Privacy-Safe Network Trace Sharing via Secure Queries

Jelena Mirkovic
USC Information Sciences Institute
4676 Admiralty Way ste 1001
Marina Del Rey, CA 90292, USA
sunshine@isi.edu

ABSTRACT

Privacy concerns relating to sharing network traces have traditionally been handled via sanitization, which includes removal of sensitive data and IP address anonymization. We argue that sanitization is a poor solution for data sharing that offers insufficient research utility to users and poor privacy guarantees to data providers.

We claim that a better balance in the utility/privacy tradeoff, inherent to network data sharing, can be achieved via a new paradigm we propose: secure queries. In this paradigm, a data owner publishes a query language and an online portal, allowing researchers to submit sets of queries to be run on data. Only certain operations are allowed on certain data fields, and in specific contexts. Query restriction is achieved via the provider's privacy policy, and enforced by the language's interpreter. Query results, returned to researchers, consist of aggregate information such as counts, histograms, distributions, etc. and not of individual packets. We discuss why secure queries provide higher privacy guarantees and higher research utility than sanitization, and present a design of the secure query language and a privacy policy.

Categories and Subject Descriptors: H.3 Information Storage and Retrieval: Data sharing
General Terms: Legal aspects, security, standardization.
Keywords: Network traces, sharing, sanitization, privacy.

1. MOTIVATION

Network packet traces are essential for data-mining and validation in networking research. Sharing network data is necessary to provide researchers with diverse data sets, but it creates a lot of privacy risk to data providers. This utility/privacy tradeoff is the crux of network data sharing. *Research utility* of data is difficult to quantify since it depends on the specifics of research questions for which the data is used. In general, a useful trace contains a lot of information, which correlates with the amount of data fields that are released in their original, unmodified version and collection duration. Also, each trace is representative only of the traffic at its collection location, thus an ISP trace cannot be used to validate end-network solutions.

Privacy risk from data sharing is also difficult to quantify, because the information contained in packet traces cannot be fully enumerated. Some data fields are known to pose a privacy risk, because they contain user, host or network-identifying information, private data such as passwords, or can be used to infer the collection network's private data, such as topology, operating systems on hosts, open ports, etc. A straightforward approach to handle known risk is to remove or obscure information in privacy-sensitive fields. But, a lot of privacy risk stems from yet unknown threats. Data that is considered privacy-safe today may correlate with some privacy sensitive information; this can be exploited by future threats. Another problem are attacks that use auxiliary information to infer privacy-sensitive content. For example, an attacker that knows a host X has a rare port Y open can identify this host in the trace and extract information about its other open ports. Dwork et al. showed that absolute privacy cannot be achieved in presence of auxiliary information [1].

1.1 Attack Classes

A *passive* attacker can observe a publicly released trace and use some source of auxiliary information, such as published Web material [2], to infer privacy-sensitive data.

An *active attacker*, can inject traffic into a trace during the collection time, and identify it in the public release. This provides an effective auxiliary information channel that can then be misused in a passive attack.

1.2 Trace Sanitization and Its Drawbacks

Utility/privacy tradeoff in network trace sharing is commonly addressed via *trace sanitization*. It removes or anonymizes privacy-sensitive packet fields, such as contents and IP addresses. The resulting "one size fits all" trace is released to the public. We argue that data release via sanitization is an inherently bad idea that does not address well either utility or privacy aspects of trace sharing.

Sanitization severely decreases **utility** because a data field found to be privacy-sensitive in any one context is removed from or obscured in the entire trace. This disables many research contexts that would not use given data in a privacy-sensitive way. For example, packet length can be used to identify Web pages visited by a user [3]. This attack is possible only when all reply packets are observed within one TCP connection with a Web server, or when this data is summarized in a `NetFlow` trace [4]. Removing packet length from

NDA'08, October 31, 2008, Fairfax, Virginia, USA.

a trace, or only from TCP packets from Web port 80, solves the privacy risk. But it also disables innocent research, e.g., about packet length distribution on the Internet, that never accesses packet length field in the risky context.

Table 1 shows the overview of papers that were published in SIGCOMM and the Internet Measurement Conference, in 2006 and 2007. We examined each paper and classified it as using a public, a private or no traffic trace. A paper could use a mix of private and public traces, in which case it was counted in both categories. Out of 144 papers total 49 used some traffic trace, but only 10 used a public trace and the rest used privately collected traces. This shows that researchers prefer to use a private traffic trace (if they have an access to one) instead of a public trace. A likely reason for this is lower research utility of public traces.

	SIG 06	IMC 06	SIG 07	IMC 07
Total papers	37	34	35	38
Used a traffic trace	9	15	10	15
Used a public trace	2	1	6	1
Used a private trace	8	14	6	14

Table 1: Trace usage in SIGCOMM and IMC papers.

