitted values, and doesn't omit any rows in T that satisfy f. These proofs, typically done in zero-knowledge, greatly complicate the protocol and impact efficiency.

We will, however, present a protocol that is robust against any coalition of *malicious participants*. After all, the same trust assumptions that hold for the proxy and database does not extend to the potentially large number of participants.

Security against coalitions. Another important aspect of security is the ability to preserve privacy even when several adversarial players try to break security by sharing the information they gained during the protocol. In this aspect, we insist on providing security against any coalition of an arbitrary number of participants together with the database. This is essential as otherwise the database can perform a Sybil attack [18], *i.e.*, create many dummy participants and use their views, together with his own view, to reveal sensitive information. Similarly, we require security against any coalition of the proxy and the participants. On the other hand, in order to have an efficient and scalable system, we are willing to tolerate vulnerability against a coalition of the database and the proxy, which could otherwise break participant and keyword privacy.

Releasing the value column. Our protocol releases those keys whose values satisfy f, but the database also learns the entire value column ($\mathsf{T}[k_i], \forall i$), even though it learns no additional information about the corresponding k_i's. This asymmetric design was chosen as revealing all $\mathsf{T}[k_i]$ may be seen as a privacy violation.

That said, in other settings it may be acceptable to release the entire value column, so that all parties see identical information. This also serve another practical purpose, as it may be hard to fully specify f *a priori* to collecting clients' inputs. For example, how should an anomaly detection system choose the appropriate frequency threshold τ? In some attacks, 10 observations about a particular IP address may be high (*e.g.*, suspected phishing), while in others, 1000 observations may be necessary (*e.g.*, for bots participating in multiple DoS attacks). Furthermore, a dataset may naturally expose a clear gap between frequency counts of normal and anomalous behavior; the very reason data operators like to "play" with raw data in the first place.

We also note that the acceptable set of input values and the system's security assumptions has some bearing here. If the domain $\mathcal{D}$ of possible values is large, a client can try to "mark" a key k by submitting it together with an uncommon value $w \in \mathcal{D}$. If a value column that somehow includes w is revealed, the client can discover other clients' values for that same key. That said, a similar problem exists when the value column is not released and one is concerned about collusions between a client and database (who can search for the $\mathsf{T}[k]$ that includes w). This problem does not arise when the domain is relatively small (*e.g.*, when values are grades over some limited scale).

We mention that this asymmetry and/or security issue can be completely eliminated by first having participants encrypt their values under the public keys of both proxy and database, and by then using additional cryptographic protocols for the aggregation of the values. While these tools are relatively expensive, the structure of our system allows us to employ them only for the two-party case (for the proxy and database) which results in a significant efficiency improvement over other more distributed solutions.

4.3 Our PDA Protocol

In this section, we describe our protocol and analyze its security. Section 4.3.1 describes a simplified version of the protocol that achieves somewhat weaker security properties. This version will be extended to support a stronger notion of security in Section 4.3.2. Our protocol employs several standard cryptographic tools (*e.g.*, public-key encryption schemes, pseudorandom functions, and the oblivious evaluation of a pseudorandom function). We will elaborate on these tools and suggest

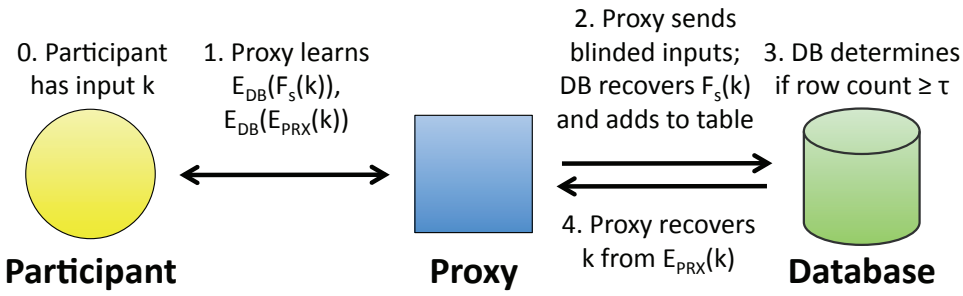

Figure 4.1: High-level system architecture and protocol. F_s is a keyed hash function whose secret key s is known only to the proxy.

concrete instantiations in Section 4.3.3. More details about the extended protocol and sketches of formal security proofs are given in the Appendix.

4.3.1 The Basic Protocol

Our protocol consists of four basic steps (see Figure 4.1). In the first two steps, the proxy interacts with the participants to collect the blinded keys together with their associated values encrypted under the database's public-key, and then passes these encrypted values on to the database. Then, in the third step, the DB aggregates the blinded keys together with the associated values in a table and decides which rows should be revealed according to a predefined function f. Finally, the DB asks the proxy to unblind the corresponding keys. Since the blinding scheme F_s is not necessarily invertible, the revealing mechanism uses some additional information that is sent during the first phase.

- **Parties**: Participants, Proxy, Database.
- **Cryptographic Primitives**: A pseudorandom function F, where $F_s(k_i)$ denotes the value of the function on the input k_i with a key s. A public-key encryption E, where $E_{\mathrm{K}}(x)$ denotes an encryption of x under the public key

K.

- **Public Inputs**: The proxy's public key PRX, the database's public key DB.
- **Private Inputs.** *Participant:* A list of key-value pairs $\langle k_i, v_i \rangle$. *Proxy:* key s of PRF F and secret key for PRX; *Database:* secret key for DB.

1. Each participant interacts with the proxy as follows. For each entry $\langle k_i, v_i \rangle$ in the participant's list, the participant and the proxy run a sub-protocol for oblivious evaluation of the PRF (OPRF). At the end of this protocol, the proxy learns nothing and the participant learns only the value $F_s(k_i)$ (and nothing else, not even the key s of the PRF). The participant computes the values $E_{\mathrm{DB}}(F_s(k_i))$, $E_{\mathrm{DB}}(v_i)$, and $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$, and it sends them to the proxy. (The last entry will be used during the revealing phase.) The proxy adds this triple to a list and waits until most/all participants send their inputs.

2. The proxy randomly permutes the list of triples and sends the result to the DB. The DB uses its private key to decrypt all the entries of each triple. Now, it holds a list of triples of the form $\left\langle F_s(k_i), v_i, E_{\mathrm{PRX}}(k_i) \right\rangle$. The DB inserts these values into a table which is indexed by the (blinded) key $F_s(k_i)$. At the end, the DB has a table of entries of the form $\left\langle F_s(k_i), \mathsf{T}[k_i], \mathsf{E}[k_i] \right\rangle$, where $\mathsf{T}[k_i]$ is (in general) a list of all the v_i's that appeared with this key (or simply the number of times a client inputted k_i in the case of threshold set intersection), and $\mathsf{E}[k_i]$ is a list of values of the form $E_{\mathrm{PRX}}(k)$.

3. The DB uses some predefined function f to partition the table into two parts: R, which consists of the rows whose keys should be revealed, and H, which consists of the rows whose keys should remain hidden. Then, it sends all the rows of R to the proxy.

4. The proxy goes over the received table R and replaces all the encrypted $E_{\mathrm{PRX}}(k_i)$ entries with their decrypted key k_i. Then it publishes the updated table.

Variants. One may consider several variants in which different information is released to the participants by the database. For example, it is possible to release only the *keys* k_i which are chosen by the function f without the corresponding values $\mathsf{T}[k_i]$. On the other extreme, the DB can release more data by publishing the pairs $(k_i, \mathsf{T}[k_i])$ for the k_i's that are selected by f, together with the values $\mathsf{T}[k_i]$ of the keys that were not selected by f without the corresponding keys (*i.e.*, the entries $\mathsf{T}[k_i]$ of the table H). This might be useful to the participants and, in some scenarios, the additional information might not constitute a privacy violation (in the "real-world" sense). Consider, for example, the case where the values are always one, *i.e.*, the participants only want to increment a counter for some key. In this case, the table R simply consists of keys and their frequencies, and H is simply a frequency table of all the unrevealed keys.

Security Guarantees. We claim that this protocol guarantees privacy against the following attacks:

Coalition of honest-but-curious participants. Consider the view of an honest-but-curious participant during the protocol. Due to the security of the OPRF sub-protocol, a single participant sees only a list of pseudorandom values of the form $F_s(k_i)$, and therefore it learns nothing beyond the output of the protocol (formally, this view can be easily simulated by using truly random values). The same holds for a coalition of participants.

In fact, this protocol achieves a reasonable level of security against malicious participants as well. Recall that the interaction of the proxy with a participant is completely *independent* of the inputs of other participants. Hence, even if the participants are malicious, they still learn nothing about the data of other honest participants. Furthermore, even malicious participants will be forced to choose their inputs *independently* of the inputs of other honest participants. For example, they cannot duplicate the input of some other honest participant. (Similar security definitions were also considered in [58, 33].) However, malicious participants can still violate the *correctness* of the above protocol. This issue will be fixed in the extended protocol.

