# Differential Privacy for Collaborative Security

Jason Reed, Adam J. Aviv, Daniel Wagner,
Andreas Haeberlen, Benjamin C. Pierce, Jonathan M. Smith

University of Pennsylvania

## ABSTRACT

Fighting global security threats with only a local view is inherently difficult. Internet network operators need to fight global phenomena such as botnets, but they are hampered by the fact that operators can observe only the traffic in their local domains. We propose a *collaborative* approach to this problem, in which operators share aggregate information about the traffic in their respective domains through an automated query mechanism. We argue that existing work on differential privacy and type systems can be leveraged to build a programmable query mechanism that can express a wide range of queries while limiting what can be learned about individual customers. We report on our progress towards building such a mechanism, and we discuss opportunities and challenges of the collaborative security approach.

## 1. INTRODUCTION

The constant evolution of the Internet has given rise to new distributed software systems supporting antisocial and criminal behaviors. Among these are *botnets*, collections of exploited network hosts, or *bots*, under a common control. Botnets can place a severe drain on network resources, as they are often leased [1] for the purposes of distributed denial of service [18] or massive spamming campaigns [19]. Detecting and mitigating botnets is thus an area of urgent concern.

Current approaches to botnet detection often rely on the fact that bots act in concert. Intrusion detection systems (IDS) use extensive logging and data collection to identify anomalous behavior and to verify it as botnet activity [2, 7, 12, 13, 14, 15, 20, 22, 29]. However, this approach is hampered by the fact that, for privacy reasons, networks do not usually share collected data across administrative domains: each domain maintains its own IDS, whose view is limited in scope. This helps botmasters to hide their bots' activity by spreading it across diverse regions of the Internet. Similarly, disrupting a botnet's command and control (C&C) structure can be difficult within a single administrative domain because the control structure can change dynamically and span multiple domains.

A *collaborative approach* to botnet detection could mitigate these visibility restrictions. For example, suppose the IDS in network $\alpha$ signals an unusual spike in UDP traffic on port $55555$, but the operator in network $\alpha$ is unsure if this is caused by a botnet or is simply a false alarm. If the operator had access to statistics from an adjacent network, say $\beta$, she could check whether this anomaly is a broader phenomenon. But there are two obvious difficulties. First, $\beta$ will not make the necessary data available to $\alpha$ without assurances that it does not reveal sensitive information about $\beta$'s customers. Deriving such assurances may require detailed and expensive analysis of the data being transferred. Second, even if $\beta$ could establish some assurance on the sensitivity of this particular query, the next anomaly will likely cause $\alpha$ to ask for completely *different* statistics, requiring $\beta$ to repeat the same expensive analysis. Without automated support, this sort of cross-domain information sharing seems infeasible.

We believe that *differential privacy* [21, 3, 4, 9, 8, 10, 11, 27] offers a promising approach to enabling collaboration across administrative domains. Differential privacy offers statistical guarantees about the privacy of randomized query functions, avoiding the limitations of deterministic anonymization [8]. It does this by quantifying how much an individual's privacy is affected by revealing the result of a particular query, and thus how much random noise is needed to perturb the results sufficiently to assure privacy. In the above example, only a little random noise would be needed, since UDP traffic statistics can be aggregated over many flows from many different customers; the information contributed by any individual customer would be negligible. Moreover, it is possible to *algorithmically* derive a bound on the impact of a given query [24]. There is reason to hope that this approach can address both of the problems discussed above—first, by allowing networks to offer an *automated* service by which other networks can query their IDS's data; and second, by providing well-understood and automatically verifiable *guarantees* on the amount of privacy loss, which are robust even against adversaries with external information. The latter property is important for avoiding unexpected deanonymization, a significant worry in practice (as Netflix recently discovered [26]).

However, applying differential privacy to queries over network statistics poses some fundamental challenges. Existing work on differential privacy considers queries over a static database, whereas in the Internet new traffic needs to be collected and queried continuously. Query responses might be correlated over time by the attacker, so enforcing an upper bound on information leakage in this setting is difficult. Another significant challenge stems from the multi-directionality of communication over the Internet: a given party may communicate with many different partners over time, causing potentially sensitive information to be widely distributed across varied administrative domains; this makes it difficult to organize the stored information to protect the privacy of individuals.

