Some Problems in Sanitizing Network Data

Matt Bishop, Rick Crawford, Bhume Bhumiratana, Lisa Clark, Karl Levitt
Dept. of Computer Science
University of California at Davis
Davis, CA 95616-8562
{*bishop*, *crawford*, *bhumirbh*, *clarkl*, *levitt*}*@cs.ucdavis.edu*

Abstract

The problem of removing sensitive information from data before it is released publicly, or turned over to less trusted analysts, underlies much of the unwillingness to share data. The solution is to sanitize, or deidentify, parts of the data. When dealing with network addresses, the set of available addresses is finite. This limits some aspects of the sanitization. We analyze this problem in detail, and suggest approaches to ameliorate it.

1. Introduction

The sanitization problem involves three entities: a *collector*, an *analyst*, and an *adversary*. The *collector* captures data, then gives it to the *analyst* for analysis. The data contains information that must be kept private, such as passwords, information that identifies individuals, and medical records. To keep this data secret, the collector sanitizes the raw data, removing enough information that the sensitive data cannot be determined, before sharing the data with the analyst.

The goal of the *adversary* is to recover as much of the original, raw data as possible. The adversary may do so under various assumptions.

1. The adversary may assume an active role in generating the data. Similar in concept to a known (or chosen) plaintext attack in cryptography, the adversary attacks by creating specific markers in the unsanitized traffic[1]. If these markers remain recognizable after sanitization, then they help the adversary derive the original, raw data from the sanitized data.
2. The adversary may have access to private data, such as organizational information, the role of specific systems, or other sensitive information, that when combined with the sanitized data allows her to deduce the raw information.
3. The adversary may be able to deduce the raw data based on non-sanitized components of the data that are supplied to the analyst. This is the database inference problem in a different context.
4. The adversary has no access to other sources of private information that, when combined with the sanitized data, would enable her to deduce the raw information. In what follows, we assume this case.
5. We also assume that the analyzer operates with full public transparency. The collector makes its sanitized data equally available to the analyzer and the adversary. Many situations relax this requirement by providing the data under non-disclosure or confidentiality agreements. However, even with those agreements in place, it is still possible that various parts of the sanitized data may leak and become known to the adversary. For example, the analyst's network and data may be compromised through no fault of the analyst. Hence we simply assume the worst case.

An obvious question is whether this model is realistic. Consider the situation in which multiple, mutually-distrusting collectors feed data to one mutually-trusted analyst. Since all collectors trust the analyzer never to reveal their sanitized data, then why not simply give their raw data directly to the analyst? Besides guarding against inadvertent leakage by the analyst, legal or contractual requirements may forbid the collectors from sending unsanitized data to the trusted analyzer. Conversely, even if multiple collectors trust each other to keep the data secret, but don't trust the external analyzer with their raw data, the analyst may have resources and expertise that exceed those of the collectors, making the external analysis attractive (or necessary) to the collectors.

The collector's goal is to sanitize the data in a way that maximizes the efficiency and accuracy of the

[1] For example, the attacker can seek to resolve three consecutive unknown addresses, and look for similar data in the sanitized stream.

analyzer's task, while minimizing the efficiency and accuracy of the adversary's attempt to derive the raw data. This paper examines constraints surrounding this goal, in the context of sanitizing network traces.

2. Network Data Sanitization

Researchers need real traffic from networks to further their work in intrusion detection and attack analysis. The problem with using such data is that it typically contains confidential information. Three approaches to this problem have been considered.

First, data can be synthesized. To do this, one monitors a network to gather statistical parameters of the network traffic. One then generates artificial traffic that preserves the statistical parameters of the actual traffic. The problem with this approach is the determination of *which* statistical parameters are relevant. These may not be known before the analysis. An additional issue is the need for specific *types* of traffic, such as attacks, which may exist in the raw data but not be captured by the synthetic data.

Second, one can obtain and use a body of reference data. This is network traffic captured from some other source that has no privacy constraints. Common sources for this type of data are honeypots. The problem is that reference data is not widely available, and is biased in the sense that the traffic patterns and statistical parameters typically do not match normal traffic at other sites.

