How to Sanitize Data

Matt Bishop, Bhume Bhumiratana, Rick Crawford, Karl Levitt

Department of Computer Science
University of California, Davis
One Shields Ave.
Davis, CA 95616-8562
{bishop, bhumirbh, crawford, levitt}@cs.ucdavis.edu

Abstract

Balancing the needs of a data analyst with the privacy needs of a data provider is a key issue when data is sanitized. This work treats both the requirements of the analyst and the privacy expectations as policies, and composes the two policies to detect conflicts. The result can be applied to an intermediate data representation to sanitize the relevant parts of the data. We conclude that this method has promise, but more work is needed to determine its effectiveness and limits.

1. Introduction

The Internet has blossomed into a medium used for communications by all segments of society that have access to computers. It has become a major influence in the commercial, academic, and government sectors. People use it for personal as well as for commercial purposes. This rapid growth has meant that the Internet is outstripping the tools available to it, and forcing developers and architects to devise new technologies to allow people to use the Internet safely.

The catch is the word "safely". What is safe to one person or entity may be unsafe to another. For example, most businesses accepting transactions over the Internet store the numbers of customer credit cards on-line at least until the transaction is completed, and sometimes even longer as a convenience to the customer (who need not re-enter it). To the store, the credit cards being on-line is a necessity and, because of the protections on their system, one they believe is safe. But a customer may feel that the credit card being on-line is unsafe, because of the threat that an unauthorized user may gain illicit access to the card number.

A similar situation arises in co-operative relationships among businesses, government, and in fact any two or more entities that share information. A company may need to share information with one of its associated firms, but not reveal *all* information in order to protect the privacy of its clients. For example, a business may wish to reveal to its trade association the amount of widgets sold in each foreign country, but not whom the purchasers were. In this case, the company must redact information from the raw data before providing it. The redaction may take one of two forms: *summary* or *sanitization*.

A summary is a completed analysis of the data in which the relevant information is used to compute statistics such as counts, means, and so forth. Typically, a gross description of the data is given to provide context. The U. S. Bureau of Statistics presents summaries of raw data to the public, and does so in such a way that the raw data cannot be rederived from the summary. Hence the summary conceals all aspects of the raw data that the Bureau wishes to suppress. A summary of the corporate data mentioned above would name the continents and state the number of widgets purchased by customers from that continent.

Sanitization takes the opposite approach. The raw data is presented for others to analyze, but the data is transformed so that sensitive items are suppressed. Medical records given to researchers are treated in this fashion. Diagnostic information, symptoms, and treatments are given, but information identifying the specific patient, such as name, address, phone number, and so forth, are redacted. A sanitized set of the corporate data mentioned above might consist of a list of purchasers, their addresses, and the number of widgets each purchased, but the names and addresses (except for the continents) would be replaced by meaningless strings.

The benefits of sanitization are that the recipient of the data can analyze the raw data and derive statistics, or take actions, based upon the data itself rather than the provider's summary of the data. But it also introduces a drawback when the recipient needs to derive information based upon the components of the data that are sanitized. In our medical example, the recipient may want to check the prevalence of a particular disease with respect to geography. As the provider has suppressed

We gratefully acknowledge funding through grant CCR-0311671 from the National Science Foundation to the University of California, Davis.

addresses, the recipient cannot determine the geographical location of the people with the disease from the medical records.

This tension is particularly acute in the realm of computer security. When analyzing systems for intrusions, computer security analysts typically look for specific traces, or signatures, indicating attacks have occurred. This requires access to data such as system logs, network traffic, and the like. But embedded in this data is confidential information, passwords being the obvious example. This data is not to be revealed to the analysts. It is important to realize that the issue here is not only one of trust; some information is required by law or contract not to be disclosed.

The goal of the project described in this paper is to study the tension between the needs of analysts for data, and the needs of others for privacy of components of that data. The specific goal is to alert both analysts and providers of data when a conflict between the analytic needs and the privacy requirements exists, and to ensure that once the conflict (if any) is resolved, the sanitization can meet the needs of both parties. Section 2 discusses the approach for resolution in detail. Section 3 discusses the mechanics of sanitization, for both data that is complete when sanitization begins and for data that is being accumulated throughout the sanitization process. Section 4 presents a proposed architecture for a sanitization system, and section 5 contrasts this approach with previous work. Section 6 summarizes our progress.