In this paper, we analyze both the opportunities and the challenges of implementing collaborative botnet detection using differential privacy. Our contributions include (1) a sketch of a simple cross-domain IDS architecture, (2) a proposal for a core query language with a type system that can automate the process of determining the appropriate amount of random noise needed to sufficiently anonymize a query result, (3) a set of sample botnet detection queries that can be shown to be sufficiently privacy-preserving, and (4) an analysis of some important remaining challenges, including issues of continuous collection of data, correlating query results over time, and determining appropriate "privacy budgets."

## 2. BACKGROUND

Before presenting our model for collaborative security, let us briefly review the basic concepts of differential privacy.

### 2.1 Differential Privacy

A key requirement for collaborative security is the ability to share data without unduly compromising the privacy of individuals. At a high level, we can do this by sharing not the data itself, but rather aggregate statistics about the data. However, it is not obvious *a priori* how much information a particular statistic reveals about a specific individual.

Differential privacy provides a way to quantify this. In general, it is not possible to guarantee that publishing even an aggregate statistic will preserve an individual's privacy in *absolute* terms, since an attacker can always come to some unwanted conclusion if he has access to outside information [8]. However, differential privacy can provide a strong *relative* limit on how much *more* an individual's privacy is jeopardized by including her in the aggregate, compared to computing the aggregate on the rest of the data only.

Consider an adversary that is trying to learn something about a specific individual by observing the results of queries against some database. Intuitively, a mechanism for releasing data from the database is private if, with high probability, anything that could be deduced from the released data could also be deduced from the data without the contribution from any given individual. This means that any individual can safely allow their data to be included in a database because an adversary cannot make any conclusions that could not already have been made, except with low confidence.

This intuition can be formalized as follows. A database is a collection of *tuples* (or *rows*), where we think of each tuple as belonging to a user whose privacy we are concerned with protecting. For example, in a hospital database, one tuple might contain a patient's entire medical record. Let $T$ be the set of all possible tuples and the powerset $B = \mathcal{P}(T)$ of $T$ be the set of all possible databases. Write $b \sim b'$ ("$b$ is close to $b'$") to mean that $b'$ differs from $b$ by the inclusion or deletion of one tuple.

A randomized function $f : B \to \mathbb{R}^n$ that computes a vector of real numbers from a database is called *$\epsilon$-differentially private* (or, it 'achieves $\epsilon$-differential privacy') iff, for any databases $b, b'$ with $b \sim b'$ and for any set $S \subseteq \mathbb{R}^n$,

$$Pr[f(b) \in S] \leq e^{\epsilon} \cdot Pr[f(b') \in S]. \qquad (*)$$

The number $\epsilon$ is a parameter that the system designer can choose based on how much privacy is desired — the smaller the $\epsilon$, the smaller the statistical violation of privacy. Intuitively, the set $S$ plays the role of a 'conclusion' that an adversary might try to make based on the vector of real numbers that is output. The impact of $(*)$ is that the probability that the adversary comes to that conclusion cannot vary much — only at most by a factor of $e^{\epsilon}$ — depending on the inclusion or exclusion of one individual in the database.

### 2.2 Mechanisms for Differential Privacy

Continuing the hospital example, we might want to compute a histogram of weights of the patients. This function is *not* $\epsilon$-differentially private on its own because it returns precise results without any random perturbations. A set $S$ such as:

$$S = \{x \in \mathbb{R}^n \mid x \text{ says} \geq 1 \text{ person weighs 200-210 lbs.}\}$$

can very well include or not include the result of a query based on the presence or absence of just one person.

To achieve differential privacy for this query, we add a bit of noise to the output of the deterministic query. In particular, when the deterministic query returns a vector of real numbers, we can apply the well-known *Laplace mechanism* [11] to analyze the underlying function and apply the appropriate amount of random noise to achieve $\epsilon$-differential privacy.

The analysis consists of considering the deterministic function's *sensitivity* to the input database. We say that $f : B \to \mathbb{R}^n$ is $\eta$-sensitive if, for any databases $b, b'$ with $b \sim b'$, we have $||f(b) - f(b')||_1 \leq \eta$, where $||v||_1$ is the $L^1$-norm of a vector $v \in \mathbb{R}^n$, defined as usual by $||(x_1, \ldots, x_n)||_1 = \sum_i |x_i|$. A function that is $\eta$-sensitive for small $\eta$ only varies (deterministically) a little for each change in the database, and so adding just a bit of noise to it will effectively conceal the influence of individual participants.