Honest-but-curious proxy. The proxy's view consists of three parts: (1) the view during the execution of the OPRF protocol—this gives no information due to the security of the OPRF; (2) the tuples that the participants send—these values are encrypted under the DB's key and therefore reveal no information to the proxy; and (3) the values that the DB sends during the last stage of the protocol—these are just key-value pairs (encrypted under the proxy's key) that should be revealed anyway, and thus they give no additional information beyond the actual output of the protocol.

Honest-but-curious coalition of proxy and participants. The above argument generalizes to the case where the proxy colludes with honest-but-curious participants. Indeed, the joint view of such coalition reveals nothing about the inputs of the honest participants.

Honest-but-curious database. The DB sees a blinded list of keys encrypted under his public key DB, without being able to relate the blinded entries to their owners.

For each blinded key $F_s(k_i)$, the DB also sees the list of its associated values $\mathsf{T}[k_i]$ and encryptions of the keys under the proxy's key $E_{\mathrm{PRX}}(k)$. Finally, the DB also sees the output of the protocol. The values $F_s(k_i)$ and $E_{\mathrm{PRX}}(k)$ bear no information due to the security of the PRF and the encryption scheme. Hence, the DB learns nothing but the value table of the inputs (*i.e.*, the $\mathsf{T}[k_i]$'s for all k_i's).[2]

4.3.2 The Full-Fledged Protocol

In the following, we describe how to immunize the basic protocol against stronger attacks.

Honest-but-curious coalition of participants and database. A careful examination of the previous protocol shows that it is vulnerable against such coalitions for two main reasons.

First, a participant knows the blinded version $F_s(k_i)$ of its own keys k_i, and, in addition, the DB can associate all the values $\mathsf{T}[k_i]$ to their blinded keys $F_s(k_i)$. Hence, a coalition of a participant and a DB can retrieve all the values $\mathsf{T}[k_i]$ that are associated with a key k_i that the participant holds, even if this key *should not be revealed* according to f. To fix this problem, we modify the first step of the protocol. Instead of using an OPRF protocol, we will use a different sub-protocol in which the participant learns nothing and the proxy learns the value $E_{\mathrm{DB}}(F_s(k_i))$ for each k_i. This solves the problem as now that participant himself does not know the blinded version of his own keys. To the best of our knowledge, this version of encrypted-OPRF protocol (abbreviated EOPRF) has not appeared in the literature before. Fortunately, we are able to construct such a protocol by carefully modifying

[2] Formally, we define a functionality in which this additional information is given to the database as part of its output. See the appendix for details.

the OPRF construction of [25] and combining it with El-Gamal encryption (see Section 4.3.3).

Second, we should eliminate subliminal channels, as these can be used by participants and the database to match the keys of a participant to their blinded versions (that were forwarded to the DB by the proxy). Indeed, public-key encryption schemes use randomness (in addition to the public key) to encrypt a message, and this randomness can be used as a subliminal channel. To solve this problem, we use an encryption scheme that supports re-randomization of ciphertexts; that is, given an encryption of x with randomness b, it should be possible to recompute an encryption of y under fresh randomness b' (without knowing the private key). Now we eliminate the subliminal channel by asking the proxy to re-randomize the ciphertexts—$E_{\mathrm{DB}}(F_s(k_i))$, $E_{\mathrm{DB}}(v_i)$, and $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$—which are encrypted under the DB's public key (at Step 1). Furthermore, we should be able to re-randomize the *internal* ciphertext $E_{\mathrm{PRX}}(k_i)$ of the last entry as well (we will show that this can be achieved through variant of El-Gamal encryption).

A coalition of malicious participants. As we already observed, malicious participants can violate the correctness of our protocol. Specifically, they might try to submit ill-formed inputs. Recall that the participant sends to the proxy triples $\langle a, b, c\rangle$, where in an honest execution we have $a = E_{\mathrm{DB}}(F_s(k_i)), b = E_{\mathrm{DB}}(v_i)$ and $c = E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$ for some k_i and v_i. However, a cheating participant might provide an inconsistent tuple, in which $a = E_{\mathrm{DB}}(F_s(k_i))$ while $c = E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k'_i))$ for some $k'_i \neq k_i$. We can prevent such an attack by asking the proxy to apply a consistency check to R in the last step of the protocol and to make sure that $E_{\mathrm{PRX}}(k'_i)$ and $F_s(k_i)$ match. The proxy omits all the inconstant values (if there are any) and asks the DB to check again if the corresponding row should be revealed

after the omission. (This modification suffices as long as the function f is local, *i.e.*, it is applied to each row separately. See appendix for more details.)

Another thing that a cheating participant might do is to replace b with some "garbage" value $b' = E_{\mathrm{DB}}(v')$ for which he does not know the plaintext v' (while this might not seem to be beneficial in practice, we must prevent such an attack in order to meet our strong definitions of security). To prevent such attack, we ask the participant to provide a zero-knowledge proof that shows that he knows the plaintext v to which that b decrypts. As seen in the next section, this does not add too much overhead.

Finally, our sub-protocol for the EOPRF should be secure against malicious participants in the following sense: a malicious participant should not be able to generate a blinded value $E_{\mathrm{DB}}(F_s(k_i))$ for a key k_i that he does not know.

In the appendix, we show that our modifications guarantee full security against malicious participants.

4.3.3 Concrete Instantiation of the Cryptographic Primitives

In the following section, we assume that the input keys are represented by m-bit strings. We assume that m is not very large (*e.g.*, less than 192–256); otherwise, one can hash the input keys and apply the protocol to resulting hashed values.

Public Parameters. Our implementation mostly employs Discrete-Log based schemes. In the following, g is a generator of a multiplicative group $\mathbb{G}$ of prime order p for which the decisional Diffie-Hellman (DDH) assumption holds. We publish (g, p) during initialization and assume the existence of algorithms for multiplication (and thus also for exponentiation) in $\mathbb{G}$. We let $\mathbb{Z}_p$ denote the field of integers mod-

ulo p, the set $\{0, 1, \ldots, p-1\}$ with multiplication and addition modulo p. We will let $\mathbb{Z}_p^*$ denote the multiplicative group of the invertible elements $\mathbb{Z}_p$.

El-Gamal Encryption. We will use El-Gamal encryption over the group $\mathbb{G}$. The private key is a random element a from $\mathbb{Z}_p^*$, and the public key is the pair $(g, h = g^a)$. To encrypt a message $x \in \mathbb{G}$, we choose a random b from $\mathbb{Z}_p^*$ and compute $(g^b, x \cdot h^b)$. To decrypt the ciphertext (A, B), compute $B/A^a = B \cdot A^{-a}$ (where $-a$ is the additive inverse of a in $\mathbb{Z}_p$). In case we wish to "double-encrypt" a message $x \in \mathbb{G}$ under two different public-keys (g, h) and (g, h'), we will choose a random b from $\mathbb{Z}_p^*$ and compute $(g^b, x \cdot (h \cdot h')^b)$. This ciphertext as well as standard ciphertexts can be re-randomized by multiplying the first entry (resp. second entry) by $g^{b'}$ (resp. $h^{b'}$) where b' is chosen randomly from $\mathbb{Z}_p^*$. Finally, a zero-knowledge proof for knowing the decryption of a given ciphertext is described in [71]. The scheme adds only 3 exponentiations and does not increase the overall round complexity as it can be applied in parallel to the EOPRF protocol.

Naor-Reingold PRF [59]. The key s of the function $F_s : \{0,1\}^m \to \mathbb{G}$ contains m values $(s_1, \ldots, s_m)$ chosen randomly from $\mathbb{Z}_p^*$. Given m-bit string $k = x_1 \ldots x_m$, the value of $F_s(k)$ is $g^{\prod_{x_i=1} s_i}$, where the exponentiation is computed in the group $\mathbb{G}$.

Oblivious-Transfer [66, 58]. To implement the sub protocol of Step 1, we will need an additional cryptographic tool called Oblivious Transfer (OT). In an OT protocol, we have two parties: sender and receiver. The sender holds two strings (α, β), and the receiver has a selection bit c. At the end of the protocol, the receiver learns a *single* string: α if $c = 0$, and β if $c = 1$. The sender learns nothing (in particular, it does not know the value of the selector c).

The Encrypted-OPRF protocol

Our construction is inspired by a protocol for oblivious evaluation of the PRF F, which is explicit in [25] and implicit in [57, 58]. We believe that this construction might have further applications.