Sanitization further offers very low **privacy** guarantees for several reasons. First, many known passive attacks are not currently handled through sanitization because this would severely reduce resulting trace utility. Examples of this include attacks that infer Web page identity from packet length [3, 4], attacks that use packet timing to identify physical hosts [5, 6], and attacks that use anonymized address clustering and behavior modeling to break anonymization [2]. A trace with obscured or deleted packet length and packet timing, and with randomly anonymized addresses (to prevent clustering) would be useless for many researchers!

Second, no sanitization can defeat active attacks, where an attacker injects well-crafted packets during trace collection time, and identifies them in the sanitized trace thus breaking the anonymization. Attackers can always craft packets that data providers cannot detect — this is an instance of a covert channel problem, which itself is intractable. We note that privacy risk does not really lie in the attacker's ability to inject traffic or to later locate it in the trace. Instead, it lies in the fact that a lot of information about hosts can be mined from a sanitized trace, such as operating system, active services, communication patterns, etc., which is desired for research. Active attacks simply enable an attacker to link this rich information with a real IP.

Finally, sanitization offers poor protection against **future attacks**. Much information about hosts and data provider's network is revealed in a sanitized trace. Each user has access to all the trace's information, even if she needs to mine only a small portion of it. Data providers have no control over trace usage once it is downloaded by a user. Attackers can thus mine trace information at will, and design future attacks against anonymization. Once an effective attack is published *all* past users can apply it.

1.3 New Paradigm: Secure Queries

We explore a novel direction to address the privacy/utility tradeoff of trace sharing: secure queries on original traces under provider's control. Data provider publishes a query language and an online portal, allowing researchers to submit sets of queries to be run on data. The provider's privacy policy restricts queries on some data fields and in some contexts. This facilitates precise specification of situations that may lead to private data leakage in presence of passive and active attacks, and maximizes trace's research utility. Queries are analyzed against the privacy policy and permitted queries are run. The results, returned to researchers, consist of aggregate information such as counts, histograms, distributions, etc. and not of individual packets. Such information is still very useful to researchers – majority of published papers present information mined from network traces via graphs or tables containing aggregate measures.

Trace queries were originally proposed in [7], but they required human verification for privacy leaks. We introduce *secure queries* that are automatically verified via the language interpreter. There is extensive work in databases on privacy-preserving sharing of data (Section 4). We build on this work, which is not done by previous trace sharing approaches.

1.3.1 Advantages over Sanitization

Secure queries address utility/privacy tradeoff much better than sanitization. On the **utility** side, secure queries show a potential to reveal more data to researchers. Packet fields that were removed in sanitization, such as application headers or even contents, can be processed securely using queries. Fine-grain control via query language enables processing of many fields in the application header, and even hashing of sensitive application content, while satisfying the provider's privacy concerns. It also enables precise sanctioning of queries only in contexts that pose a known risk, reducing information loss. A high-level query language will further allow users to more easily specify data-mining tasks, instead of writing low-level parsers. This simplifies development, debugging and verification.

Secure queries protect provider's **privacy** better than sanitization. First, they offer fine-grained control to data providers over the use of their traces. Passive attacks that cannot be handled via sanitization, because of research utility loss, can be handled via the privacy policy that precisely identifies the risky context when data should be obscured. Second, since all data access occurs via a provider-owned portal, the provider can log and audit all queries, thus learning about usage patterns and possible misuse attempts. Many active attacks can be handled via a combination of query restriction, result set restriction (forbidding queries that relate to small host sets), and auditing.

Finally, **future attacks** can be handled by adding new rules into the privacy policy. Only past users that have run now-forbidden queries may apply discovered attacks, unlike the sanitization case where all past users may do so. Because networking research is diverse, percentage of users that have run any specific query should be low. Moreover they can all be identified via query logs.

1.3.2 Discussion

We now openly address some difficult questions.

Many researchers need access to raw packets, and our paradigm forbids that. Lots of researchers need to examine packets in the exploratory stages of their research, when they investigate a new phenomenon. Packet access enables them to spot regularities or anomalies and improve their hypothesis but it creates large privacy risk. Exploratory research will

need to be performed on private traces, until it advances to the point that it has a hypothesis that can benefit from aggregate results shown by secure queries. A data provider may also choose to approve access to a sanitized version of his traces to trusted researchers.

Further, some researchers may need packet access because they are designing traffic replay tools and need to feed raw traces into these tools. These researchers will never benefit from secure queries but they represent a small percentage of the overall networking community.

Can you foresee all the query types that users will need? No, and likely some users will need an operation that is not yet supported. There are many ways to handle addition of new keywords and operations to the language, and we opt for the open-source approach. Data provider releases language interpreter as an open source and users modify it to implement desired operations, test them on private traces, and petition the provider for an update. Interpreter updates then need to be manually examined by providers and applied to the version installed on their portal.

If secure query paradigm were widely accepted it would be desirable to have the single, standard query language. Language updates could be reconciled with the standard in the following way. The standard could be maintained by a community of data providers. All updates accepted by providers could be discussed in the community and those that are deemed useful and privacy-safe by the majority would result in the standard update. Providers opposed to new language constructs can prohibit them via their privacy policy.