The Laplace mechanism takes a deterministic function $f$ and adds noise to its output, distributed according to the Laplace distribution, transforming it into an $\epsilon$-differentially private randomized function. Formally, a random vector $\mathcal{L}_\sigma^n$ in $\mathbb{R}^n$ has the Laplace distribution in $n$ dimensions with parameter $\sigma$ iff it satisfies:

$$Pr[\mathcal{L}_\sigma^n = v] \propto e^{-||v||_1/\sigma}$$

The distribution is peaked around $v$ being zero, and falls off quickly as $v$ gets bigger. In other words, we are most likely to add little noise. The parameter $\sigma$ controls the spread of the distribution: a larger $\sigma$ means a more spread-out, more 'noisy' distribution. The key theorem, which can be found in [11], is this:

THEOREM 2.1. *Suppose $g : B \to \mathbb{R}^n$ is a deterministic function that is $\eta$-sensitive. If we set $\sigma = \eta/\epsilon$, and define the random function $f : B \to \mathbb{R}^n$ by $f(x) = g(x) + \mathcal{L}_\sigma^n$, then $f$ is $\epsilon$-differentially private.*

Theorem 2.1 relates the choice of how much noise is added (controlled by $\sigma$) to our ability to achieve $\epsilon$-differential privacy. Driving the selection are $\eta$ (the maximum possible variation in the output of $g$ due to the absence of any given individual) and $\epsilon$ (the amount of privacy required). For large $\eta$ (i.e, large variation is possible) and for small $\epsilon$ (strong privacy requirements), $\sigma$ must be large, and thus a greater amount of noise is necessary. Conversely, if privacy requirements are weaker, then less noise is needed; and if there is less variation in the contribution of an individual, then again less noise is needed to ensure $\epsilon$-differential privacy of the function.

There are other mechanisms for differential privacy [23, 16], dealing with output types other than $\mathbb{R}^n$. For brevity, we do not discuss them here; we conjecture that they could be incorporated into our proposed framework.

## 3. SYSTEM MODEL

In this section, we describe our assumptions about the environment in which a query mechanism would operate, and the threats it needs to protect against. We also introduce some terms and definitions.

### 3.1 Assumptions and Definitions

We model a network as a set of interconnected *autonomous systems* (ASes). Each user is connected to at least one AS by an *access link*, such as a DSL or cable connection. An access link may have multiple connected users, and a user may connect via multiple links. However, it is assumed that a flow will cross exactly two access links: upstream from the sender and downstream to the receiver.

At any point in time, some fraction of the users' machines may be compromised and actively participating in a botnet. The AS administrators want to identify these infected machines.

To aid in this effort, administrators share information regarding the traffic within their networks across AS boundaries. Initially, we assume that an AS $\alpha$ has a database $I_\alpha$ that can be queried

locally. $I_\alpha$ consists of packet traces[1] over all access links within $\alpha$'s network, and is collected during a fixed time $T$, e.g., a specific day. Section 6.1 discusses generalizing this assumption to allow continuous collection. An AS $\beta$ can request a query over $I_\alpha$ in the form of a function $f$, to which $\alpha$ responds (if it chooses to) by returning $f(I_\alpha) + \nu$, where $\nu$ is the Laplace noise term.

Initially, we assume the database has one tuple for each access link, and that each tuple contains all the packets sent or received over the corresponding link. Since the differential privacy guarantees are stated in terms of tuples, organizing the database in this way means that we obtain privacy guarantees in terms of access links. This is useful because each link typically corresponds to either a single customer or a group of closely related customers (e.g., family members). Alternative choices are discussed in Section 6.2.