- **Parties**: Participant, Proxy.
- **Inputs.** *Participant:* m-bit string $k = (x_1 \dots x_m)$; *Proxy:* secret key $s = (s_1, \dots, s_m)$ of a Naor-Reingold PRF F.

1. Proxy chooses m random values $u_1, \dots, u_m$ from $\mathbb{Z}_p^*$ and an additional random $r \in \mathbb{Z}_p^*$. Then for each $1 \leq i \leq m$, the proxy and the participant invoke the OT protocol where proxy is the sender with inputs $(u_i, s_i \cdot u_i)$ and receiver uses x_i as his selector bit. That is, if $x_i = 0$, the participant learns u_i and otherwise it learns $s_i \cdot u_i$. The proxy also sends the value $\hat{g} = g^{r/\Pi u_i}$. (These steps can be done in parallel.)

2. The participant multiplies together the values received in the OT stage. Let M denote this value. Then, it computes $\hat{g}^M = (g^{\Pi_{x_i=1} s_i})^r = F_s(k)^r$. Finally, the participant chooses a random element a from $\mathbb{Z}_p^*$ and encrypts $F_s(k)^r$ under the public key $\text{DB} = (g, h)$ of the database. The participant sends the result $(g^a, F_s(k)^r \cdot h^a)$ to the proxy.

3. The proxy raises the received pair to the power of r', where r' is the multiplicative inverse of r modulo p. It also re-randomizes the resulting ciphertext.

Correctness. Recall that $\mathbb{G}$ has a prime order p. Hence, when the pair $(g^a, F_s(x)^r \cdot h^a)$ is raised to the power of $r' = r^{-1}$, the result is $(g^{ar'}, F_s(k) \cdot h^{ar'})$, which is exactly $E_{\text{DB}}(F_s(k))$. Thus, the protocol is correct.

Privacy. All the proxy sees is the random tuple $(u_1, \ldots, u_m, r)$ and $E_{\mathrm{DB}}(F_s(k)^r)$. This view gives no additional information except of $E_{\mathrm{DB}}(F_s(k))$. (Formally, the view can be perfectly simulated given $E_{\mathrm{DB}}(F_s(k))$.) On the other hand, we claim that all the participant sees is a sequence of random values and therefore it also learns nothing. Indeed, the participant sees the vector $(s_1^{x_1} \cdot u_1, \ldots, s_m^{x_m} \cdot u_m)$, whose entries are randomly distributed over $\mathbb{G}$, as well as the value $\hat{g} = (g^{1/\Pi u_i})^r$. Since r is randomly and independently chosen from $\mathbb{Z}_p^*$, and since $\mathbb{G}$ has a prime order p, the element $\hat{g}$ is also uniformly and independently distributed over $\mathbb{G}$. The protocol supports security against malicious participants (in the sense that was described earlier) as long as the underlying OT is secure against a malicious receiver.

Implementing Oblivious Transfer

In general, oblivious transfer is an expensive public-key operation (*e.g.*, it may take two exponentiations per single invocation). In the above protocol, then, we execute an OT protocol for each *bit* of the participants input k (which would result, for example, in 64 exponentiations just to input a single IP address). However, Ishai *et al.* [36] show how to reduce the amortized cost of OT to be as fast as matrix multiplication. This "batch OT" protocol uses a standard OT protocol as building block. We implemented this batch OT protocol on top of the basic OT protocol of [58].[3]

[3]The "batch OT" protocol also has a version which preserves security against a malicious receiver. This increases the number of multiplications by a multiplicative factor, but does not affect the number of expensive public-key operations.

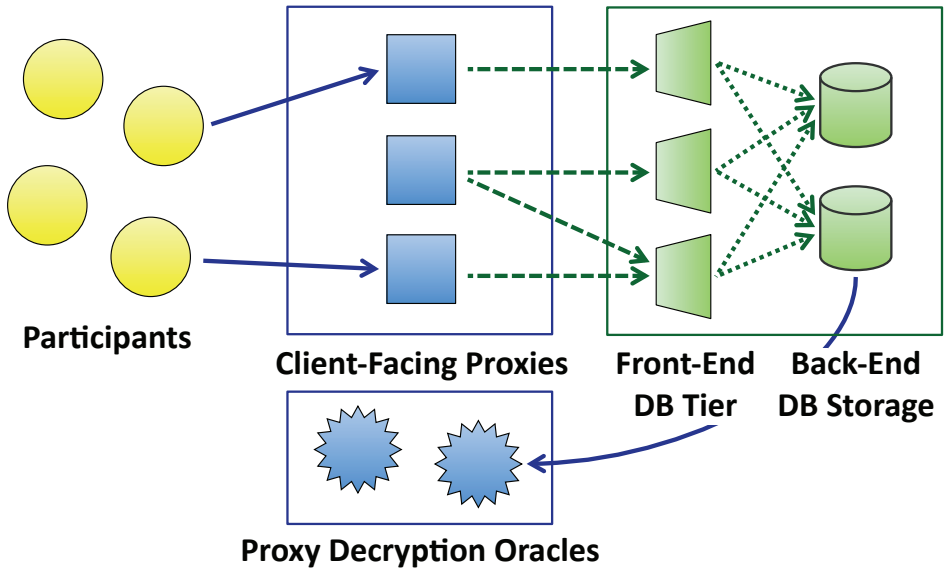

Figure 4.2: Distributed proxy and database architecture

4.3.4 Efficiency of our Protocol

In both the basic and extended protocol, the round complexity is constant, and the communication complexity is linear in the number of items. The protocol's computational complexity is dominated by cryptographic operations. For each m-bit input key, we have the following amortized complexity: (1) The participant who holds the input key computes 3 exponentiations in the basic protocol (respectively 8 in the extended protocol), as well as $O(m)$ modular multiplication / symmetric-key operations in both versions. (2) The proxy computes 5 exponentiations in the basic protocol (resp. 12 in the extended protocol) and $O(m)$ modular multiplication / symmetric-key operations. (3) The database computes 3 exponentiations in the basic protocol (resp. 5 in the extended protocol).

4.4 Distributed Implementation

In our system, both the proxy and database logical components can be physically replicated in a relatively straightforward manner. In particular, our design can scale out horizontally to handle higher loads, by increasing the number of proxy and/or database replicas, and then distributing requests across these replicas. Our distributed architecture is shown in Figure 4.2. Our current implementation covers all details described in the basic protocol, as well as some security improvements of the extended version (*e.g.*, including the EOPRF, but not ciphertext re-randomization, proofs of knowledge, or the final consistency check).

4.4.1 Proxy: Client-Facing Proxies and Decryption Oracles

One administrative domain can operate any number of proxies. Each proxy's functionality may be logically divided into two components: handling client requests, and serving as decryption oracles for the database when a particular key should be revealed. None of these proxies need to interact, other than having all client-facing proxies use the same secret s to key the pseudorandom function F and all decryption-oracle proxies use the same public/private key PRX. In fact, these two proxies play different logical roles in our system and could even be operated by two different administrative domains. In our current implementation, all proxies register with a single group membership server, although a distributed group membership service could be implemented for additional fault tolerance [11, 84].

To discover a client-facing proxy, a client contacts this group membership service, which returns a proxy IP address in round-robin order (this could be replaced by any technique for server selection, including DNS, HTTP redirection, or a local load

balancer). To submit its inputs, a client connects with this proxy and then executes an amortized Oblivious Transfer (OT) protocol on its input batch. This results in the proxy learning $\left\langle E_{\mathrm{DB}}(F_s(k_i)), E_{\mathrm{DB}}(v_i), E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i)) \right\rangle$ for each input tuple, which it pushes onto an internal queue. (While Section 4.3.3 only described the use of ElGamal encryption, its special properties are only needed for $E_{\mathrm{DB}}(F_s(k_i))$; the other public-key operations can be RSA, which we use in our implementation.) When this queue reaches a certain length (10,000 in our implementation), the proxy randomly permutes (shuffles) the items in the queue, and sends them to a database server.

The database, upon determining that a key k_i's value satisfies f, sends $E_{\mathrm{PRX}}(k_i)$ to a proxy-decryption oracle. The proxy-decryption oracle decrypts $E_{\mathrm{PRX}}(k_i)$ and returns k_i to the database for storage and subsequent release to other participants in the system.

4.4.2 Database: Front-end Decryption and Back-end Storage

The database component can also be replicated. Similar to the proxy, we separate database functionality into two parts: the *front-end* module that handles proxy submissions and decrypts inputs, and a *back-end* module that acts as a storage layer. Each logical module can be further replicated in a manner similar to the proxy.

The servers comprising the front-end database tier do not need to interact, other than being configured with the same public/private keypair DB. Thus, any front-end database can decrypt the $E_{\mathrm{DB}}(F_s(k_i))$ input supplied by a proxy, and the proxies can load balance input batches across these database servers.