Can you prove that no private information is leaked by secure queries? We cannot. It is impossible to prove that no data is leaked by any protection mechanism, save for complete refusal to share anything. However, we show that certain attack classes, currently known to pose privacy risk, are handled by the security policy we propose. It is possible that in our analysis we missed some attacks that will occur to readers. When that happens, we are confident that a privacy policy can be designed that prevents them. Future will also bring new, effective attacks on any approach to data sharing that our current policy does not handle. But secure queries are better suited to deal with those attacks, via policy modifications, than is sanitization.

1.4 Paper Overview

We summarize network data sharing via sanitization in Section 2. We propose a secure query language and a privacy policy in Section 3 and discuss their privacy guarantees. Lessons learned from database privacy are summarized in Section 4, and related work on trace sharing in Section 5. Section 6 offers a brief conclusion.

2. TRACE SANITIZATION

Trace sanitization is commonly done by removing packet contents and anonymizing source and destination addresses. The following anonymization approaches were developed.

Positional anonymization [8] replaces each address in the trace with a number indicating its order of appearance. This results in inconsistent mapping of any address across traces, which can be amended by developing a map dictionary. When anonymizing multiple traces, dictionary entries depend on the order of file processing.

Cryptographic anonymization [9] replaces each address in the trace with an encrypted value. If the same key is used to anonymize multiple traces, a given address will always map to the same anonymized value.

Prefix-preserving anonymization [9] is cryptographic anonymization applied to portions of the IP separately, so that hosts that share a prefix in the original trace continue to do so in the anonymized trace. Prefix-sharing information is used by researchers to detect hosts that belong to the same network.

2.1 Attacks on Sanitization

Known attacks on sanitization focus on retrieving the original IP addresses from the anonymized trace, and using this information to determine communication patterns for the identified hosts, or to discover some features (e.g., open ports) that can be used to attack these hosts in the future. If the anonymization is prefix-preserving, a discovery of a single original IP means the discovery of portions of other IPs that share a prefix with it [9]. We now present an overview of known attacks and name them for easier reading.

The *Web Page attack* [3, 4] identifies Web pages based on the number and length of objects in Web replies.

The *Clock Skew attack* identifies a host by calculating the skew between the packet sender's clock and some referent clock, and uses it to uniquely identify physical hosts. One way to calculate the skew is to observe the evolution of the difference between the sender's clock, recorded in TCP timestamps, and the collection machine's clock, recorded in packet capture time [5]. Another way is to select multiple packets that represent some automated activity, known to happen at intervals of size T, perform a Fourier transform on packet capture times and infer the skew as the difference between the measured frequency f' and the intended frequency $f = 1/T$.

In [2] authors use link-layer header data to infer network topology, address clustering to detect subnets of IPs anonymized in prefix-preserving manner, and behavior models of popular, known servers to break their anonymization. We refer to these attacks as *Link Layer attack*, *Clustering attack* and *Behavior attack*. Each attack sets off a chain of inference since one deanonymized address brings information about other addresses with the same prefix, and together with topology yields much of sensitive internal information.

The *Scan attack* [6] uses the fact that port scans from a single machine frequently target IPs in order. Observing a scan sequence from the same source in a sanitized trace, an attacker can infer the relationship between destination addresses. Identifying any of the addresses in the sequence then reveals the entire sequence.

There is an infinite number of active attacks because an attacker can use any combination of packet fields and timing to generate injection traffic he can easily spot, but that seems inconspicuous to the data provider. We consider all active attacks as instances of this general class we seek to handle.

3. SECURE QUERIES

We now propose a design of a secure query language, which we call *Trol* and a sample privacy policy.

3.1 Trol Language

While we plan to develop Trol operations for application headers and contents, as the first step we focus on processing IP and transport header fields only. Trol is not yet fully implemented, but we present here its design, which is complete. Trol's grammar is shown in Figure 1 in `yacc` format.

operations: operations operation
| operation

operation: INIT QUOTE NAME QUOTE
| SELECT WHERE condition
| GROUP BY condition START condition END condition dups
| FOREACH GRP NAME DO operations DONE
| FOREACH PKT NAME DO operations DONE
| SORT BY feature direction
| IF condition THEN operations FI
| IF condition THEN operations ELSE operations FI
| var ASSIGN aexpression
| OUTPUT aggregates

condition: /* empty */
| rexpression
| NOT condition
| LPAREN condition RPAREN
| condition AND condition
| condition OR condition

rexpression: aexpression EQ aexpression
| aexpression NE aexpression
| aexpression LT aexpression
| aexpression GT aexpression
| aexpression LE aexpression
| aexpression GE aexpression

aexpression: mexpression
| aexpression PLUS mexpression
| aexpression MINUS mexpression

mexpression: pexpression
| mexpression MULT pexpression
| mexpression DIV pexpression