Each AS $\alpha$ maintains trust relationships with some other ASes $\beta_1, \ldots, \beta_k$. For each AS $\beta_i$, $\alpha$ chooses a privacy budget $\varepsilon_i$ and a number $n_i$ of queries it is willing to answer from $\beta_i$. When $\beta_i$ asks a query $f$, $\alpha$ determines the sensitivity $\eta_f$ of $f$ (see Section 4) and then chooses the noise $\nu$ such that $f$ is $\varepsilon_i/n_i$-differentially private. Thus, $\alpha$ can enforce a hard limit on the exposure of its customers' sensitive information. By reasoning about the effects of multiple queries (see for instance [24]) we can see that the privacy losses add together, and the total exposure is bounded by $\sum_i \varepsilon_i$. Note the tradeoff between privacy and detection power: lower values of $\varepsilon_i$ result in better privacy but cause more noise to be added to each query, which decrease their usefulness.

## 3.2 Threat Model

We assume that the privacy of the customers in an AS $\alpha$ must be protected against an adversary who is interested in learning (potentially private) information about an individual customer or a small group of customers. We assume that the adversary (1) can choose all of the queries that are asked of $\alpha$, (2) can eventually learn all of the results, and (3) may have arbitrary prior knowledge about $\alpha$'s customers. This is a very conservative assumption, but we believe that it is necessary because ASes are unlikely to deploy a query mechanism unless they can give their customers strong assurances that their privacy will not be compromised.

We also assume that the adversary does not have direct access to $I_\alpha$, and that it is permissible for the attacker to learn aggregate statistics about large groups of customers. The latter is inherent in our approach because the query mechanism is *intended* to reveal such aggregate information.

# 4. INFERRING QUERY SENSITIVITY

Botnet detection and mitigation is an arms race; botnets are continually evolving and adapting. Even in the last five years, there have been huge advances in command and control, coordination, and propagation strategies. Maintaining pace with these changes involves adaptability and flexibility: detection of future botnets will likely need different queries than the detection of today's botnets. Thus, rather than supporting a fixed set of queries, we believe it is necessary to provide open-ended support in the form of a flexible and expressive programming language. A critical question, then, is whether we can design this language so that it also tracks the sensitivity as queries are composed from operators.

Recent work by McSherry [24] proposed the idea of a rich, compositional language for differentially private queries and identified the notion of sequential and parallel composition of queries to preserve privacy properties. However, we believe that a language can be much more flexible by building upon the notion of a *differential privacy type system*.

We have designed the theoretical core of a functional programming language for queries, with a type system that expresses the deterministic notion of sensitivity. For any well-typed program written in our language, the sensitivity of the outputs to the inputs can be read off, informing how much noise is required to provide the desired degree of differential privacy.

In outline, the key idea of this language is that we equip each data type $\tau$ of the programming language with a *metric* that allows us to measure how far apart two values of $\tau$ are from one another. A metric is a function $d_\tau : \tau \times \tau \to \mathbb{R}$ required to satisfy the usual metric space axioms:

$$\begin{aligned} d_\tau(x,x) &= 0 \\ d_\tau(x,y) &= d_\tau(y,x) \\ d_\tau(x,y) + d_\tau(y,z) &\geq d_\tau(x,z) \end{aligned}$$

For instance, while the type of real numbers would have the usual metric $d_{\mathbb{R}}(x,y) = |x - y|$, the type of databases would have a metric such that $d_B(b,b') = 1$ when $b$ and $b'$ differ by one tuple ($b \sim b'$). With such metrics, the previous notion of $\eta$-sensitivity generalizes to arbitrary functions: a function $\tau_1 \to \tau_2$ is said to be $\eta$-sensitive if, for any $x$ and $y$ such that $d_{\tau_1}(x,y) = r$, it is the case that $d_{\tau_2}(f(x), f(y)) \leq \eta r$.

This notion of sensitivity is *compositional*: We can reason about the sensitivity of large programs in terms of their smaller parts. For example, note that $f \circ g$ is $\eta_1\eta_2$-sensitive if $f$ is $\eta_1$-sensitive and $g$ is $\eta_2$-sensitive. Similar properties hold concerning the calculations involving more complex data structures, and from these arise the language's typing rules.

Our design is still preliminary but, even with a modest set of primitive operations, it is already possible to capture many common basic programming idioms. In particular, our core language supports: Queries that use linear arithmetic; SQL-like operations such as joins of tables and selecting rows satisfying an arbitrary predicate; functional programming idioms like map and fold; and aggregation such as sums, max, and min of collections. The type system permits programming the sort of histogram computations typical in the differential privacy literature [27] in a natural way. Some examples from Chaudhuri [5]—including sorting algorithms and finding minimum-cost paths in graphs—can be adapted to our setting. It is also possible to program divide-and-conquer analyses like the Fast Fourier, Haar Wavelet, and Walsh-Hadamard transforms, enabling queries to take advantage of existing work in spectral analysis of network traffic [6].