Third, one can sanitize the data. The nature and degree of sanitization varies from locality to locality. An organization may require that the sanitization prevent any action from being associated with an individual. If many network nodes are single-user systems, this means that MAC and IP addresses, host names, and internal email addresses must be changed or deleted wherever they occur. Thus, one would need to handle packet content as well as packet headers.

For this paper, we focus on the packet headers, and ignore (overwrite) the content. An alternate approach, that of structuring the packet data when possible [1], would allow us to preserve more of the packet content. But sanitizing secondary detail seems premature until the underlying problem structure is better understood.

This naturally leads to sanitizing data link, network (IP), and transport (TCP/UDP) layer headers. In what follows, we focus on IP addresses because they were the key data that needed to be protected in our experiments. But the issues arise in the context of *any* finite set of names (a *namespace*).

The specific privacy policy requires that no IP address be associated with an individual when a network trace is analyzed. In effect, this means that all IP addresses must be sanitized. Even though the communication may have gone outside the local network to a particular recipient, the privacy policy would be violated if that same message were provided to an arbitrary third party.

Two types of data anonymization prove useful. *Pseudo-anonymous* transformations map all instances of a particular raw identifier to the *same* unique identifier in the target namespace. An example is replacing all occurrences of "John" by "Paul". *Fully-anonymous* transformations map each instance of a particular raw identifier to a *different* identifier in the target namespace. An example is replacing the first occurrence of "John" with "Paul", the second "John" with "George", and the third "John" with "Ringo". The advantage of pseudo-anonymous transformations is that an analyst may correlate data related to an identifier without knowing what the raw identifier is. For example, given a set of network traces, an analyst can determine if two connections are between the same hosts. The disadvantage is that the adversary may be able to deduce private information from that knowledge.

Either mapping may be done explicitly, using a table with each raw identifier and its corresponding sanitized identifier, or via hash functions, in which the mapping from sanitized identifier to raw identifier is not preserved. Explicit maps (i.e., tables) are useful when the original data may need to be derived from the sanitized data (for example, inverting it in preparation to be re-sanitized). Hash functions eliminate the need to store a large table of data.

3. Addresses and Namespaces

Sanitizing data raises several potential problems. We focus on three problems that arose during our work because, not only are they crucial for properly sanitizing network traffic, they also present obstacles in many other application domains, due to their finite namespaces and/or semantic properties.

3.1. Properties and IP Address Ranges

Consider the situation in which a single analyzer aggregates data from several different collectors. In some contexts, all the collectors must pseudo-anonymously map certain raw IP addresses to the same target namespace addresses to ensure consistency of identity. Conversely, if the mapping is fully-anonymous, then each collector must scatter its raw IP addresses across different IP target ranges, in order to prevent conflation of two occurrences of any IP addresses to the same target address.

To our knowledge, all prior approaches using hashing to accomplish these results have required that all collectors use the same hash function for the entire namespace. Pseudo-anonymity would require that the IP address be hashed; full-anonymity could be implemented by hashing on a combination of the IP address, the particular collector's unique identifier, and the number of previous times that IP address occurred in that collector's data stream. This is unacceptable for collectors who trust the analyzer but who do not trust each other. The obvious insider attack is to guess an IP address (and, in the case of fully-anonymous sanitization, the ancillary unique data), hash the guess, and compare to the sanitized data. Explicit maps can mitigate this problem by allowing mutually distrusting collectors to share—and hence risk—only portions of a codebook. This feature could also be implemented by binding hash functions to specified input IP address regions, and sharing only some of those hash functions. Both methods allow fine-grained control over privacy risks caused by sharing sanitization functions.

Unlike a generalized hashing algorithm, explicit sanitization maps can preserve namespace properties such as locality. For example, some types of analysis may require that sanitized IP traffic preserve the integrity of certain reserved address ranges. It would not be possible to implement this by first testing an original address to see if it is reserved, and then hashing only the non-reserved addresses. True, the reserved addresses would be preserved, but a generalized hash function would also map many non-reserved addresses into reserved address ranges, thus confusing the analysis.