pexpression: LPAREN aexpression RPAREN
| rhs
| NUMBER
| QUOTE NAME QUOTE
| NAME

utilityf: HASH LPAREN feature RPAREN
| SHARESUBNET LPAREN feature COMMA feature COMMA NUMBER RPAREN
| ISCLOSE LPAREN feature COMMA feature COMMA NUMBER RPAREN
| ISCOMMERCIAL LPAREN feature RPAREN
| ISACADEMIC LPAREN feature RPAREN
| ISISP LPAREN feature RPAREN
| BIN LPAREN feature RPAREN

elem: PKT
| GRP

aggregates: COUNT elem
| COUNT UNIQUE rhs
| PDF XAXIS rhs
| CDF XAXIS rhs
| HIST XAXIS rhs

rhs: datafield
| utilityf
| var
| lead datafield
| lead utilityf

var: svar
| lead svar

svar: NAME
| NAME LBRACKET var RBRACKET
| NAME LBRACKET NUMBER RBRACKET

lead: FIRST DOT
| LAST DOT
| NAME DOT
| NAME DOT LAST DOT
| NAME DOT FIRST DOT
| GRP DOT
| GRP DOT FIRST DOT
| GRP DOT LAST DOT

dups: /* empty */
| DUPLICATES

direction: INCREASING
| DECREASING

datafield: DATALEN
| CAPTIME
| IPVER
| IPIHL
| IPTOS
| IPTOTLEN
| IPID
| IPDF
| IPMF
| IPOFFSET
| IPTTL
| IPPROTO
| IPCHECK
| IPSRC
| IPDST
| TCPSRC
| TCPDST
| TCPSEQNUM
| TCPACKNUM
| TCPOFFSET
| TCPFLAGS
| TCPWIN
| TCPCHECK
| TCPURGPTR
| TCPTIMESTAMP
| UDPSRC
| UDPDST
| UDPLEN
| UDPCHECK
| ICMPTYPE
| ICMPCODE
| ICMPCHECK

Figure 1: Trol grammar.

For space reasons we do not show token declarations but they should be obvious from the grammar.

INIT operation loads trace data from a file in `pcap` format into the *working set*, which is an array of packet structures. Multiple INIT operations can be used to enlarge the working set. All other operations manipulate this set.

SELECT operation reduces the working set only to packets that meet a given condition. This is a permanent change, meaning that operations following a SELECT occur on a reduced set. A condition may be empty which defaults to "true", or it could be a combination of relational expressions and logical operators.

GROUP operation groups packets in the working set by some similarity condition specified after BY keyword. Internally, the working set is partitioned into a set of *packet arrays*, each holding members of one group. After this operation, packets that do not belong to a group are removed from the working set. Conditions after START and STOP keywords describe criteria a packet must match to be the first or the last packet in a group. DUPLICATES keyword may follow, denoting that repeated packets that match the similarity condition should be added to the group. Features of the first and the last packet in the group can be accessed by using keywords "first" and "last" in front of a name of any packet feature. Figure 2 shows a sample Trol script that groups all TCP packets into connections. A new connection starts with a SYN packet or after a 60-second pause in an existing connection. A connection ends with an ACK packet following a FIN-ACK packet, with a RST packet or after a 60-second pause. Repeated packets will be included in the connection.

GROUP statements can be nested, e.g., one could group packets per host, then group all host's traffic by connection. Internally, each nested GROUP statement transforms an existing packet array, representing a current group, into an array of packet arrays. This brings up an issue of the target of statements following a GROUP statement. It is conceivable that a user may want to apply some of the following operations on each group separately, and others on the entire working set. In case of nested GROUP statements the user must be able to refer to each grouping level and manipulate it. FOREACH statement helps disambiguate these cases by allowing explicit iteration over groups at the same level as this statement, and iteration over packets in a group. It also allows naming of the current group or packet so they can be manipulated with statements enclosed between DO and DONE as part of the FOREACH statement. Users can further declare and manipulate *group variables* that have an instance for each group. Within the body of a FOREACH statement this is done by prefacing a variable name with the group's name. Outside of a FOREACH statement, the keyword *group* can be used to refer to groups at the same level as this statement. Figure 3 shows code that could follow statements from Figure 2 to calculate total traffic in a connection, and then select connections that had more than 1,000 bytes.

SORT statement orders packets in the working set or packets within a group in increasing or decreasing order of the specified feature. IF-THEN-ELSE statements enable selective application of operations depending on some condition. ASSIGN statements assign a value calculated in an arithmetic expression to a left-hand-side variable.

```
SELECT WHERE ip.proto == "tcp"
GROUP BY (first.ip.src == ip.src OR first.ip.src == ip.dst)
        AND (first.tcp.src == tcp.src OR first.tcp.src == tcp.dst)
        START (tcp.flags == "S" OR captime - last.captime < 60)
        END ((tcp.flags == "A" AND last.tcp.flags == "AF")
        OR tcp.flags == "R" OR captime - last.captime > 60) DUPLICATES
```

Figure 2: Grouping TCP packets into connections.

```
FOREACH group G DO
G.sum = 0
  FOREACH pkt P DO
      G.sum = G.sum + P.datalen
  DONE
DONE
SELECT WHERE group.sum > 1000
```

Figure 3: Selecting connections with more than 1,000 bytes.