The back-end database storage, on the other hand, needs to be more tightly

coordinated, as we ultimately need to aggregate all $F_s(k_i)$'s together, no matter which proxy or front-end database processed them. Thus, the back-end storage tier partitions the keyspace of all 1024-bit strings over all storage nodes (using consistent hashing [40]). All such front-end and back-end database instances also register with a group membership server, which the front-end servers contact to determine the list of back-end storage nodes. Upon decrypting an input, the front-end node determines which back-end storage node is assigned the resulting key $F_s(k_i)$, and sends the tuple $\left\langle F_s(k_i), v_i, E_{\text{PRX}}(k_i) \right\rangle$ to this storage node.

As these storage nodes each accumulate a horizontal portion of the entire table T, they test the value column for their local table to see if any keys satisfy F. For each such row, the storage node sends the tuple $\left\langle F_s(k_i), T[k_i], E_{\text{PRX}}(k_i) \right\rangle$ to a proxy-decryption oracle.

4.4.3 Prototype Implementation

Our design is implemented in roughly 5,000 lines of C++. All communication between system components—client, front-end proxy, front-end database, back-end database storage, and proxy-decryption oracle—is over TCP using BSD sockets. We use the GnuPG library for large numbers (bignums) and cryptographic primitives (*e.g.*, RSA, ElGamal, and AES). The Oblivious Transfer protocol (and its amortized variant) were implemented from scratch, and comprised a total of 625 lines of code. All RSA encryption used a 1024-bit key, and ElGamal used a corresponding 1024-bit group size. AES-256 was used in the batch OT and its underlying OT primitive. The back-end database simply stores table rows in memory, although we plan to replace this with a durable key-value store (*e.g.*, BerkeleyDB [61]).

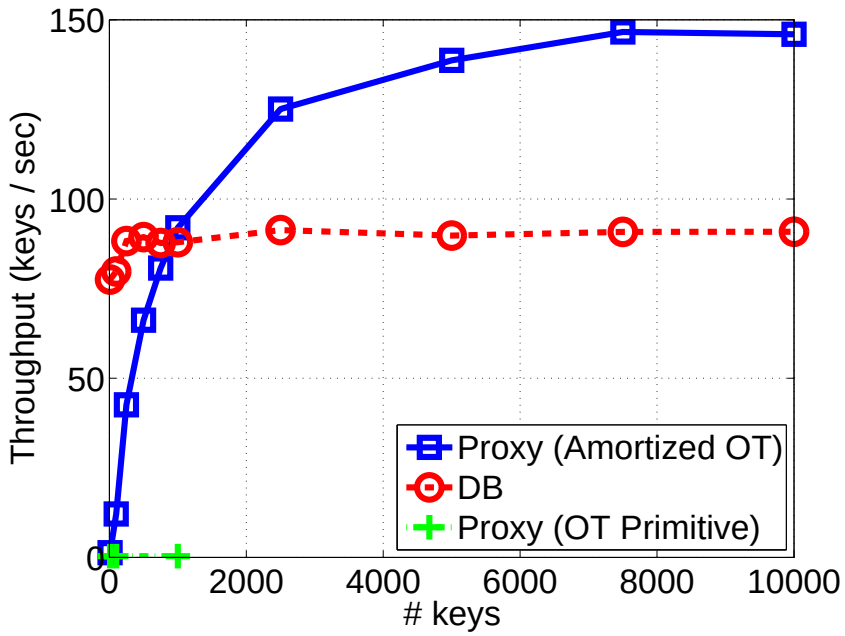

Figure 4.3: Throughput as a function of number of keys

4.5 Performance Evaluation

We wish to evaluate our system along three primary dimensions. (a) Given fixed computing resources, what is the throughput of our system as a function of the size of the input set? (b) What are the primary factors limiting throughput? And, (c) how does the throughput scale with increasing computing resources? In each case, we are concerned with both (1) how long it takes for clients to send key-value pairs to the proxy during the OT phase (*proxy throughput*) and (2) how long it takes for the DB to decrypt and identify keys with values that satisfy the function f (*DB throughput*). We have instrumented our code to measure both. For a given experiment requiring the proxy to process n keys, proxy throughput is defined as n divided by the time it takes between when the first client contacts any client-facing proxy and when the last key is processed by some client-facing proxy. Similarly,

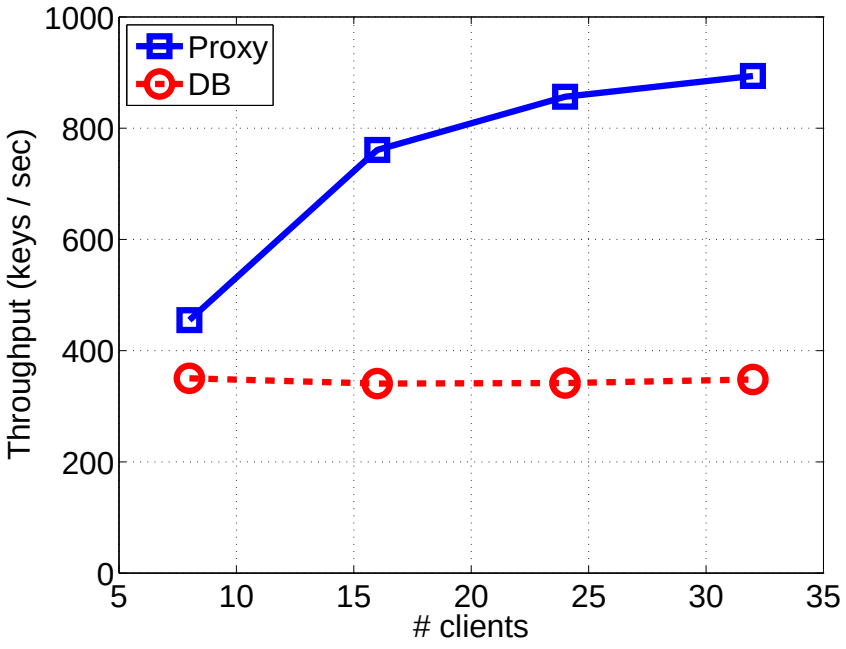

Figure 4.4: Throughput as a function of number of participants

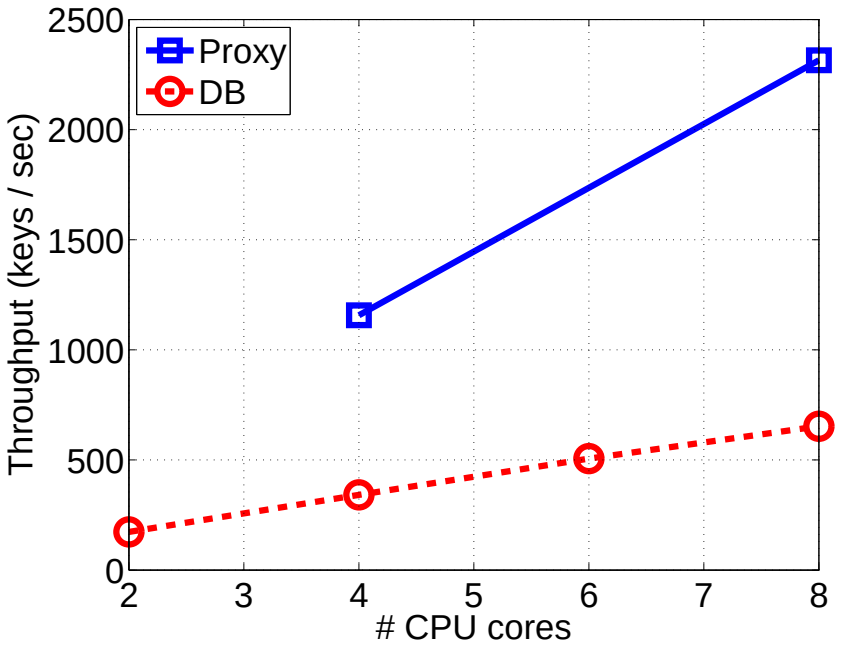

Figure 4.5: Throughput as a function of number of proxy/DB replicas

database throughput is defined as the number of keys processed between when the first client-facing proxy forwards keys to some DB front-end and when the DB back-end storage processes the last submitted keys.

Our experiments were run on multiple machines. The servers (proxy and DB) were run on HP DL160 servers (quad-core Intel Xeon 2 GHz machines with 4 GB RAM running CentOS Linux). These machines can perform a 1024-bit ElGamal encryption in 2.2 ms, ElGamal decryption in 2.5 ms, RSA encryption in 0.5 ms, and RSA decryption in 2.8 ms. Due to a lack of homogeneous servers, the clients were run on different machines depending on the experiment. The machines used for the clients were either (A) of the same configuration as the servers, or one of either (B) Sun SunFire X4100 servers with two dual-core 2.2 GHz Opteron 275 processors (four 64-bit cores) with 16GB RAM running CentOS, or (C) Dell PowerEdge 2650 servers with two 2.2 GHz Intel Xeon processors and 5 GB of memory, also running Linux.