2. Privacy and analysis

The tension between privacy and analysis underlies all of sanitization. If privacy is to be respected, parts of the raw data must be deleted. If analysis is to be successful, the parts of the raw data used must be left untouched. What happens when privacy constraints dictate that some parts of the raw data used for analysis must be altered?

We formalize this problem as follows. Given a policy X and a data set Y, we write $X(Y)$ when the data set Y satisfies the policy X. A *privacy policy* is a set of properties p that the sanitized data must satisfy. A *sanitization function* s: $D \times P \rightarrow D$ takes a data set $d \in D$ and a privacy policy $p \in P$ and produces a second data set $d' \in D$ such that d' satisfies p, written $p(d')$. An *analysis policy* is a set of properties a that data must satisfy in order to do the desired analysis correctly. A privacy policy p and an analysis policy a are *consistent* under the sanitization function s if and only if, for all sets of data d, d satisfies a if and only if $s(d, p)$ satisfies a:

$$a, p \text{ consistent under } s \Leftrightarrow (\forall d \in D)[a(d) \Leftrightarrow a(s(d, p))]$$

This relation also defines equivalence classes of sanitization functions. Let s_1 and s_2 be sanitization functions. Then they are equivalent if they can be used interchangeably. More formally, $s_1 \equiv_{a,p} s_2$ when for all data sets D, $a(s_1(d, p))$ holds if and only if $a(s_2(d, p))$ holds.

As an example, consider a site with an Internet connection. The managers wish to analyze the network traffic for the *land attack*. This attack occurs when a network packet has the same source and destination ports and addresses [1], and can disable systems. The analysis policy may be expressed as:

a packet constitutes a land attack if and only if
the source and destination IP addresses are equal
the source and destination ports are equal

The company values the privacy of its employees, and each employee has his or her own computer system. Allowing the security analysts to see the IP address would immediately allow them to tie the network traffic to an individual, and the company is concerned that, for example, security analysts could build profiles of the individuals from their electronic mail. Hence the IP addresses must be kept private. So, the privacy policy is:

no IP addresses shall be revealed

This policy is actually ambiguous, because it does not state *how* the addresses are to be suppressed. First, the IP addresses can simply be deleted. In this case, the data set $s_1(d, p)$ has no IP addresses, and does not satisfy the policy a as the veracity of the first property cannot be determined. So a and p are not consistent under s_1. Simply replacing the IP addresses by a single string, say "IPADDRESS", may false positives, and again a and p are not consistent under this sanitization function. The IP addresses can be replaced by random addresses. In this case, the data set $s_2(d, p)$ has IP addresses, but the same IP address may be sanitized into two different addresses. So two IP addresses in $s_2(d, p)$ may produce false negatives. Again, a and p are not consistent under s_2. Finally, the IP addresses can be replaced by random addresses, but consistently so that each IP address is replaced by the same random string whenever it occurs. Given two IP addresses in $s_3(d, p)$, the first property of a holds in d if, and only if, it holds in $s_3(d, p)$. Thus, a and p are consistent under s_3. Note that the *particular* random addresses used are irrelevant; in fact, all sanitization functions providing such a mapping are in the same equivalence class.

A critical issue is how to express the above policies. One goal is to derive from the policies specific criteria that the sanitization function must meet in order for the two policies to be consistent with respect to that function. Thus, the language chosen to express the policies must lead directly to the derivation of an appropriate sanitization function. If the policy language is too high level, this will not be automatic. But if the policy language is too low-level, it may lack the expressive power needed, or be too detailed to use. Complicating the selection is the need to reconcile the two policies and determine inconsistencies. One approach is to compose them, and then check the resulting policy for any inconsistencies. However this is done, the policy language chosen must have an analysis engine available that allows this reconciliation.

2.1. Example policy language

An obvious approach to policy representation is to choose a policy language that allows the sanitization functions to be derived easily. Coupled with the wide variety of data formats, it would be very difficult to construct a policy language that would cover all situations for all types of data. Instead, we assume that the data itself is structured using XML, and express policies in terms of that structure. In essence, our policy language is a low-level one, as it is aware of the particular semantics of the data being protected. But it is general, because it is not tied to any particular semantics.