For namespaces in other problem domains, hashing *per se* could never match the functionality of explicit sanitization maps. In the medical database example, an analyzer might partition all names into three equivalence classes: male names, female names, and a third class of ambiguous names.

Other namespace properties can be preserved by constructing appropriate sanitization maps. For example, if an analyzer recognizes a "sweep" of IP addresses only when an intruder probes an ascending succession of contiguous IP addresses, there are 256 pseudo-anonymous sanitization functions that will preserve such sweep signatures in the low-order byte.

3.2. Implications of Finite Namespaces

Another important aspect of IP addresses as identifiers is that they are drawn from a namespace of finite size. This has two crucial implications. First, any pseudo-anonymous mapping on a finite namespace must be a permutation. Therefore, if a mapping implementation is not parsimonious and "over-reserves" target space for a block of IP addresses in the original namespace, the target namespace will overflow when all possible input names are sanitized.

Second, the pigeonhole principle shows that any fully-anonymous mapping of $n+1$ name occurrences on a namespace of n names is impossible. For example, if a long conversation between two network nodes is to be sanitized fully-anonymously, the target IP address namespace will eventually become exhausted and repetitions of sanitized names will occur. Hence, in the absence of some special controls to distribute the allocation of sanitized names, a simple first-come-first-serve implementation of full-anonymity would protect the privacy of the most verbose early nodes, at the expense of nodes appearing later in the data stream.

This forces the collector to make explicit decisions on namespace resource allocation during configuration. One (or more) small block(s) in the original namespace may be "expanded" and scattered across a much larger block in the target namespace. The collector may map some source blocks pseudo-anonymously, and other source blocks fully-anonymously, into the same target namespace block. Conversely, certain source namespace blocks may be "compressed" to map into a smaller target namespace block. Clearly, if any target namespace region might potentially overflow, then the collector should specify overflow-handling policies for all blocks that map to that region.

In general, by instrumenting fully-anonymous sanitizing routines with appropriate overflow-handlers, a collector can implement anonymization functions on a per-block basis that span the spectrum between pseudo- and full-anonymization. To use a medical database example, successive instances of "Bob" might map to "Bill", "Bo", or "Bruce". Successive assignments might be random, or might cycle within the target block's choices. In contrast, to give "Alice" more privacy protection, the collector might allocate more than three target pseudonyms to her.

The preceding discussion of allocation issues assumes sanitization is to be done in a single pass, even for non-realtime contexts. But sanitizing data in multiple stages allows many other approaches. For example, even the constraint of a finite namespace can be mitigated by first sanitizing to a larger target namespace, then optimizing a subsequent namespace compression stage to minimize analyzer errors caused by duplicate (i.e., conflated) names [8].

3.3. Namespace Nuances

A sanitization procedure must be sensitive to the semantics of its input data. For example, compare the problem of sanitizing data in a medical database to that of sanitizing network packets. In both cases, the goal is to protect identities. In the case of network packets, we would anonymize the IP headers and blank out the packet bodies, and declare the data sanitized. In a medical record, we would sanitize the patient's name and address. Unfortunately, this is insufficient, because other data—for example, the patient's height, weight, and date of birth—when taken together might comprise a composite key that identifies the patient uniquely.

We cannot merely apply either form of anonymization to these three additional fields because many analyses of medical data are sensitive to the values of these fields. If sanitization randomly replaces one value of these fields with another, we risk significantly altering their semantics even if we comply with integrity constraints (such as the sanitized height and weight being reasonable positive numbers, and the date of birth being in the not-too-distant past). For example, the ratio of a person's height to weight is medically significant. Moreover, preserving the approximate order of ages among all patients is important for many medical analyses. Thus, for some data fields, there exist potential conflicts between the requirements of sanitization and the requirements of analysis that may not be apparent to the developer of the privacy policy.