OUTPUT statements display a selected aggregate statistic, calculated on the working set or over groups, to the user. The statistic can be a count, count of unique values of a certain feature, a probability density function, a cumulative distribution function or a histogram. For the last three options, a user can choose a feature to be shown on the x-axis. For example, "OUTPUT HIST XAXIS group.sum" appended to code in Figures 2+3 outputs a histogram of connections that had more than 1,000 bytes, with connection size on x-axis.

Arithmetical, logical and relational operations in Trol are defined as usual. They manipulate strings, numbers, packet fields, scalar, array or group variables, or outputs of utility functions. Packet fields include standard IP, TCP, UDP and ICMP packet header fields, the packet capture time and size of packet contents. Utility functions serve to extract useful informations from privacy-sensitive data in a safe manner and include: (1) hashing a value, (2) binning a variable by value, (3) operations that test if an IP address belongs to a commercial entity, an academic institution or an ISP, (4) operations that test if two IPs belong to the same subnet of specific length, or (5) if they are geographically closer than a certain distance. We explain the binning operation in Section 3.4.

There are many useful operations that could be added to Trol grammar. We decided to start with a small set of common operations that we could analyze and implement, then extend it as we proceed.

3.2 Privacy Policy Specification

A provider-defined privacy policy, along with the Trol interpreter that analyzes submitted queries based on the policy, restricts certain operations that may reveal privacy sensitive data. Some restrictions are enforced at the parsing stage, and others during execution or at the display stage.

3.3 Privacy Requirements

The following set of privacy requirements is frequently adopted by today's trace data providers, and in this section we show how the privacy policy is extracted from the requirements and enforced. It is likely that many data providers will need to further customize these requirements and the proposed privacy policy to fit their organizational policy [6].

The privacy risk to the data provider from sharing the trace ***should not greatly increase*** *with respect to the attacker learning the following information: (1) Packet contents, (2) Presence or absence of a host in the trace, (3) Presence or absence of communication between two hosts, (4) Open ports or OS type and version per host. Learning this information for the entire organization is acceptable as long as it cannot be paired to a specific host.*

Our formulation of the privacy requirement relies directly on Dwork's definition of *differential privacy* [1] for statistical databases. The publication [1] shows that a stronger statement "there is no privacy risk" cannot be supported because an attacker with auxiliary information may retrieve private data from even the smallest piece of shared information, which by itself appears privacy-safe. As we discuss in Section 4 much existing wisdom from database research can be reused in context of trace sharing.

3.4 Linking Requirements to Attacks

We now link privacy requirements to packet fields and operations that would pose privacy risk. Revealing a single packet's content or any IP address is unacceptable according to our requirements 1, 2 and 3. Because of the simple algorithm for checksum calculation, the cheksum field may also leak information about packet contents.

Our privacy policy achieves the desired protection by requiring that any access to fields whose single value could be sensitive occurs via a utility function. Utility functions provide privacy-safe, commonly researched information about sensitive fields. Packet contents and checksum can only be accessed via the hash function, while IP addresses can be accessed via the hash function, and the functions that test the membership of IPs in the subnet, their geographic proximity and their type. To comply with privacy requirements, the GROUP BY condition in Figure 2 should be modified to "(HASH(first.ip.src) == HASH(ip.src) OR HASH(first.ip.src) == HASH(ip.dst))".

Presence or absence of a single or a pair of hosts in the trace can also be inferred from other packet fields via passive or active attacks. Currently known passive attacks that are applicable to Trol queries are the Web Page [3, 4] and the Clock Skew attack [5]. The Link Layer attack [2] does not apply because Trol operations we proposed do not access link-layer data. Assuming all access to IP addresses is via utility functions, the Clustering [2] and the Scan [6] attacks do not apply either. The Behavior attack [2] poses no risk under differential privacy definition because the access to the trace does not significantly increase attacker's knowledge: the attack only reveals existence of known, popular servers in the trace, but due to their popularity the attacker could guess their being in the trace even without data sharing.

Many active attacks can violate our privacy requirements 2 and 3. For example, the attacker could learn communication patterns in the following manner. He spoofs several packets from X to Y transferring total of 123 bytes. Once the trace is published, he uses Trol to group packets per host pair, and ask for count of groups that exchanged 123 bytes. If the result is 0, X and Y talk to each other; if the result is 1 they do not. A result higher than 1 means that connection size 123 is not unique, and the attacker selects another size and repeats the attack. Active attacks can also violate our privacy requirement 4. For example the attacker can learn open ports on host X by sending a packet to X to some rarely used destination port, e.g., 78. Once trace is published, he uses Trol to group packets per destination IP, and display a histogram of destination port numbers per group. If port 78 is only seen in attacker's probes, host X is the only host whose histogram

will show this, and all other open ports. Otherwise, the attacker chooses another port number and repeats the attack.

