

Towards Packet Anonymization by Automatically Inferring Sensitive Application Fields

Po-Ching Lin*, Yi-Wen Lin*

*Department of Computer Science and Information Engineering

National Chung Cheng University, Chiayi, Taiwan

Email: {pclin, lyw98m}@cs.ccu.edu.tw

Abstract—Real network traffic is an important asset for research and development in the field of network security, but due to privacy concerns of leaking personal and host information, acquiring real traffic is quite restricted in practice. Over the past years, researchers have developed anonymization techniques to specify sensitive application fields or patterns to be hidden, but the specification will take great effort to investigate the sensitivity of a large number of application fields beforehand, and can be inaccurate due to the lack of patterns for some sensitive information. This work presents an innovative method towards *automatically* inferring where sensitive information is in the application messages. The method can leverage existing application protocol parsers to locate the application fields or optionally infer the fields by *clustering* and *aligning* similar application messages. After that, this method can infer the degree of sensitivity of application fields with the C4.5 decision tree algorithm based on the three measures: *entropy*, *diversity*, and *one-to-one mapping*. The experimental results demonstrate that the inference of sensitivity is effective with low false-negative and acceptable false-positive rates.

keyword: anonymization, packet trace, application protocol.

I. INTRODUCTION

Packet traces from real network traffic are valuable assets for network research and development. Although it is possible to acquire network traffic using tools such as the Harpoon traffic generator [1], the emulated traffic is usually criticized for being unrealistic because real traces are rather versatile, containing traffic from peer-to-peer applications, online games, various attack vectors, besides that from common applications such as Web and email. For characterizing network traffic, studying malicious network behavior, comparing competing ideas, validating the robustness of a new implementation, and so on, acquiring real network traffic is strongly desired.

Acquiring or sharing packet traces from real network environments is usually restricted in practice due to the privacy concerns. Two types of sensitive information can be in the traces: (1) personal information such as passwords, Web browsing history and email addresses, and (2) system- or application-related information such as the OS version and the server program on a host. The former is undoubtedly sensitive. The latter is also considered sensitive because an attacker can leverage the information through *passive fingerprinting* to find out vulnerable targets to be attacked [2].

Packet anonymization can help to sanitize sensitive information in the packet traces [3]. Existing methods have been effective in anonymizing the fields in the TCP/IP and the link-layer

headers [3]–[5], and some also support anonymizing the fields in the application protocols [6]–[8]. For the latter, existing approaches include specifying (1) the application fields that are considered sensitive (e.g., the URL in an HTTP request) to be anonymized, or (2) patterns of potentially sensitive information (e.g., those of email addresses, URLs, and IP addresses) to be searched for in the application payloads.

Manually specifying the right application fields to be anonymized is quite an effort. First, given the large number of application fields in real packet traces, it is non-trivial to study the semantics of every field and find out all sensitive ones. Second, the specification relies on accurate parsing of application protocols to locate the right fields. Even though a network protocol analyzer such as Wireshark (www.wireshark.org) can support hundreds of protocol parsers, neither the fine granularity of its protocol parsing nor the number of supported application protocols is sufficient. For example, Wireshark can identify an FTP response message, but the parsing does not go down to the fields in the message, which can contain sensitive information such as file names. Moreover, Wireshark does not support parsing some proprietary application protocols such as those in many on-line games, but the messages in them may be sensitive as well.

Specifying sensitive patterns is also unreliable. For example, there are no specific patterns to represent personal accounts and passwords, which can be specified arbitrarily by users. Moreover, sensitive information may be represented in multiple ways. For example, the IP addresses in the IP header carried in the payload of an ICMP reply are simply represented in four bytes (see RFC 792), not in the common dotted representation of ASCII characters. The IP address and port number in the FTP PORT command are represented in the form like 192,168,150,80,14,178. The first four values are an IP address and the last two encode a port number. This representation separates the values in an IP address with commas, so it will be missed if an anonymization tool seeks a dotted pattern of IP addresses.