As noted in the introduction, our system can be used in different contexts. One of the most prominent is that of anomaly detection: specifically, networks collaborating to identify attacking IP addresses—*e.g.*, belonging to a botnet—with greater confidence. Modern botnets can range up to roughly 100,000 unique hosts [67], and we would like our system to be able to correlate suspicions of hundreds of participating networks within some numbers of hours. In order to support such a usage scenario, our implementation will need to be able to process millions of keys in the span of hours or many hundreds of keys per second. We will revisit the feasibility of our implementation for our supporting applications in Section 4.5.2, but these numbers should provide rough expectations for the throughput numbers to be presented in Section 4.5.1.

4.5.1 Scaling and Bottleneck Analysis

Effect of number of keys (Figure 4.3). The input trace to our system is parameterized by the number of clients and by the number of keys they each submit. In Figure 4.3, we measure the throughput of our system as a function of the number of keys. More precisely, we run a single client, a single proxy, and a single DB in order to measure single-CPU-core proxy throughput and single-CPU-core DB throughput. The top solid curve shows proxy throughput when the proxy and client utilize the amortized OT protocol, the middle dashed curve shows DB throughput, and the bottom partial curve shows proxy throughput when the proxy and client utilize only the standard OT primitive, which does not include our amortization-based extensions. The throughput of the OT primitive is exceedingly low (less than one key per second), which is why it was not evaluated on the full range of x-values.

Proxy throughput scales well with the number of incoming keys when the client and proxy utilize the amortized OT protocol. Throughput increases with increasing numbers of keys per batch, as the amortized OT calls the primitive OT a fixed number of k times regardless of the number of input keys n. With small n (*e.g.*, up to 1000), the cost of these calls to the primitive OT dominate overall execution time and leave the proxy underutilized. However, as the size of the input set increases, the cost of encrypting keys on the client becomes the primary bottleneck, which is the plot shows minimal increase in throughput above $n = 8000$.

DB throughput, on the other hand, does not scale with the number of keys. The reason for this is that the intensive work on the DB is decryption, which is performed in batch, and it is therefore entirely CPU limited. The DB becomes CPU limited at 10 keys and remains CPU limited at 10,000 keys (*i.e.*, latency goes

up and throughput remains constant). We noted earlier that the machines on which the DB and proxy run require 2.5 ms per decryption. Since the DB has to perform 3 decryptions per key, the DB therefore has a maximum throughput of 135 keys per second on a single CPU core. Figure 4.3 shows that our DB implementation achieves throughput of roughly 90 keys per second.

The amortized OT protocol [36] introduces a trade-off between the message overhead and memory consumption. The memory footprint of this protocol per client-proxy interaction for n keys is $n \times 32 \times 2 \times 1024/8 = 8196n$ bytes (*i.e.*, we assume 32 bits per key, the 2 values for the OT primitive, 1024-bit keys, and 8 bits per byte). For $n = 10,000$ keys, for example, this requires 82 MB on both the proxy and the client. A proxy communicating with 100 clients would therefore require in excess of 8GB of memory. A user of the protocol could choose to execute the amortized OT protocol in stages, however, by sending k keys at a time, which would reduce the memory footprint. Our system is parameterized to support this, and because Figure 4.3 shows that there is little to gain from batch sizes in excess of 5,000 keys, the remainder of our experiments will use batch sizes of 5,000 keys.

Our architecture is designed to maximize throughput, not minimize latency. In fact, providing a meaningful measure of latency is challenging for multiple reasons: (a) the DB processes $f \stackrel{\text{def}}{=} \mathsf{T}[k_i] \geq \tau$ once every t seconds (*i.e.*, not upon arrival, which wouldn't make sense unless $\tau = 1$); (b) the proxy batches and randomly permutes/shuffles key-value pairs for security; and (c) the substantial benefit of the amortized OT (over the OT primitive: see again Figure 4.3) is lost if the client submits only a 1 key-value pair, which is required for a "true" latency experiment. These qualifiers notwithstanding, Figure 4.3 does provide a form of "mean" latency. That is, a single client with 5000 keys would see mean proxy latency of 7.2 millisec-

onds per key and mean DB latency of 11.1 milliseconds per key.

Effect of number of participants (Figure 4.4). Here we evaluate the throughput of our system as a function of the number of clients sending keys. In this experiment, we limit the proxy and DB to one server machine each. Four client-facing proxy processes are launched on one machine and four front-end DB processes are launched on the other. They can therefore potentially utilize all eight cores on these two machines. Figure 4.4 shows that the proxy scales well with the number of clients. Proxy throughput increases by nearly a factor of two between 8 and 32 clients. This signifies that, when communicating with a single client, a proxy spends a substantial fraction of its time idling. The four proxies in this experiment are not CPU limited until they handle 32 clients, at which time the throughput approaches 900 keys per second. The DB, on the other hand, is CPU-bound throughout. It has a throughput of about 350 keys per second, independent of the number of clients.

Effect of number of replicas (Figure 4.5). Finally, we wish to analyze how our distributed architecture scales with the available computing resources. In this experiment, we provide up to 8 cores across two machines to each of the proxy and DB front-ends. While the proxy is evaluated on 64 clients, computing resource constraints meant that the DB is evaluated on 32 clients.

Both our proxy and DB scale linearly with the number of CPU cores allocated to them. Throughput for the DB with 2 cores when handling 32 clients was over 173 keys per second, whereas at 8 cores the throughput was 651 keys per second: a factor of 3.75 increase in throughput for a factor of 4 increase in computing resources. The proxy has throughput of 1159 keys per second when utilizing 4 cores and 2319 when utilizing 8 cores: an exact factor of 2 increase in throughput for an equal increase

Global		Within amortized OT					
wait	encrypt	wait	pow	AES	arith	other	OT
60%	1%	0%	16%	4%	4%	6%	7%

Table 4.2: Breakdown of proxy resource usage

Global		Within amortized OT					
wait	encrypt	wait	pow	AES	arith	other	OT
0%	40%	31%	16%	2%	1%	3%	7%

Table 4.3: Breakdown of client resource usage

in computing resources. This clearly demonstrates that our protocol, architecture, and implementation can scale up to handle large data sets. In particular, our entire system could handle input sizes on the order of millions of keys in hours.

Micro-benchmarks. To gain a deeper understanding of the factors limiting the scalability of our design, we instrumented the code to account for how the client and proxy were spending their CPU cycles. While the DB is entirely CPU bound due only to decryptions (*i.e.*, its limitations are known), the proxy and client engage in the oblivious OT protocol whose bottlenecks are less clear. In Tables 4.2 and 4.3, we therefore show the fraction of time the client and proxy, respectively, spend performing various tasks needed for their exchange. In this experiment, we have a single client send keys to a single proxy at the maximum achievable rate.

At the highest level, we split the tasks performed into (a) waiting (called "wait"), (b) encrypting or decrypting values ("encrypt"), or (c) engaging in the amortized OT protocol. We further split work within the amortized OT protocol into time spent waiting, performing modulo exponentiations ("pow"), calling AES256, performing basic arithmetic such as multiplication, division, or finding multiplicative inverses ("arith"), calling the OT primitive ("OT"), and any other necessary tasks ("other") such as XOR'ing numbers, generating random numbers, allocating or de-

allocating memory, etc.

Table 4.2 shows that when communicating with a single client, the client-facing proxy spends more than 60% of its time idling while waiting for the client—it is *more than* 60% because some part of the 7% of time spent within the OT primitive is also idle time. The 60% idle time is primarily due to waiting for the client to encrypt k_i and $F_s(k_i)$. The single largest computational expense for the proxy is performing powmods at 16%; the remaining non-OT tasks add up to 15%. In order to make the proxy more efficient, therefore, utilizing a bignum library with faster exponentiation and basic arithmetic would be advantageous.

The client also spends a non-trivial amount of time waiting—31% of total execution time—but substantially less than the proxy. It spends 40% of its time encrypting values. The reason this 40% does not match up with the 60% idle time of the proxy is because the proxy finishes its portion of the amortized OT before the client does its portion. That is, 20 out of the proxy's 60% idle time is due to the client processing data sent by the proxy in the last stage of the amortized OT protocol, and the remaining 40 is due to the client encrypting its values. A with the proxy, the client would benefit from faster exponentiations, but encryption is clearly the major bottleneck. We noted before that the GnuPG cryptographic library we use performed public-key operations in approximately 2.5–2.8 ms. On the same servers, we benchmarked the Crypto++ library to perform RSA decryption in only 1.2 ms, increasing speed by 130%. Crypto++ would also allow us to take advantage of elliptic curve cryptography, which would increase system throughput. In future work, we plan to modify our implementation to use this library.