An example will clarify this. Consider network traffic. The privacy policy and analysis policy are as above. Let the IP addresses and port numbers be mapped into the fields with tags "ip" and "port", and attributes "src" and "dest" for source and destination. The low-level analysis policy is:

LAND_ATTACK iff
ip:src = ip:dest and port:src = port:dest

and the privacy policy requires that the IP addresses be sanitized. This may be translated into three different low-level privacy policies, p_1, p_2, and p_3:

p_1 ip:src/contents delete; ip:dest/contents delete

p_2 ip:src/contents random; ip:dest/contents random

p_3 ip:src/contents chash1; ip:dest/contents chash1

respectively, and the sanitization functions s_1, s_2, and s_3 shown above can be generated automatically. (In this little language, ":" separates tags and attributes, and "/contents" refers to the contents of the named tag with the given attribute.)

Consider how a policy reconciliation engine might work, using policy p_1 as an example. As the source and destination IP addresses are "deleted", the engine notes that they are no longer available. It then compares this to the analysis policy, which requires both, and reports that the two are inconsistent.

Now use policy p_3 as an example. Here, the source and destination addresses are mapped using "chash1", which is a cryptographic hash function. If f is a function, $x = y$ means that $f(x) = f(y)$, so when the engine compares this to the analysis policy, it notes that the two IP addresses are equal whenever the result of applying s_3 to both are equal. Hence privacy policy p_3 is consistent with the analysis policy.

2.2. Discussion of XML

Three ways to derive objects to be sanitized are:

1. *Fixed objects*. Whenever a particular object is encountered in the input, it is sanitized.
2. *Objects in position*. The position of an object is determined, and if at a specific position, the object is sanitized. This is useful for data streams in which objects have fixed positions.
3. *Objects in context*. The context of an object is evaluated, and the result of the evaluation determines how (whether) the object is sanitized.

Pang and Paxson [2] discuss four types of inference attacks, the prevention of which involves separation of data. An alternate view of these countermeasures is to require sanitization to depend on context. For example, a version of "known text matching" consists of an attacker inserting IP addresses into a network session. The network traffic is anonymized, and the attacker obtains it. She then compares her known session text with the sanitized one. This will immediately show her how IP addresses were sanitized. The solution is to recognize that network addresses in the data area are to be sanitized differently than those in the header. In "shared text matching", the user name "alice" should be sanitized in contexts where the string refers to a login name, but not in a context when it is part of a file name. "Structure recognition" uses context derived from multiple probes within a trace, and "fingerprinting" is similar to known-text matching.

XML offers several advantages for this process. First, XML structured data has a tree form in which the tags are the nodes and content are the leaves, so building contexts involves traversing the tree, keeping track of the ancestor nodes of the current content. So sanitization of attributes and content can easily take context into account. Secondly, in the event that the structure of the data itself is to be changed, the transformation language XSLT, tailored for XML, can express the change easily. Third, many tools for processing XML exist, and there are established standards for describing many types of data and transformations, so the use of XML can draw on a body of available work.

Unstructured input must be transformed into structured data. We discuss this in section 4 below.

3. Mechanics

A key question in sanitizing data is how effective the sanitization is. Many of the results from anonymity and pseudonymity in the cryptographic world apply here, the difference being that in this context anonymity and pseudonymity apply to hiding data rather than identity. The key question is how difficult an attacker would find inverting the sanitization mapping. In many cases, one need not invert the mapping completely; merely eliminating possible data may provide information that the sanitizer wishes to suppress.

Second, in some cases the data to be sanitized is not known initially. For example, a policy may require login names to be changed. The sanitizing engine (which applies the sanitizing functions) must determine this data by looking for words that follow the string "login:". The data not being known initially, the function is in a sense constructed after, or as, the data is being analyzed.

When the complete set of data to be sanitized is available at the time of sanitization, the sanitization functions can be derived completely before the data is sanitized. This is *static sanitization*. When the complete set of data to be sanitized is not available at the time of sanitization, the sanitization function may change as the data becomes available and is sanitized. This is *dynamic sanitization*. These are handled differently. In what follows, we assume the sanitization functions are defined, at least in part, by the contents of the complete set of data being sanitized, as in our example above.

3.1. Static sanitization

Static sanitization arises when a file or data set will not change during the sanitization process. In this case, the data can be analyzed to determine the sanitization function completely before the analysis proceeds. The sanitization engine makes passes over the complete data set until the sanitization function is constructed, and then makes a final pass over the data to perform the sanitization. This is analogous to a two-pass assembler, in which symbol names and definitions are determined on the first pass, and then the object code is assembled on the second pass.