# 5. SAMPLE QUERIES

We now discuss several examples based on present-day botnet detection techniques, to give a feel for the sorts of queries we have in mind, their privacy implications, and the way these can be addressed by the differential-privacy-sensitive programming language sketched in the previous section. There are three main use-cases: confirming that a local anomaly is part of a global phenomenon, discovering C&C channels used in other networks and preemptively throttling or censoring them, and measuring the magnitude of some botnet action. For the sake of familiarity, we present the examples in SQL-like pseudocode.

## 5.1 Rishi

Bots using `IRC` for C&C must all choose unique nicknames, and although botmasters may change the location of the `IRC` server, the procedure for choosing `IRC` nicknames usually remains fairly

[1]To simplify the discussion, we assume that the AS collects complete traces. In practice, the AS might choose to collect less data to save space, or to avoid liability.

consistent. The *Rishi* botnet detection system exploits this fact; it identifies new bots by observing the nickname selected (and some other characteristics) as hosts join an IRC chat session [12]. For example, some botnets use nicknames that contain a country code, operating system info, and a random sequences of digits. The Rishi system provides a heuristic for assessing how bot-like a nickname is. Suppose an operator wanted to estimate the size and threat level of an IRC botnet based on the nicknames. Internally, the threat may seem relatively small and not worth the disruption effort, but externally there may be thousands of bots participating. An operator could pose this query to learn such information.

```
Select Count(
       RISHI_Score(results) > RISHI_Thresh)
Where  type == IRCMessage &&
       IRC.channel = target_channel &&
       IRCHeader In (NICK, JOIN, MODE)
```

From the point of view of the differential privacy analysis, this is a very straightforward query. A simple count of even a rather complex property of users is 1-sensitive, since the presence or absence of a single user can only change the count by one.

## 5.2 Work weight

Binkley *et al.* proposed another method for detecting IRC C&C channels. They observed that many bots have a high ratio of TCP control packets to data packets during denial-of-service attacks [2]. They called this ratio the *TCP work weight*, and observed that it was possible to correlate spikes in work weight with IRC traffic to identify potentially evil channels. This detection scheme can be used even without knowing which IRC servers a botnet connects to or the command protocol within the channel. As operator could frame a work-weight query in three parts, separately finding the work weight of each host, the IRC channels each host participated in, and finally the statistic of interest based on these two pieces of information.

```
Select (source,
        Count(control) / Count(all))
Where   type == TCP
GroupBy source
As      TCP(source, weight)

Select (target, channel)
Where   type == IRCMessage
As      IRC(target, channel)

Select (IRC.channel, Count(*), Sum(TCP.weight))
From    InnerJoin TCP IRC
        On TCP.source == IRC.target
GroupBy IRC.channel
```

Such a query could potentially enable the discovery of channels that bots in other ISPs were connected to, and preemptively regulate traffic on those channels within our own network.

The intent is that only the results of the final Select are noised and published. Query programmers are allowed to phrase their queries in terms of (not obviously privacy-safe) intermediate data, provided the final result has sufficiently low sensitivity to inputs.

Analyzing the sensitivity of this query is more subtle. Part of it is easy: the count and the sum are each of sensitivity 1, so the sensitivity for the two together is simply 2. Although we have not discussed how to analyze the sensitivity of returning channel names (which are not numbers), there is a way to make the analysis formal; we omit the discussion for brevity.

Ultimately, we want to know whether the *average* work weight is anomalously high, but this query returns the population and sum-of-work-weights, leaving the operator to divide them. This is to achieve a low sensitivity (relative to the size of the result of the query). A query that returned just the average work weight would yield a number between 0 and 1, but still with a sensitivity of 1 (since the worst case is that there is one person in the channel, and the result reflects exactly that person's work-weight), so query results would have to be very noisy.

This example highlights the importance of flexibility in the query language. It gives the programmer the freedom to choose at what point in the computation noise is added to maximize the effectiveness of the query.