Thus, certain fields of the data connote additional semantics beyond serving a merely denotational purpose. Were this not so, pseudo-anonymous sanitization would satisfy the requirements of the analyst. But because the data in some fields are descriptive attributes, characteristic of measurements, rather than arbitrarily-assigned identifiers, pseudo-anonymous sanitization may conflict with analysis. Further, that a datum has semantic significance may not be immediately apparent. As another example, for some analyses even patient names are not absolutely "meaningless" identifiers. Patients named "Alice" are more prone to pregnancy than patients named "Bob;" patients with Japanese surnames are more likely to be lactose-intolerant than those with English surnames; and patients with African surnames are more prone to sickle-cell anemia than those with Russian surnames. Accurate analysis may require that the sanitization function preserve these semantic connotations via equivalence classes of names. Thus, we need an explicit analysis *policy* and analysis *metric*.

Similar semantic issues complicate the sanitization of IP traffic, because IP addresses may be associated with attributes via the behavioral semantics of their transactions. As an example, if packets to and from a particular host all have TCP address (port number) 53, then the fact that that host is a DNS server will be obvious even if fully-anonymous sanitization is used. Similar semantic signatures in the transaction traffic allow an adversary to map the network infrastructure, although the exact IP addresses of the infrastructure hosts (and indeed the number of such hosts) may not be identified.

Namespace issues require human decisions and analysis. No automated tool can determine whether additional semantics not known to it would unduly assist the adversary, or hinder the analyst. We emphasize that an explicit *threat model*, *privacy policy*, and *analysis policy* must guide decisions regarding what data needs to be sanitized, and how.

4. Tool

In designing our prototype network packet sanitizer, *tcpsani*, our goal was to create a research vehicle that would yield immediate practical benefits, while also serving as a platform to research basic, unresolved issues that can arise at any level, and rapidly prototype new methods for exploring the fundamental problems of sanitization. *Tcpsani* inputs a *tcpdump* "savefile" file, sanitizes it as described below, and then creates a new "savefile" containing the sanitized data. This file can be fed to *tcpdump*, or any compatible program, for display or other operations.

Tcpsani is a modified version of *tcpdump* that invokes Perl routines to do network layer sanitization. Two default modes are supplied; the user may write others if different sanitization algorithms are desired.

In pseudo-anonymous mode, *tcpsani* maps an IP address byte-by-byte. The map for an IP address byte is determined by its IP address prefix (a prefix may consist of 0, 1, 2, or 3 bytes). This approach preserves common (byte-aligned) prefixes under sanitization. The collector can configure certain maps as *shared* by multiple IP address prefixes. Thus, two input IP addresses with different prefixes may have some of their subsequent bytes sanitized via the same map.

Shared maps that permute the low order byte or bytes of IP addresses may be used to preserve certain regularities common to related subnets, for example keeping the IP addresses of local nameservers or switches in the same relative order. A reduction in privacy is the tradeoff cost for tractable analysis of the sanitized data.

Fully-anonymous mode maps a region *R* of the original IP address space to a target IP address region *T*. When *tcpsani* encounters an input IP address from region *R* that requires a new address in region *T*, it randomly picks an empty slot in region *T*, marks it as allocated, and enters that target address in a hash table keyed by that input address.

Tcpsani can be configured to implement all the *hybrid* (combinations of pseudo- and fully-anonymous) allocation schemes described earlier in section 3.2.

5. Related Work

One property whose preservation benefits analyzers is that of IP address prefixes. By default, *tcpsani* implements only byte-aligned prefix preservation. *Tcpsani*'s sanitization modules could be augmented with code to implement prefix preservation on the fly. This would provide the functionality of *tcpdpriv* [4].

If prefix preservation is the sole objective, a better alternative is CryptoPan [5,6,7], an elegant and efficient specialized hash. CryptoPan also has the virtue that multiple collectors can implement the same prefix-preserving permutation for aggregation by a single analyzer merely by sharing a small secret key, rather than sharing large explicit maps.