As an example, a log file is to have user names removed. The third field of the log file contains the user names; other fields may also have them, but the format of the log entries varies after the first three fields. Hence the sanitizer must scan the log file first to build up a collection of user names from the contents of the third field of each entry. Then it maps each into a unique replacement string, and processes the log file line by line. The names are replaced by the replacement strings regardless of which field they appear in.

It is possible to build a one-pass sanitizer. Given a structured data format in which the words to be sanitized occur in well-defined places, exactly the same techniques that create one-pass assemblers could be used to construct one-pass sanitizers.

3.2. Dynamic sanitization

Dynamic sanitization occurs when the data source continues to add to the data as sanitization proceeds. In this case, the sanitization function may not be known until the processing is completed. If the sanitization is ongoing, the sanitization function may not be a fixed mapping, but may change throughout the sanitization process.

This problem is considerably more complex than static sanitization. First, when the sanitization function changes, must the already processed input be redone to conform to the new sanitization function? This policy decision must take into account the nature of the data already processed, to determine if the change would apply to any previously processed data; it must account for the availability of the already processed data, which may have been transferred out of the control of the entity doing the sanitizer; and it must account for the cost of re-doing all the processing.Second, the complexity of the changes to the function may affect the cost of sanitizing incoming data; the concern is that if the sanitization becomes prohibitively expensive, it will be stopped. This area requires more study.

3.3. How to sanitize data

Basic ways to sanitize objects are:

1. *Deletion*. Here, the objects to be sanitized are simply deleted.
2. *Fixed transformation*. All occurrences of the object are replaced by a fixed string.
3. *Variable transformation*. Occurrences of the object are transformed in different ways depending upon the context and structure of the object. For example, translating an IP address into one value for FTP connections, and a different value for HTTP messages, is an example of variable replacement. Replacing objects with random data is another.
4. *Typed transformation*. This is a form of variable transformation, except that the replacing objects are related when the types of the object being replaced are the same. For example, replacing all file names with a value generated by one cryptographic hash function, and all IP addresses with addresses

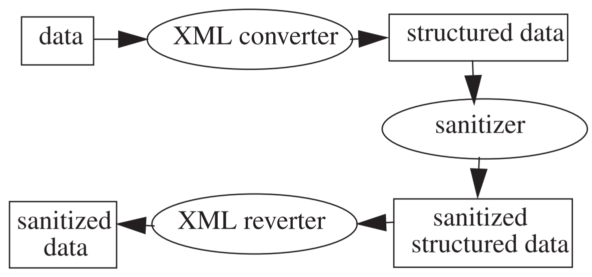

Figure 1. The architecture of sanitization. Raw data is translated into an appropriate structured format. The sanitizer sanitizes the structured data, which is then reprocessed into the original format, giving a sanitized representation of the input data.

selected from the 10. network, would be an example of this.

4. Architecture of a Sanitization Engine

Pulling this together, Figure 1 describes a simple sanitization engine. The advantage of using a structured representation of the input data is that the sanitizer can easily derive context from the XML form, and use that to determine how to sanitize the data.

4.1. XML converter and reverter

The XML converter takes unstructured data as input and applies a structure to it. This application must know the semantics of the input data to convert it to a useful form. For example, if the data is network traffic, the converter must structure the data so the desired information can be gleaned from it. FTP control session data has a well-defined format [RFC], so the contents of the packets can be structured (GET is followed by a file name, for example, and USER by a user name). The precise structuring must take into account the information to be used in the analysis of the data (as stated by the analysis policy), as well as the data to be suppressed (as stated by the privacy policy).

As a detailed example, consider sanitizing FTP network traffic in an environment in which the analysis policy requires access to the packet bodies. The privacy policy requires that IP addresses be suppressed. But some FTP control commands require that IP addresses be sent in the bodies of the packets. The naive solution, sanitizing at the IP level, will fail to meet the privacy policy because the addresses in the packet bodies will not be sanitized. But if the session is reconstructed, the commands that require IP addresses to be in the session will be obvious. Hence the converter must supply enough information to allow the sanitizer to determine, from context, which occurrences of the IP addresses to sanitize. It is worth noting that the representations of the IP addresses in these contexts will differ. IP addresses in packet headers are binary; IP addresses in the session layer will be in ASCII. These representation details should be hidden from the sanitizer, and handled at the conversion into structured format and restoration into unstructured format.