The above discussion illustrates that any information that is unique to a single host or host pair is potentially privacy sensitive because it could be used to identify it. In both passive and active attacks the unique information is known to the attacker via an auxiliary channel such as interaction with the host or spoofing a packet. But passive attacks do not insert attacker's traffic into the trace, i.e., they search for the uniquely identifying information that is already present, while active attacks insert traffic to create uniquely identifying information and use it to pinpoint hosts or pairs of interest.

To deal with active and passive attacks it is critical that any piece of information presented to a user relates to a group of hosts and diverse pairs, both in source and in destination dimension. This is a known principle in database privacy of "result set restriction" aka "hiding in a crowd". In statistical databases each row refers to one person, and result set restriction is easily applied by requiring that any query must reference no less than L rows and no more than N-L rows, where N is the size of the database and L is the threshold set by the administrator. Requiring result sets of at least size L ensures hiding in the crowd in the case of a positive query – asking directly for information of interest. Requiring sets of at most size N-L ensures hiding when a query is negative – asking for an opposite information than that of interest. For example, assume an attacker knows that clock skew of host X is a, and he seeks to learn if this host is in the trace. He groups packets per host and calculates clock skew for each group in variable cs. A positive query is `SELECT WHERE group.cs == a`, with result size 1 proving the presence of host X and result size 0 proving its absence. A negative query is `SELECT WHERE group.cs != a`, with result size N-1 proving the presence and N proving the absence.

Implementation of result set restriction in Trol is more complex than that in statistical databases, because Trol's expressiveness enables users to define, manipulate and display custom variables. A simple approach would associate with each variable a set of source and destination IPs whose traffic was used to calculate the variable. This would have too large a memory cost and would limit scalability. We propose the following solution.

(1) Forbid IF statements except within a "FOREACH group" statement. This ensures that a user can group packets per host or per pair of hosts only via the GROUP BY statement.

(2) After a GROUP BY statement, the Trol interpreter creates a source IP and a destination IP table of size L for each group, and crawls packets within the group to populate the tables. A group that has less than L members in either table is marked as *tainted* and its table is preserved. Otherwise, the group is *untainted* and the tables are deleted. It may appear that to prevent negative queries we must also check for too large tables, but this would be too expensive. The next check achieves the same goal at a lower cost.

(3) After a GROUP BY or a SELECT statement Trol analyzes either the packets left in the working set or the deleted packets, whichever smaller, and populates new source IP and destination IP tables of size L. If either table does not fill, the result set restriction was violated and the query is rejected.

(4) A group variable is tainted if any group that calculates an instance of this variable is tainted. No source or destination IP tables are needed since this information is already remembered for the entire group.

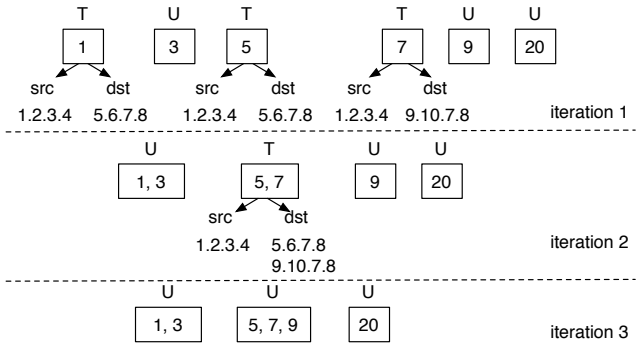

Figure 4: Illustration of binning.

(5) In output statements tainted variables can only be accessed via the BIN utility function. This function orders all variable instances in increasing order of their value, then places them into bins. If the variable was tainted its bin is also marked as such, and its source and destination IP tables are associated with the bin. Neighboring bins are merged if at least one of them is tainted, and merge candidates are selected using the nearest neighbor principle on a bin's average value. If two neighbors are equally close, the bin with fewer members is selected for the merge. The merge combines the source IP and destination IP tables from original bins. After the merge, if both tables have L or more members the bin is marked as untainted. Merging continues until there are no tainted bins left.

We illustrate binning in Figure 4, and refer to bins by values of some tainted variable v they store. Let L=2. In the first iteration, there are six bins with three of them (1, 5 and 7) being tainted. In the second iteration, bins 1 and 3 are merged, which results in untainting because bin 3 was untainted. Bins 5 and 7 are also merged, but the result is still tainted because it refers to only one source. In the third iteration bin 5, 7 merges with closest bin 9. The resulting bin is untainted and the binning stops.

(6) Variables which were defined outside of a group statement, may change their value inside the statement through direct calculation only if all the variables on the right hand side of the calculation are either untainted or accessed via the BIN function, which returns the average of values in the bin containing the variable instance. If an outside variable changes its value within an IF statement, the same restriction holds for the condition of that IF statement. For example, assume an attacker knows that clock skew of host X is 55. Code shown in Figure 5 would reveal the presence or the absence of host X in the trace via the value of the variable c. To prevent this, *G.skew* must be binned in the IF statement.