We speculate that CryptoPan and similar work (for example, Peuhkuri [8]) have not been more widely accepted as "the solution" to the tension between IP address sanitization and the desire for aggregated analysis because they require a high degree of trust among different collectors. For example, given a prefix-preserving hash key, any trusted collector (or its rogue insiders) can invert any target IP address by a sequence of 32 chosen-plaintext attacks. These attacks are performed offline—there is no need to inject them into a monitored traffic stream. Hence, if any aggregated CryptoPan-sanitized dataset is ever made available to an adversary, that data set will remain vulnerable to these insider attacks for all eternity.

The sharing of identical, explicit maps (even if they do not preserve prefixes) by multiple collectors likewise carries this perpetual vulnerability if the aggregate sanitized dataset is published. Thus, regardless of how the mutually-identical sanitization is implemented, a collector is perpetually vulnerable to any other entity with access both to the permutation key/map and to the permuted data. In light of recent security breaches at many commercial analyzers of credit information, it is worth noting that a collector who trusts an aggregating analyzer with its data today, must also trust that analyzer in the future not to fall prey to an adversary masquerading as a new collector who wants to sanitize its data using the common, historical sanitization function.

Pang and Paxson [9] studied how to make public network packet trace data without compromising the privacy requirements of their site. They separated the problem of network protocol level sanitization from that of application protocol level sanitization, and implemented policy scripts to operate on the latter. By way of contrast, we focus only on the network level data, and use modules to describe the privacy policy of that data. We do not yet deal with the application layer protocols (but see the next section).

Sobirey, Fischer-Hübner, and Rannenberg [10] first suggested pseudo-anonymous sanitization, in the context of intrusion detection. They discuss the need to balance pseudonymity with the preservation of enough information to perform an adequate analysis, but do not describe how to achieve that balance. This paper also identifies the problem of conditional reconstruction, in which one may map pseudonyms to users given additional (external) knowledge.

Biskup and Flegel [11] considered the pseudo-anonymous known user and host names that appeared in file names, and in the user and host fields of the logs. They discussed in detail several possible architectures for pseudonymizing log files.

Lundin and Jonsson [12] describe an experiment in which they developed a "pseudonymizer" that exchanges pseudonyms for names in firewall logs. The mapping between names and pseudonyms was not amenable to reconstruction. The authors concluded that even pseudonymized users sometimes could be reidentified through their behavior, and some information (such as working hours) could be deduced from the sanitized logs. Further, knowledge of the users' behavior helped distinguish false alarms from legitimate reports of intrusions. Sanitizing the logs reduced this knowledge, increasing the need to investigate alarms that otherwise would have been quickly dismissed as patently false.

6. Conclusion

The data sanitization problem is similar to several other interesting problems.

View a privacy policy and an analysis policy as constraints on *inferences*. An acceptable sanitization must produce a dataset from which "good" analysis inferences can be drawn, but "bad" privacy-penetrating inferences cannot be drawn beyond some threshold degree of accuracy or probability.

Another related problem is the database query-audit problem: given a central database and multiple query-makers, this problem asks how the database management system should respond to each query so that the aggregated results reveal no more than the sum of the individual query results. The problem of sanitizing data aggregated from multiple collectors is, in some sense, a dual of the database query-audit problem: how can we *confidentially* perform a globally uniform sanitization on multiple distributed writes to a central database, so that analyzing the aggregate results *will* reveal more than the sum of analyzing each separate database write without revealing the raw data?

Many notions inherent in sanitizing network traffic arise in the development of anonymous and pseudonymous network and cryptographic protocols. For example, the notion of crowds [13] raises the issue of the degree of anonymity, which also appears from the semantic issues discussed above. The notion of an anonymity set [14] is inherent in sanitization done in a finite name space.