4.2. Sanitizer

The sanitizer takes the structured data, converts it into a tree representation, and applies the sanitization rules of the privacy policy. One model for the sanitizer is to derive structure from the contents of the data, but this would require that the sanitizer know about the format of the data. For simplicity and to avoid writing a new sanitization engine for each type of input data, we use a model in which the sanitizer assumes that *all* context is encapsulated in the XML structure. Thus, the converter must be aware of the contexts in the privacy policy.

As an example, consider FTP traffic. The sanitizer must sanitize IP addresses at both the session and IP layers. The sanitizer will keep two types of states. The first, the IP state, is applied to the fields in the IP header (as represented in XML). The second, the FTP state, is applied to the session data. Note that the FTP state may span multiple IP packets, so the FTP state will need to persist even when an IP packet completes. This information can be encapsulated in the privacy policy using appropriate constructs.

4.3. XML reverter

The XML reverter deconstructs the structure of the sanitized structured data, restoring it to its original form while keeping the sanitizations. This preserves the original format of the input data, so tools that work on the original input data will work on the sanitized data. It is the reverter's responsibility to attend to components that must be changed as a result of the sanitization. For example, returning to our FTP network traffic, the TCP checksums for the packets that have sanitized data in them are now incorrect, and must be patched.

5. Related work

Pang and Paxson [2] explore the sanitization of network traffic. Their goal was to make public full trace data (as opposed to only packet headers) without compromising the privacy requirements of their institution. They implemented policy scripts to operate on the application-protocol-level data, and split the sanitization into network-protocol-level data sanitization and application-protocol-level data sanitization.Their tools allowed verification that the anonymization mechanisms worked correctly and adequately protected the data to be kept private.

Other than Pang and Paxson, most work has focused on the issue of anonymizing or suppressing data. Many cryptographic studies have explored how to do this, but none examine the trade-off between analysis and privacy, focusing instead on preventing or limiting the reconstruction of anonymized data. Inference attacks in databases similarly examine controls to suppress data, but not the trade-off between access and privacy. As the trade-off is the central theme of our study, our work differs considerably from theirs.

Sobirey, Fischer-Hübner, and Rannenberg [3] first suggested pseudonymity (consistent replacement of words to be sanitized) for intrusion detection. They focus on protecting user privacy. 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 pseudonymizers to users given additional (external) knowledge.

Biskup and Flegel [4] discussed the pseudonymized 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, but did not explore how this affects the ability to analyze data especially under criteria that change as the analysis progresses.

Lundin and Jonsson [5] 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. This work was experimental, and did not examine the balance between security and privacy in detail.

Our work is closest to that of Pang and Paxson but uses a slightly different approach. We abstract the network notions into structures, and focus on the structures. This gives us generality and extends the framework to make applying a contextual model of analysis simpler.

6. Conclusion

A primary goal of this project is to make explicit the conflicts between analysis and privacy, and to enable policy makers to determine what the trade-off should be when the two conflict. Because we want our results to be widely applicable, we are focusing on structured data using XML, and methods to sanitize data in that context.

Our project has developed a sanitizer using this technology for network data. It can sanitize header information including TCP, UDP, IP, and MAC addresses, as well as the bodies of the packets at any of these network layers. Our department has used a prototype of this tool to sanitize wireless network traffic, as well as logs containing DHCP lease information, to meet University privacy requirements so that a research group could use the data to analyze patterns of use of wireless. Preliminary performance data implies the tool's performance is acceptable. These results are encouraging.

In our society, the need for security is great, and with that need comes the requirement that data be analyzed for threats. But the importance of privacy, the "right to be let alone" [6], is fundamental to human societies, and must be respected. How the conflict between security and privacy is to be balanced is a matter of policy, and open to debate; but that those debating must understand what the conflicts are seems self-evident. We hope this work, in its own small way and within its specific problem domain, contributes to this understanding.

7. References

1. 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).
2. R. Pang and V. Paxson, "A High-Level Programming Environment for Packet Trace Anonymization and Transformation", *Proceedings of SIGCOMM 2003* pp. 339–351 (Aug. 2003)
3. 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).
4. 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).
5. E. Lundin and E. Jonsson, "Anomaly-Based Intrusion Detection: Privacy Concerns and Other Problems," *Computer Networks* **34**(4) pp. 623–640 (Oct. 2000).
6. S. Warren and L. Brandeis, "The Right to Privacy", *Harvard Law Review* **4** pp. 193ff (1890).