4.5.2 Feasibility of Supporting Applications

In this section, we revisit several potential applications of our system. We consider our results in light of their potential demands on request rate: the number of requests per unit time that must be satisfied, the number of keys which must be stored in the system, and the number of participants.

Anomaly detection. Network operators commonly run systems to detect and localize anomalous behavior within their networks. These systems dynamically track the traffic mix—*e.g.*, the volume of traffic over various links or the degree of fanout from a particular host—and detect behavior that differs substantially from the statistical norm. For example, Mao *et al.* [52] found that most DDoS attacks observed within a large ISP were sourced by fewer than 10,000 source IPs, and generated 31,612 alarms over a four-week period (0.8 events per hour). In addition, Soule *et al.* [75] found that volume anomalies occurred at a rate of four per day on average, most of which involved fewer than several hundred source IPs. Finally, Ramachandran *et al.* [68] found were able to localize 4,963 Bobax-infected host IPs sending spam from a single vantage point. We envision our system could be used to improve accuracy of these techniques by correlating anomalies across ISP boundaries. We found our system could handle 10,000 IP addresses as keys, with a request rate of several hundred keys per second, even with several hundred participants. Given our system exceeds the requirements of anomaly detection, our system may enable the participants to "tune" their anomaly detectors to be more sensitive, and reduce false positive rates by leveraging other ISPs' observations.

Cross-checking certificates. Multiple vantage points may be used to validate authenticity of information (such as a DNS reply or ssh certificate [64, 81]) in the

presence of "man-in-the-middle" attacks. Such environments present potentially larger scaling challenges due to the potentially large number of keys that could be inserted. According to [38], most hosts execute fewer than 15 DNS lookups per hour, and according to [70], ssh hosts rarely authenticate with more than 30 remote hosts over long periods of time. Here, we envision our system could simplify the deployment of such schemes, by reducing the amount of information revealed about clients' request streams. Under this workload (15 key updates per hour, with 30 keys per participating host), our system scales to support several hundred hosts with only a single proxy. Extrapolating out to larger workloads, our system can handle tens of thousands of clients storing tens of thousands of keys with under fifty proxy/database pairs.

Distributed ranking. Search tools such as Alexa and Google Toolbar collect information about user behavior to refine search results returned to users. However, such tools are occasionally labeled as *spyware* as they reveal information about the contents of queries performed by users. Our tool may be used to improve privacy of user submissions to these databases. It is estimated that Alexa Toolbar has 180,000 active users, and it is known that average web users browse 120 pages per day. Here, the number of participants is large, but the number of keys they individually store in the system is smaller. Extrapolating our results to 180,000 participants, and assuming several thousands of keys, our system can still process several hundred requests per second (corresponding to several hundred thousand clients) per proxy/database pair.

4.6 Extended Protocol and Security Proof

Here, we describe the extended protocol of Section 4.3.2.

1. Each participant interacts with the proxy as follows. For each entry $\langle k_i, v_i \rangle$ in the participant's list, the participant and the proxy run a sub-protocol for encrypted oblivious evaluation of the PRF (EOPRF). At the end of this protocol, the participant learns nothing and the proxy learns only the value $E_{\mathrm{DB}}(F_s(k_i))$. The participant sends the values $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$ and $E_{\mathrm{DB}}(v_i)$ together with a proof of knowledge (POK) for knowing the plaintext of the last entry. If the POK succeeds, then the proxy re-randomizes the ciphertexts and adds the triple to a list. Otherwise, if the POK fails, the proxy ignores the triple.

2. Same as in the original protocol.

3. The DB builds the tables R and H as in the original protocol. For each row in R, the DB sends to the proxy the value $F_s(k_i)$ together with the corresponding list $\mathsf{E}[k_i]$ which supposedly contains ciphertexts of the form $E_{\mathrm{PRX}}(k_i)$. The DB also re-randomizes these ciphertexts.

4. The proxy goes over the received table. For each entry of the received table, it decrypts all the values in the list $\mathsf{E}[k_i]$ and verifies that the plaintext corresponds to the blinded key $F_s(k_i)$. It reports inconsistencies to the DB and sends k_i if it appears in the list $\mathsf{E}[k_i]$.

5. For each row, the DB updates the list $\mathsf{T}[k_i]$ by omitting the values v_i for which inconsistencies were found. Then, it applies f again to the updated row, checks whether it should be released, and, if so, publishes the corresponding key k_i

together with the updated list of values $\mathsf{T}[v_i]$. (The value k_i was given by the proxy as at least one of the ciphertexts in $\mathsf{E}[k_i]$ was consistent with the blinded key.)

We now sketch the proofs for the security of the protocol. First let us formally define the functionality we consider. Consider all submitted key-value pairs as a table, where each distinct key k_i is associated with a list $\hat{\mathsf{T}}[k_i]$ of all values v_i submitted with it. Let $\hat{\mathsf{R}}$ be the sub-table that consists of all the rows that should be revealed (according to f), and let $\hat{\mathsf{H}}$ be the table that contains all the other entries with the key column omitted. Our functionality outputs $\hat{\mathsf{R}}$ as a public value and $\hat{\mathsf{H}}$ as a private output for the DB. We prove that our protocol securely computes this functionality.

Honest but curious coalition of participants and a proxy. The joint view of the proxy and the honest-but-curious (HBC) participants contains the following: (1) the inputs (k_i, v_i) of the HBC participants and the public outputs $\hat{\mathsf{R}}$; (2) the information exchanged by the proxy and the HBC participants during the first stage; (3) the view of the proxy when interacting with other participants in the first stage, which consists of the proxy's view of the sub-protocols (EOPRF and POK) as well as triples of the ciphertexts $E_{\mathrm{DB}}(v_i)$, $E_{db}(F_s(k_i))$, and $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$; and (4) the table R sent by the DB to the proxy at the "revealing" phase of the protocol.

This view can be simulated, given the corresponding inputs (k_i, v_i) and the outputs $\hat{\mathsf{R}}$, as follows. Choose a random PRF key s, as well as public keys PRX and DB. Simulate (1) and (2) in the natural way (all the information needed for these computations is given). To simulate (3), use the simulators of the sub-protocols and generate garbage ciphertexts $E_{\mathrm{DB}}(0), E_{\mathrm{DB}}(0), E_{\mathrm{DB}}(0)$. To simulate (4), encrypt the

values in $\hat{\mathsf{R}}$ under PRX and blind the keys under s.

Honest-but-curious coalition of participants and a DB. The joint view of the proxy and the HBC participants contains the following: (1) the inputs (k_i, v_i) of the HBC participants and the public outputs $\hat{\mathsf{R}}$; (2) the view of the HBC participants during the interaction with the proxy, which consists of the view of the sub-protocols (EOPRF and POK) as well as triples of ciphertexts $E_{\mathrm{DB}}(v_i)$, $E_{db}(F_s(k_i))$, and $E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$; and (3) the view of the DB when interacting with the proxy, which consists of the tables R and H (encrypted under the DB's public key).

Given the corresponding inputs (k_i, v_i), the public output $\hat{\mathsf{R}}$, and the DB's private output $\hat{\mathsf{H}}$, we show how to simulate the above view. First, choose a random PRF key s, as well as public keys PRX and DB. Then, simulate (1) and (2) in the natural way (all the information needed for these computations is now given). It remains just to simulate R and H. The table R can be computed from $\hat{\mathsf{R}}$ and s. To simulate H, we should somehow add blinded values to $\hat{\mathsf{H}}$ (and encrypt the tuples under DB). We do this by building a key-value table for the inputs of the HBC participants. Then, for each row k_i, we choose a random consistent row in $\hat{\mathsf{H}}$ and add the value $F_s(k_i)$ as an additional blinded-key column. (A row is consistent with a key k_i if the list of values of the HBC's that are associated with k_i appear as part of the value list of the row in $\hat{\mathsf{H}}$.) Finally, for those rows which are left with no blinded key column, a random value is added.