## 5.3 Domain flux

Some botnets—notably Torpig [28]—locate their C&C channels using a technique called *domain flux*. A bot in a domain-flux botnet has a long list of domain names that could potentially be control servers. When the time comes to contact the control server, it runs down the list and attaches to the first one that behaves like a control server. To help avoid detection, the list is rotated frequently. Suppose an AS observes an unusually popular string of DNS requests and wonders whether it is part of a domain-flux botnet. In this case, the AS can run a query like:

```
Select   Count((evil.monday.com,
                evil.monday.org,
                evil.monday.net)
            All In target)
Where    type == DNSRequest
GroupBy  source
```

which, being a count of users, has a sensitivity of 1. This means that end-users can expect their interest in suspected subdomains to be well-masked by a small amount of noise.

## 5.4 Fast flux

Some botnets use *fast flux* to protect the identity of the command servers. In fast flux, some of the bots self-identify as proxy bots, which exist only to forward messages between normal bots and the command servers. To contact the proxy bots, there is an agreed-upon domain name for the normal bots to look up. This domain name is given a very short time-to-live, so that the IP address that it resolves to can be rapidly rotated between the addresses of the available proxy bots. This behavior can be detected with fairly high accuracy [17]. In particular, it is unusual for benevolent domain names to resolve to IP addresses in many different autonomous systems; from this observation, Holz *et al.* derived a classification function that can be formulated as a query in the following way:

```
Select  Top 10
        (   1.32*Count(Unique(resultIP))
        +  18.54*Count(Unique(AS(resultIP)))
        - 142.38, target)
Where   type == DNSRequest
GroupBy target
```

The result of this query can be thought of as a histogram. For any particular target domain name, the sensitivity is 19.86 (that is, $1.32 + 18.54$), and, since sensitivities add, the total sensitivity would be the highest so far: 198.6. This means that providing privacy would require a lot of noise. Still, the query can be useful because the expected aggregates are also quite large.

## 5.5 Backscatter analysis

Finally, we consider *backscatter analysis*, a technique for estimating the size of distributed denial-of-service attacks [25]. Some DDoS attacks involve flooding the victim with SYN packets; to avoid detection, these packets have forged return IP addresses (which we assume are drawn approximately uniformly at random from all possible IP addresses to make our final analysis easier). One way to detect this is to scan for unsolicited SYN/ACK packets, that is, packets sent to a host with a different identifier than any SYN packets that host sent:

```
Select Count(target)
Where  type == TCPSynAck &&
       (target, packetNumber) Not In
            (Select (source, packetNumber)
             Where type == TCPSyn)

Select Count(target)
```

Once an AS has the number of hosts that received stray ACKs (the first `Select` above) and the total number of hosts (the last `Select` above), it can extrapolate to the number of hosts anywhere on the net that were impersonated; this is also the size of the attacking botnet. Since the accuracy of this measurement depends critically on the number of hosts covered by the result, cross-domain collaboration would be ideal.

Again, the sensitivity of this and any other pure counting query is low: 1 for each of the `Select` clauses, for a total of 2. Thus, only a small amount of noise will need to be added to mask any particular individual's contribution to the count.

# 6. CHALLENGES

In this section, we identify some fundamental challenges to address in future work.

## 6.1 Continuous data collection

The most straightforward application of differential privacy is the execution of a single query (or a finite number of queries) on a database, after which the entire database is discarded. However, the monitoring of network traffic is an ongoing event. Even if we assume just a single query can be requested for each day's worth of data, the privacy guarantees can only be applied to that day's collection. However, when correlated, queries performed over multiple days may actually reveal sensitive information.

Suppose the adversary is interested in whether or not a person $X$ has visited `embarrassing.com`. If $I_\alpha$ just contains data for one day, the adversary can learn very little. However, if $I_\alpha$ is replaced with fresh data every day and the privacy budgets are replenished, the situation is different. While the adversary may not be able to conclusively prove that $X$ visited `embarrassing.com` on a *particular* day, his confidence that $X$ visited `embarrassing.com` on *some* day will continue to increase without bound for each subsequent query. We are not aware of any technique that could properly apply differential privacy in a setting with streaming data collection and a potentially unbounded number of queries.