To minimize overhead, binning is performed once per vari-

```
c = 0
FOREACH group G DO
  calculate G.skew
  IF G.skew == 55 THEN
     c = 1
  FI
DONE
```

Figure 5: Privacy risk without binning.

```
RESTRICT RESULT ip.src > 2 AND ip.dst > 2
ACCESS ip.src, ip.dst VIA hash, shareSubnet, isClose, isCommercial,
                          isAcademic, isISP
ACCESS ip.cont, ip.check, udp.check, tcp.check VIA hash
```

Figure 6: Privacy policy for our requirement set.

able. This requires that group operations be processed breadth-first (across groups) instead of depth-first (within each group).

We now show that binning and utility functions provide protection against most common attack classes. Consider an attacker that seeks some uniquely identifying information in the trace, through active or passive means. Use of utility functions prevents attacks that directly access privacy-sensitive data, where revealing even the smallest portion of this data, such as the first octet of an IP address, may increase provider's risk. Binning alone cannot handle these attacks because parts of sensitive information would still be revealed when binned, and because binning IP addresses, contents or checksums would destroy research utility. The rest of attacks use auxiliary knowledge to infer uniquely identifying information or link it to privacy-sensitive data. Binning prevents inference of any information that does not relate to a large enough group of hosts, thus handling most of active and passive attacks. What remains are two classes of attacks that can infer desired information from results that relate to large groups of hosts. The first class, known as *trackers* [10], asks successive queries that narrow down the information of interest but always relate to a large enough group. For example, if the uniquely identifying information is connection size of 123 the first query may ask for count of all connections with size >123, and the second would either count connections with size > 122 or connections with size < 123. A known protection from trackers [10] is to log result sets and audit them either for a large overlap or for a small disjoint set. Audits can be per user or for all users. We plan to implement at least per-user audits in the Trol interpreter, but we leave their design for future work. The second class of attacks, we call *fakers*, injects identifying information for spoofed, non-existing IPs to form a large enough result set with the desired unique identifier. For example, if L=2 and the attacker looks for the clock skew of 55 he inserts a fake host with clock skew 55 into the trace by spoofing its traffic. Queries that relate to clock skew 55 either have a large enough result set, signaling the presence of a real host with clock skew 55, or are rejected, signaling the host's absence. Investigating how to handle fakers is part of our future work.

3.5 Privacy Policy

The privacy policy specification language has two types of statements: (1) Result set restriction specifications that list packet fields that must have L unique values in the result set. (2) Operation restriction specifications that deny access to a list of packet fields and variables except via the list of utility functions. Figure 6 shows the privacy policy for our chosen set of privacy requirements, assuming the result set size of 2. A data provider that wants to further restrict its privacy policy would add ACCESS statements to limit operations on fields she regards as privacy-sensitive, possibly also adding new utility functions to allow some limited access to these fields.

4. LESSONS FROM DATABASES

Databases have faced similar problems related to sharing non-private data from a database of patient visits to a hospital or from census data [11, 12, 13]. Solutions to privacy/utility tradeoff can broadly be classified as sanitized databases or statistical databases. We now summarize similarities and differences between sharing database and trace data.

(1) Both database and trace records contain some identifying information (e.g., a person's name vs packet's source and destination IPs), and some sensitive information (e.g., a person's illness vs packet's contents). Statistical databases remove identifying information and allow aggregate operations on sensitive fields, but prevent disclosure of any individual field values. Sanitized databases disclose sensitive fields but remove identifying information. Trace sharing usually removes or obscures sensitive fields, obscures identifying information, and discloses non-sensitive fields.

(2) Statistical databases allow only a very small set of aggregate queries, such as max, sum, mean, etc. on all fields. Their utility remains high even if individual cells or outputs to queries are modified or perturbed to protect privacy [10], as long as results to aggregate queries remain close to those obtained on the original data. On the other hand, modifying packet fields or perturbing outputs would greatly reduce trace utility for many users.

(3) Removing identifying data from sanitized databases is not enough, since auxiliary information can be used to infer identity and link a person to sensitive information [1]. Proposed solutions heavily modify released fields to ensure that (a) each person resembles k other people [11], (b) each group of k similar people has l well represented values in sensitive attributes [13] and (c) deletions and insertions do not violate conditions (a) and (b) [12]. Such major data modification would seriously reduce research utility of traffic traces. We believe that our binning, which achieves k-similarity from (a), is sufficient for network traces because of natural traffic diversity. For example, it may happen that some organization opens the port 999 on each Web server. The queries for count of hosts with ports 80 and 999 open, and hosts with port 80 open, would reveal this and pose a privacy risk. But the risk would greatly diminish if there were a single Web server with closed port 999. Based on our experience with traces we intuit but cannot yet prove that any uniformity of host features in network traffic is either rare or well-known (which does not pose a risk under differential privacy).

(4) Secure queries with result set restriction and auditing have been proposed for statistical databases [10]. Their main drawback was that they required auditing to defeat trackers and it was expensive to implement [10]. Based on our trace processing experience to date, we estimate that cost of auditing in the Trol interpreter will be acceptable. For example, backbone traces from MAWI [14] contain less than million IPs each. Thus a million-entry hash table could store information about IPs in the final working sets, and be used to calculate overlaps and set differences. Assuming a collision-free hash function, entries only need to contain bit arrays, with each bit representing presence of an IP in a result set.