Malicious coalition of participants. Let A be an adversarial strategy for a coalition of cheating participants. We construct a simulator that achieves the same "cheating" affect in the ideal-world. The simulator S chooses a key s for the PRF, as well as pairs of private/public keys for the DB and proxy. It provides these keys to

A and executes A. For each iteration i, A generates a triple (a_i, b_i, c_i), together with a POK for knowing the plaintext encrypted in c_i. (In an honest execution $a_i = E_{\mathrm{DB}}(F_s(k_i)), b_i = E_{\mathrm{DB}}(E_{\mathrm{PRX}}(k_i))$, and $c_i = E_{\mathrm{DB}}(v_i)$, for some k_i and v_i.) The simulator S uses the POK to extract v_i; if the POK fails, then S ignores the triple. Finally, S checks (using all the above keys) whether a_i and b_i are consistent (*i.e.*, it decrypts a_i to a'_i, decrypts b_i to b'_i, and then verifies that $F_s(b'_i) = a_i$). If the check fails, S ignores the tuple. Otherwise, the simulator, which now knows both k_i and v_i, passes these entries to the trusted party.

4.7 Conclusions

In this chapter, we presented the design, implementation, and evaluation of a collaborative data-analysis system that is both scalable and privacy preserving. Since a fully-distributed solution would be complex and inefficient, our design divides responsibility between two independent parties—a proxy that obliviously blinds the client inputs and a database that identifies the (blinded) keys that have values satisfying an evaluation function. The functionality of both the proxy and the database can be easily distributed for greater scalability and reliability. Experiments with our prototype implementation show that our system performs well under increasing numbers of keys, participants, and proxy/database replicas. The performance is well within the requirements of our motivating applications, such as collaborating to detect the malicious hosts responsible for DoS attacks or to validate the authenticity of information in the presence of man-in-the-middle attacks.

As part of our ongoing work, we plan to evaluate our system in the context of several real applications—first through a trace-driven evaluation and later by extending

our prototype to run these applications. In addition, we plan to explore opportunities to deploy our system in practice. A promising avenue is distributed Internet monitoring infrastructures such as NetDimes [78] and the new M-Lab (Measurement Lab) initiative [48]. We believe our system could lower the barriers to collaborative data analysis over the Internet, enabling a wide range of new applications that could improve Internet security, performance, and reliability.

Chapter 5

Conclusions

The preceding dissertation presented a complete collaborative anomaly detection architecture, from the high-speed detection of unwanted traffic in each network to a privacy-preserving protocol that allows these networks to increase their confidence in detections via collaboration. The primary contributions can be summarized as:

1. A machine learning-based system that can translate many packet-level Snort rules into flow-level rules with a high degree of accuracy. The flow-level rules lead to more computationally efficient detection of unwanted traffic [20].

2. A measurement study that demonstrates that collaboration between victims of unwanted traffic can help improve detection accuracy because many attackers have a high degree of *fan-out*.

3. A novel cryptographic protocol that allows these victims to collaborate while protecting the privacy both of the victims themselves and of the victims' legitimate customers. Our implementation of this protocol is able to process millions of suspect IP addresses within hours when running on two quad-core

machines.

More broadly, the thesis demonstrates that both machine learning algorithms and cryptographic protocols can be leveraged efficiently in an online fashion. In particular, our experiments and analysis indicates that the prototype system we built to mimic Snort on IP flows in chapter 2 would be able to process traffic from an OC48 link, and the distributed privacy-preserving collaboration system we presented in chapter 4 could handle millions of suspect IP addresses in the span of hours.

Moving forward, we see increased integration in industry and academia between high-performance systems and both cryptography and machine learning. For example, user privacy becomes increasingly important as the amount of data stored about users' online behavior explodes. Such privacy is important to end-users to protect against identity theft, and it is important to corporations that wish to protect their reputations [44]. The companies' desire for privacy is at odds with their desire to monetize the data they keep on their customers, however, and therein lies the challenge.

Consider the YouTube-Viacom suit [34], for example. Google—as the owner of YouTube—wishes to keep some amount of data on users' behavior, but does not wish to reveal this information to third parties such as Viacom. We believe there are interesting potential applications of secure multi-party computation to this problem area. One possibility would be to split Google's data set between multiple parties $S = \{\text{Google}, P_1, \ldots, P_k\}$ such that (A) Google retains the ability to perform some set of relevant queries (*e.g.*, "what videos do users tend to view after viewing video X" or "how many videos do users watch at 3pm EDT"); but (B) no single member of S can reconstruct the original data set currently stored by Google (indicating,

among other things, which user watched which video); and (C) the data stored by any P_i does not give P_i "meaningful insights" into Google's data set, according to some privacy definition satisfactory to Google. Such a system would force a third party such as Viacom to sue multiple members of S in order to gain access to meaningful data. Moreover, it is not unreasonable to assume that other companies facing similar challenges to Google (*e.g.*, Microsoft, Yahoo!, Amazon) might be willing to fill the roles of $P_1, \ldots, P_k$ under a quid pro quo system. We leave the exploration of this to future work.

Bibliography

[1] U.S. Securities and Exchange Commission: Pump and Dump Schemes. `http://www.sec.gov/answers/pumpdump.htm`.

[2] Ahmed, T., Oreshkin, B., and Coates, M. J. Machine learning approaches to network anomaly detection. In *Workshop on Tackling Computer Systems Problems with Machine Learning Techniques* (2007).

[3] Alexa the Web Information Company, 2009. `http://www.alexa.com/`.

[4] Allman, M., Blanton, E., Paxson, V., and Shenker, S. Fighting coordinated attackers with cross-organizational information sharing. In *Hot Topics in Networks (HotNets)* (November 2006).

[5] Allman, M., Paxson, V., and Terrell, J. A brief history of scanning. In *ACM Internet Measurement Conference* (October 2007).

[6] Barford, P., Kline, J., Plonka, D., and Ron, A. A signal analysis of network traffic anomalies. In *Internet Measurement Workshop* (2002).

[7] Ben-David, A., Nisan, N., and Pinkas, B. FairplayMP: A system for secure multi-party computation. In *ACM Conference on Computer and Communications Security (CCS)* (October 2008).

[8] Berghel, H. Identity theft, social security numbers, and the web. *Communications of the ACM 43*, 2 (2000), 17–21.

[9] Bernaille, L., Teixeira, R., and Salamatian, K. Early application identification. In *Conference on Future Networking Technologies* (2006).

[10] Bogetoft, P., Christensen, D. L., Damgard, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J. D., Nielsen, J. B., Nielsen, K., Pagter, J., Schwartzbach, M., and Toft, T. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. `http://eprint.iacr.org/`.

[11] Burrows, M. The Chubby lock service for loosely-coupled distributed systems. In *USENIX Symposium on Operating Systems and Implementation (OSDI)* (November 2006).

[12] Chor, B., Goldreich, O., Kushilevitz, E., and Sudan, M. Private information retrieval. *J. of the ACM 45*, 6 (November 1998).

[13] Chou, N., Ledesma, R., Teraguchi, Y., and Mitchell, J. C. Client-side defense against web-based identity theft. In *Network and Distributed System Security Symposium (NDSS)* (San Diego, CA, USA, February 2004).

[14] Cisco NetFlow. `http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html`.

[15] Composite Blocking List. `http://cbl.abuseat.org/`.

[16] Dagon, D., Gu, G., Lee, C., , and Lee, W. A taxonomy of botnet structures. In *Annual Computer Security Applications Conference* (Miami Beach, FL, USA, 2007).

[17] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *USENIX Security Symposium* (August 2004).

[18] Douceur, J. R. The Sybil attack. In *International Workshop on Peer-To-Peer Systems* (March 2002).

[19] Dudik, M., Phillips, S., and Schapire, R. E. Performance Guarantees for Regularized Maximum Entropy Density Estimation. In *Conference on Learning Theory (COLT)* (Banff, Canada, 2004), Springer Verlag.

[20] Duffield, N., Haffner, P., Krishnamurthy, B., and Ringberg, H. Rule-based anomaly detection on IP flows. In *IEEE INFOCOM* (Rio de Janeiro, Brazil, 2009).

[21] Duffield, N., Lund, C., and Thorup, M. Charging from sampled network usage. In *Internet Measurement Workshop* (2001), pp. 245–256.

[22] Erman, J., Mahanti, A., Arlitt, M. F., Cohen, I., and Williamson, C. L. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval. 64*, 9–12 (2007), 1194–1213.

[23] Fagin, R., Naor, M., and Winkler, P. Comparing information without leaking it. *Communications of the ACM 39*, 5 (1996), 77–85.

[24] Franklin, J., Paxson, V., Perrig, A., and Savage, S. An inquiry into the nature and cause of the wealth of internet miscreants. In *ACM Conference on Computer and Communications Security (CCS)* (October 2007).

[25] Freedman, M. J., Ishai, Y., Pinkas, B., and Reingold, O. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference (TCC)* (February 2005).

[26] Freedman, M. J., Nissim, K., and Pinkas, B. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT* (May 2004).

[27] Friend-of-a-Friend Project, 2009. `http://www.foaf-project.org/`.