## 6.2 Correlation

A closely related challenge involves the correlation between tuples in the input database. As discussed in Section 3, we propose an organization indexed by access link, grouping together all packets sent from a particular link. But Internet traffic always involves two parties, a sender and a receiver. This means that some information about the receiver (namely, that he was sent a packet) is not stored in the tuple associated with the receiver. This may seem acceptable on the surface; in one sense, people can only really be held responsible for the packets they send, and not for the packets other people choose to send them.

In the context of the Internet, however, many interactions have request-response dynamics: person $x$ sends person $y$ a request (say, for the contents of the `embarrassing.com` website), and person $y$ sends back a response. Thus, potentially private information about $x$ is spread out among all the tuples with which he communicated, and is therefore not protected by the guarantees of differential privacy.

One way to avoid this problem is to organize the database $I_\alpha$ in a different way. For example, we might consider including only packets sent, rather than packets sent or received; however, this might preclude some useful queries, and it would still not prevent semantic connections between packets, e.g., in request-response communications. Another option would be to choose the tuples differently, e.g., by having not one tuple per access link, but rather one tuple per packet, per flow, per time interval, etc. However, recall that the tuples define the entities whose privacy is guaranteed, and it is not obvious what it would mean in practice to protect the privacy of, for example, an individual packet.

Another approach is to break the correlation by relaxing the privacy of some entities on the network whose privacy is considered unimportant. Domain-name servers, for example, are inherently public; suppose we could convince the owners of the servers to waive their privacy guarantees. In that case, we could then avoid storing customer $x$'s domain-name server requests (and the server's responses) in the tuple for the server – in fact, we can eliminate the server's row from the database entirely. This would allow us to give very strong guarantees about the information that could be gleaned about customer $x$'s domain-name lookup behavior. Similar waivers could protect $x$'s web traffic to large websites or `IRC` sessions with public `IRC` servers.

## 6.3 Reasoning about beliefs

Ideally, it would be desirable for an ASes to establish a fixed limit $\bar{\varepsilon}$ on the likelihood that an adversary can infer a particular (private) fact about one of their customers, such as whether or not customer $X$ visited a particular web page. This guarantee would be both extremely strong and easy to understand. However, differential privacy actually provides a slightly different guarantee. The two main differences are that (1) knowledge of a private fact is not binary, in the sense that the adversary may also *believe* that the fact is true with some probability $p$, and (2) how much the adversary actually learns from the results depends on his prior beliefs.

To see the first point, consider a situation in which an AS $\beta$ suspects a SYN flood on `cnn.com` and therefore asks AS $\alpha$ how many of its customers have sent a TCP SYN packet to `cnn.com` recently. Suppose that AS $\alpha$ has $10,000$ customers and the answer (including noise) is $6,384$. If the adversary learns about this, he can infer that about $63.8\%$ of $\alpha$'s customers have visited `cnn.com`. Thus, if the adversary has prior knowledge that some individual $X$ is a customer of $\alpha$, he can deduce that there is a good chance that $X$ has visited `cnn.com` as well. However, differential privacy limits how much the adversary can learn about the behavior of $X$ *specifically*. Thus, $X$ has plausible deniability – he can always claim to be one of the $3,616$ customers who did not send a SYN to `cnn.com`.

To see the second point, consider an incredibly powerful adversary who already knows for each of the $9,999$ other customers whether or not they visited `cnn.com` but is not sure about $X$. If there are $6,383$ other visitors and the answer to the query is $6,384$

including noise, the attacker can increase his confidence about $X$ a little by applying Bayes' rule. However, the absolute increase in confidence is still limited by the privacy budgets $\varepsilon_i$. This is a strong result, since $X$'s privacy is protected even against such a powerful attacker. However, the fact that the increase in knowledge depends on prior beliefs makes the guarantee hard to understand. If we could state the guarantee in a simpler but perhaps more conservative way, this would make it easier for ASes to justify the query mechanism to their customers.

## 6.4 Setting a privacy budget

An AS can choose the relative magnitudes of its privacy budgets $\varepsilon_i$ according to how much it trusts the corresponding ASes; however, it is not obvious how to choose a good absolute value for the total budget $\varepsilon = \sum_i \varepsilon_i$. There is obviously a trade off between privacy and usefulness, and we would like to choose a $\varepsilon$ low enough to satisfy the privacy needs of the end-users but large enough that we can extract useful information from our queries. In particular, if we expect that the results of queries will grow approximately linearly with the number $N$ of end-users in the database, then we could get useful results even if the noise was proportional to the number of end-users. In particular, even if we choose $\varepsilon$ to be as low as $1/N$, Theorem 2.1 tells us that the noise $\sigma = \eta/\varepsilon \sim N\eta$ will be acceptable in this model. This is a great result for large ISPs, since it will result in very small values of $\varepsilon$ – that is, very private queries!