The issue of semantics is crucial to proper data sanitization because shared semantics may cross multiple syntactic formats. Many network protocols, such as ARP and NTP, have some structure from which semantics can be inferred. In this case, automating the data sanitization based on that structure should work fairly well. But many other network protocols, such as TELNET and HTTP, either are unstructured or mix structured data (such as commands) with unstructured data (such as contents of files). In this case, the unstructured data will not be well sanitized. The current version of *tcpsani* sanitizes data based on structure, blanks out the unstructured data portions, and changes ancillary information such as checksums to correspond to the new values.

The latter may not always be possible. Consider the situation in which the data to be sanitized has been digitally signed by a third party. Once the data is sanitized, the digital signature cannot be preserved because the signature corresponds to the *raw* data, not the transformed (santized) data. This is a property of the digital signature, since the data to which it is bound has changed. But now an analyst could not determine whether the third party signed the raw data, given the sanitized data and the digital signature. This is an example where the privacy and analysis policies are in conflict. The only solutions are to have the third party re-sign the sanitized data, or to have the sanitizing entity sign the data. This breaks the association of the third party with the data, unless the third party delegates authority to the sanitizing entity to sign.

Data sanitization is rapidly becoming a necessity. It raises many problems and issues, and is very sensitive to the environment in which it is done as well as to the purpose to which it is put. This paper presented a tool to sanitize network traffic, and discussed some of the issues in the context of that work.

Acknowledgements. Thanks to the Department of Computer Science System Support Group, and Robert Ono of Information and Educational Technology for helping us with our experiments. This work was supported by the Trusted Computing Program award number CCR-0311671 from the National Science Foundation to the University of California.

7. References

[1] M. Bishop, B. Bhumiratana, R. Crawford, and K. Levitt, "How to Sanitize Data," *Proceedings of the 13th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises* (WETICE 2004) pp. 217–222 (June 2004).
[2] V. Jacobson, C. Leres, and S. McCanne, "Tcpdump", 3.9.3 (July 2005); http://www.tcpdump.org.
[3] T. Daniels and E. Spafford, "Identification of Host Audit Data to Detect Attacks on Low-Level IP Vulnerabilities," *Journal of Computer Security* **7** (1) pp. 3–35 (1999).
[4] G. Minshall, "Tcpdpriv", release 1.1.10 (Aug. 1997); http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.
[5] J. Fan, J. Xu, M. Ammar, and S. Moon, "Prefix-Preserving IP Address Anonymization", *Computer Networks* **46** (2), pp. 253-272 (Oct. 2004).
[6] J. Xu, J. Fan, M. Ammar, and S. Moon, "On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization", *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement* pp. 263–266 (2001).
[7] J. Xu, J. Fan, M. Ammar, and S. Moon, "Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme", *Proceedings of the 10th IEEE International Conference on Network Protocols* pp. 280–289 (Nov. 2002).
[8] M. Peuhkuri, "A Method to Compress and Anonymize Packet Traces", *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement* pp. 257–260 (2001).
[9] R. Pang and V. Paxson, "A High-level Programming Environment for Packet Trace Anonymization and Transformation", *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2003)* pp. 339–351 (Aug. 2003).
[10] M. Sobirey, S. Fischer-Hübner, and K. Rannenberg, "Pseudonymous Audit for Privacy Enhanced Intrusion Detection," *Information Security in Research and Business—Proceedings of the IFIP TC11 13th International Conference on Information Security* pp. 151–163 (May 1997).
[11] J. Biskup and U. Flegel, "Transaction-based Pseudonyms in Audit Data for Privacy Respecting Intrusion Detection," *Proceedings of the Third International Symposium on Recent Advances in Intrusion Detection* pp. 28–48 (Oct. 2000).
[12] E. Lundin and E. Jonsson, "Anomaly-Based Intrusion Detection: Privacy Concerns and Other Problems," *Computer Networks* **34** (4) pp. 623–640 (Oct. 2000).
[13] M. Reiter and A. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Transactions on Information and System Security* **1** (1) pp. 66–92 (June 1998).
[14] D. Chaum, "The Dining Cryptographers' Problem: Unconditional Sender and Recipient Untraceability," *Journal of Cryptography* **1** (1) pp. 65–75 (1988).