[28] Garriss, S., Kaminsky, M., Freedman, M. J., Karp, B., Mazières, D., and Yu, H. Re: Reliable email. In *ACM Networked Systems Design & Implementation (NSDI)* (May 2006).

[29] Geant Network. `http://www.geant.net/`.

[30] Goldreich, O. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[31] Gomes, L. H., Cazita, C., Almeida, J. M., Almeida, V., and Meira, Jr., W. Characterizing a spam traffic. In *ACM Internet Measurement Conference* (New York, NY, USA, 2004), pp. 356–369.

[32] Google Safe Browsing for Firefox, 2009. `http://www.google.com/tools/firefox/safebrowsing/`.

[33] Hazay, C., and Lindell, Y. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography Conference (TCC)* (March 2008).

[34] Helft, M. Viacom suit against google raises privacy concerns. *The New York Times* (July 2008). `http://www.nytimes.com/2008/07/04/technology/04iht-youtube.1.14234592.html`.

[35] Ianelli, N., and Hackworth, A. Botnets as a vehicle for online crime. In *CERT Coordination Center* (December 2005).

[36] Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO* (August 2003).

[37] Jiang, H., Moore, A. W., Ge, Z., Jin, S., and Wang, J. Lightweight application classification for network management. In *SIGCOMM Workshop on Internet Network Management* (2007).

[38] Jung, J., Sit, E., Balakrishnan, H., and Morris, R. DNS performance and the effectiveness of caching. *IEEE/ACM Trans. Networking 10*, 5 (October 2002).

[39] Kannan, J., Subramanian, L., Stoica, I., and Katz, R. Analyzing cooperative containment of fast scanning worms. In *SRUTI* (July 2005).

[40] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing (STOC)* (1997).

[41] Katti, S., Krishnamurthy, B., and Katabi, D. Collaborating against common enemies. In *ACM Internet Measurement Conference* (Berkeley, CA, USA, 2005), pp. 34–34.

[42] Kissner, L., and Song, D. Privacy preserving set operations. In *Advances in Cryptology - CRYPTO* (August 2005).

[43] Kompella, R. R., Singh, S., and Varghese, G. On scalable attack detection in the network. In *ACM Internet Measurement Conference* (2004).

[44] Krebs, B. Payment processor breach may be largest ever. *The Washington Post* (January 2009). `http://voices.washingtonpost.com/securityfix/2009/01/payment_processor_breach_may_b.html`.

[45] Lakhina, A., Crovella, M., and Diot, C. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM* (2004).

[46] Lakhina, A., Crovella, M., and Diot, C. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM* (2005), pp. 217–228.

[47] Lindell, Y., and Pinkas, B. Privacy preserving data mining. In *Advances in Cryptology - CRYPTO* (August 2000).

[48] M-Lab: Welcome to Measurement Lab, 2009. `http://www.measurementlab.net/`.

[49] Madhyastha, H., and Krishnamurthy, B. A generic language for application-specific flow sampling. *Computer Communication Review* (April 2008).

[50] Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y. Fairplay: A secure two-party computation system. In *USENIX Security Symposium* (August 2004).

[51] Mann, C. C. How click fraud could swallow the internet. *WIRED* (January 2006). `http://www.wired.com/wired/archive/14.01/fraud.html`.

[52] Mao, Z., Sekar, V., Spatscheck, O., van der Merwe, J., and Vasudevan, R. Analyzing large DDoS attacks using multiple data sources. In *SIGCOMM Workshop on Large Scale Attack Defense* (September 2006).

[53] Metwally, A., Agrawal, D., and El Abbadi, A. Duplicate detection in click streams. In *International World Wide Web Conference* (New York, NY, USA, 2005), pp. 12–21.

[54] Moore, A., and Zuev, D. Internet traffic classification using bayesian analysis. In *ACM SIGMETRICS* (2005).

[55] Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., and Weaver, N. Inside the slammer worm. *IEEE Symposium on Security and Privacy 1*, 4 (2003), 33–39.

[56] Moore, D., Shannon, C., Voelker, G., and Savage, S. Internet quarantine: Requirements for containing self-propagating code. In *IEEE INFOCOM* (San Francisco, CA, USA, 2003).

[57] Naor, M., and Pinkas, B. Oblivious transfer and polynomial evaluation. In *ACM Symposium on Theory of Computing (STOC)* (May 1999).

[58] Naor, M., and Pinkas, B. Oblivious transfer with adaptive queries. In *Advances in Cryptology - CRYPTO* (August 1999).

[59] Naor, M., and Reingold, O. Number-theoretic constructions of efficient pseudorandom functions. In *IEEE Symposium on Foundations of Computer Science (FOCS)* (October 1997).

[60] Olsen, S. Click fraud roils search advertisers. *CNet* (March 2005). http://news.cnet.com/Click-fraud-roils-search-advertisers/2100-1024_3-5600300.html.

[61] Oracle. Berkeley DB, 2009. http://www.oracle.com/technology/products/berkeley-db/.

[62] Paxson, V. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks 31* (December 1999), 2435–2463.

[63] Peterson, P. The billion dollar problem (interview). In *IT Security* (January 2007). http://www.itsecurity.com/interviews/billion-dollar-problem-ironport-\malware-012607/.

[64] Poole, L., and Pai, V. S. ConfiDNS: Leveraging scale and history to improve DNS security. In *Workshop on Real, Large Distributed Systems (WORLDS)* (November 2006).

[65] Privacy Rights Clearinghouse. A chronology of data breaches, January 2009. http://www.privacyrights.org/ar/ChronDataBreaches.htm.

[66] Rabin, M. How to exchange secrets by oblivious transfer. Tech. Rep. TR-81, Harvard Aiken Computation Laboratory, 1981.

[67] Rajab, M. A., Zarfoss, J., Monrose, F., and Terzis, A. My botnet is bigger than yours (maybe, better than yours): why size estimates remain chal-

lenging. In *Hot Topics in Understanding Botnets (HotBots)* (Berkeley, CA, USA, 2007).

[68] Ramachandran, A., and Feamster, N. Understanding the network-level behavior of spammers. In *ACM SIGCOMM* (2006).

[69] Schapire, R. E., and Singer, Y. Improved boosting algorithms using confidence-rated predictions. *Machine Learning 37*, 3 (1999), 297–336.

[70] Schechter, S., Jung, J., Stockwell, W., and McLain, C. Inoculating SSH against address harvesting. In *Network and Distributed System Security Symposium (NDSS)* (February 2006).

[71] Schnorr, C.-P. Efficient signature generation by smart cards. *Journal of Cryptology 4*, 3 (1991), 161–174.

[72] Schuba, C. L., Krsul, I. V., Kuhn, M. G., spafford, E. H., Sundaram, A., and Zamboni, D. Analysis of a denial of service attack on TCP. In *IEEE Symposium on Security and Privacy* (Washington, DC, USA, 1997).

[73] Shon, T., and Moon, J. A hybrid machine learning approach to network anomaly detection. *Inf. Sci. 177*, 18 (2007), 3799–3821.

[74] Snort. `http://www.snort.org`.

[75] Soule, A., Ringberg, H., Silveira, F., Rexford, J., and Diot, C. Detectability of traffic anomalies in two adjacent networks. *Passive and Active Measurement (PAM)* (2007).

[76] Soule, A., Salamatian, K., and Taft, N. Combining filtering and statistical methods for anomaly detection. In *ACM Internet Measurement Conference* (2005), pp. 1–14.

[77] SpamHaus. `http://www.spamhaus.org`.

[78] The DIMES Project, 2009. `http://www.netdimes.org/new/`.

[79] Vapnik, V. N. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[80] Weber, T. Criminals 'may overwhelm the web'. In *BBC News* (January 2007). `http://news.bbc.co.uk/1/hi/business/6298641.stm`.

[81] Wendlandt, D., Andersen, D. G., and Perrig, A. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX Annual Technical Conference* (2008).

[82] Xie, Y., Reiter, M. K., and O'Hallaron, D. Protecting privacy in key-value search systems. In *Annual Computer Security Applications Conference* (Washington, DC, USA, 2006), pp. 493–504.

[83] Xie, Y., Yu, F., Achan, K., Gillum, E., Goldszmidt, M., and Wobber, T. How dynamic are IP addresses? *ACM SIGCOMM* (2007).

[84] Yahoo! Hadoop Team. Zookeeper. `http://hadoop.apache.org/zookeeper/`, 2009.

[85] Yao, A. C. Protocols for secure computations. In *IEEE Symposium on Foundations of Computer Science (FOCS)* (November 1982).

[86] Zhang, Y., Ge, Z., Greenberg, A., and Roughan, M. Network anomography. In *ACM Internet Measurement Conference* (New York, NY, USA, 2005), ACM, pp. 1–14.