# 7. CONCLUSION AND NEXT STEPS

We have given a detailed analysis of the challenges and opportunities of applying differential privacy in a networked setting to support collaborative detection and mitigation of botnets. We sketched a new programing language and typing system that allow one to write flexible queries and to track the privacy loss of the results, and we have presented a preliminary model of a novel coordination system that would permit separate administrative domains to collaborate with consideration to privacy. We have also identified fundamental challenges to address in future work; in particular, the handling of streaming data and continuous collection, as well as managing duplicated data that may be stored across multiple domains.

Detecting, observing, and measuring botnet activity is only the first step in any defense campaign. The end goal is to mitigate and disrupt the attacks of botnets or disable them entirely. Exploring connections between our framework and known mitigation techniques such as pushback [18] may be valuable.

# References


[1] P. Bächer, T. Holz, M. Kötter, and G. Wicherski. Know your enemy: Tracking botnets, 2005. `http://honeynet.org/papers/bots`.

[2] J. R. Binkley. An algorithm for anomaly-based botnet detection. In *Proc. SRUTI*, pages 43–48, 2006.

[3] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the SuLQ framework. In *Proc. ACM PODS*, 2005.

[4] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *Proc. STOC*, pages 609–618, 2008.

[5] S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity analysis of programs. *SIGPLAN Not.*, 45(1):57–70, 2010.

[6] C.-M. Cheng, H. T. Kung, and K.-S. Tan. Use of spectral analysis in defense against DoS attacks. In *Proc. IEEE GLOBECOM*, volume 3, pages 2143–2148, 2002.

[7] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proc. SRUTI*, July 2005.

[8] C. Dwork. Differential privacy. In *Proc. ICALP*, 2006.

[9] C. Dwork. Differential privacy: A survey of results. In *Proc. TAMC*, 2008. Invited paper.

[10] C. Dwork. The differential privacy frontier (extended abstract). In *Theory of Cryptography*, Lecture Notes in Computer Science, chapter 29, pages 496–502. Springer, 2009.

[11] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, 2006.

[12] J. Goebel and T. Holz. *Rishi*: Idneitfy bot contaminated host by IRC nickname evaluation. In *Proc. HotBots*, April 2007.

[13] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. USENIX Security*, July 2008.

[14] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through IDS-driven dialog crrelation. In *Proc. USENIX Security*, Aug. 2007.

[15] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proc. NDSS*, February 2008.

[16] A. Gupta, K. Ligett, F. McSherry, A. Roth, and K. Talwar. Differentially private combinatorial optimization, Nov 2009. `http://arxiv.org/abs/0903.4510`.

[17] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting fast-flux service networks. In *Network and Distributed System Security (NDSS)*, 2008.

[18] J. Ioannidis and S. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proc. NDSS*, volume 2. Citeseer, 2002.

[19] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: an empirical analysis of spam marketing conversion. In *Proc. ACM CCS*, pages 3–14, 2008.

[20] A. Karasardis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proc. HotBots*, 2007.

[21] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? In *Proc. FOCS*, pages 531–540, October 2008.

[22] C. Livads, R. Walsh, D. Lapsley, and W. Strayer. Using machine learning techniques to identify botnet traffic. In *2nd IEEE LCN Workshop on Network Security (WoNS '06)*, 2006.

[23] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Proc. FOCS*, pages 94–103, 2007.

[24] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proc. SIGMOD*, pages 19–30, 2009.

[25] D. Moore, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. In *Proc. USENIX Security*, 2001.

[26] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proc. IEEE Security and Privacy*, 2008.

[27] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proc. STOC*, pages 75–84, 2007.

[28] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proc. ACM CCS*, pages 635–647, 2009.

[29] T. Yen and M. K. Reiter. *Traffic Aggregation for Malware Detection*, volume 5137, pages 207–227. LNCS Springer Berlin / Heidelberg, 2008.