5. RELATED WORK

In publication [7] authors discuss a design of a trace processing framework called SC2D, and a high-level query language. Unlike our automated analysis of privacy risk, in

SC2D data providers must manually analyze and approve queries. Publication [7] does not present a language specification to which we could compare our Trol design.

`Tcpdpriv` [15] sanitizes a `tcpdump` trace by transforming it into another trace in the same format, with some fields removed or anonymized. The tool has a wealth of options that enable fine-tuning of the information to be preserved in the sanitized trace. `Ipsumdump` [16] and `Ipaggregate` [17] summarize information from a trace into a human-readable format. Both tools can also perform address anonymization, traffic filtering and traffic sampling, with fine-grained options to select the information to be preserved. In [6] authors examine the problem of trace sanitization in great depth and point out that a straightforward sanitization leaves much information that can be a user privacy or a collection site security risk. They describe a fine-grained sanitization policy developed for their private traces, to facilitate their release to public, and a sanitization tool that applies the policy. In [18] authors investigate the problem of parsing and sanitizing the application content. The sanitization tool assembles the payload to generate application content, applies the policy script to the trace to sanitize it, and converts everything back to the `tcpdump` format. The authors also propose a detailed sanitization policy for FTP data, and discuss many fine-grained decisions to ensure safety from sophisticated attacks. Trol operations for application headers will build on work in [18].

We are aware of two projects that design a high-level environment for online trace *sanitization*. CoMo project [19] builds an open infrastructure for network monitoring. Users are allowed to post customized queries to the system, by providing C-language plug-ins to the CoMo API. The security is regulated through user privileges, so a given user can only post a subset of queries for which he has been authorized, for a set of packet fields. LOBSTER project [20] defines an API for a flexible trace anonymization according to the privileges and the requirements of user applications. The API contains many built-in anonymization functions.

6. CONCLUSION

From current attacks on sanitization and low usage of public traces it is clear that sanitization does not successfully resolve the privacy/utility tradeoff in trace data sharing. In this paper we presented a novel paradigm that involves publishing a query language and a portal that evaluates user queries on original traces. Secure queries have potential to improve both research utility of traces and privacy guarantees for data providers. Much work remains to be done on the query language specification and implementation. One critically important issue is if Trol's query processing will have a reasonable overhead (memory-wise and CPU-wise) for very large trace files. We are currently implementing the Trol interpreter and expect to evaluate its overhead shortly.

7. REFERENCES

[1] Cynthia Dwork. Differential Privacy. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming*, 2006.

[2] S. Coull, C. Wright, F. Monrose, M. Collins, and M. Reiter. Playing Devil's Advocate: Inferring Sensitive Information from Anonymized Network Traces. In *Proceedings of the Network and Distributed System Security Symposium*, February 2007.

[3] Q. Sun, D. R. Simon, Y. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[4] S. Coull, M.P. Collins, C.V. Wright, F. Monrose, and M. Reiter. On Web Browsing Privacy in Anonymized NetFlows. In *Proceedings of the USENIX Security Symposium*, August 2007.

[5] T. Kohno, A. Broido, and kc Claffy. Remote Physical Device Fingerprinting. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2005.

[6] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. The devil and packet trace anonymization. *ACM SIGCOMM Computer Communications Review*, 36(1):29—38, 2006.

[7] J C Mogul and M Arlitt. Sc2d: An alternative to trace anonymization. In *Proceedings of the SIGCOMM 2006 Workshop on Mining Network Data*, 2006.

[8] Vern Paxson. Trace sanitization scripts. `http://ita.ee.lbl.gov/html/contrib/sanitize.html`.

[9] J. Xu, J. Fan, M. H. Ammar, , and S. B. Moon. Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme. In *Proceedings of the IEEE International Conference on Network Protocols*, 2002.

[10] Nabil R. Adam and John C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.

[11] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[12] Xiaokui Xiao and Yufei Tao. M-invariance: towards privacy preserving re-publication of dynamic datasets. In *Proceedings of the International Conference on Management of Data*, 2007.

[13] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. In *Proceedings of the 22nd IEEE International Conference on Data Engineering*, 2006.

[14] MAWI Working Group Traffic Archive. `http://tracer.csl.sony.co.jp/mawi/`.

[15] Greg Minshall. tcpdpriv tool. `http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`.

[16] Eddie Kohler. Ipsumdump tool. `http://www.cs.ucla.edu/~kohler/ipsumdump/`.

[17] Eddie Kohler. Ipaggregate tool. `http://www.cs.ucla.edu/~kohler/ipsumdump/aggcreateman.html`.

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

[19] Gianluca Iannacone. CoMo: An Open Infrastructure for Network Monitoring — Research Agenda. `http://como.intel-research.net/pubs/como.agenda.pdf`.

[20] Lobster web page. `http://www.ist-lobster.org/publications/deliverables/D1.1a.pdf`.