This work proposes two methods towards solving the aforementioned problems: (1) inferring the fields in the application messages, particularly those without a fine-granular parser, and (2) inferring whether an application field should be sensitive or not. For the former, this work provides a *clustering* and *alignment* method on application messages to identify the fields. For the latter, this work studies the measures and the

classification method that help to identify potentially sensitive application fields.

Contributions: To the best of our knowledge, there is no prior work regarding automatically inferring sensitive information in the application messages for packet anonymization. The contributions of this work are as follows:

- 1) We present a method to automatically identify the fields in the application messages if there is not a proper protocol parser.
- 2) We present the measures and the classification method to identify potentially sensitive information in the application messages.
- 3) We evaluate the accuracy of the methods in this work with real packet traces, and demonstrate the feasibility of inferring sensitivity of application fields.

The rest of this work is organized as follows. Section 2 reviews related work and discusses the limitations. Section 3 presents the methods to automatically infer application fields and judge potentially sensitive application fields. Section 4 studies the accuracy of the algorithms with several experiments, and investigates the cases of false negatives and false positives. Section 5 concludes this work and points out the future work.

II. RELATED WORK

In this section, we review the open resources of packet traces, and the anonymization tools and techniques. We will discuss their limitations to justify the motivation of this work. Finally, we will introduce the related work on inferring fields of application protocols, which is closely related to this work.

A. Sharing the network traces

Despite the benefit of sharing packet traces, open resources available on the Internet are quite limited due to privacy concerns. OpenPacket.org (www.openpacket.org), Packetlife.net (packetlife.net), and Wireshark SampleCaptures (wiki.wireshark.org/SampleCaptures) are examples of the resources. The traces on these sites are uploaded from active participants and usually left unanonymized because the participants can submit only those packet traces they allow. Although the traces are frequently updated, the volumes of individual traces are too small for intensive network analysis and testing network security equipment.

The Web site CAIDA (www.caida.org) provides a large volume of packet traces from an OC192 Internet backbone. The IP addresses in the packets are anonymized by the CryptoPan prefix-preserving algorithm [5], and the payloads are completely removed. However, removing the payloads will significantly reduce the value of the traces for network security analysis, which usually inspects the payloads. A recent work called NCTU BetaSite [9] records network traffic from voluntary students in a university campus. The BetaSite supports anonymizing sensitive information by specifying the application fields and patterns, but the specification is still a manual process.

B. Anonymization tools and techniques

Due to the limitations of open resources, it is common that network researchers collect packet traces by themselves, say from the network of their affiliations. Several anonymization techniques and tools have been developed for the network administrator to hide sensitive information in the packet traces. They usually support three strategies: (1) directly dropping the payloads, (2) specifying the protocol fields to be anonymized, and (3) specifying the patterns that may be sensitive information. According to the preceding discussions, neither strategies can effectively anonymize sensitive information in packet traces.

Anonymizing packet traces inevitably distorts the original semantics of the application messages, thereby degrading the *utility* of the packet traces for analysis. Preserving the utility after packet anonymization is therefore an important issue. For example, a technique called *prefix-preserving* can preserve the relationship between the anonymized IP addresses in the same subnet [4]. That is, if two original IP addresses share a k-bit prefix (i.e., belonging to the same subnet), their anonymized addresses will still share a k-bit prefix — just the prefix is different. The preservation strategy can promote the utility of flow analysis. The work in [8] compares the utility of anonymized packet traces by several tools to intrusion detection systems (IDSs). The work in [10] particularly studies the utility of anonymized HTTP fields to IDSs. The studies are complementary to this work, which focuses on improving the ability to precisely anonymize sensitive information. The users can find the potentially sensitive information in the packet traces based on this work, and then make a white list of the information to be analyzed for maximizing the utility of the packet traces.

C. Inferring the fields of application protocols

Inferring the fields of application protocols has been studied in the past. The main idea, derived from multiple sequence alignment (MSA) (en.wikipedia.org/wiki/Multiple_sequence_alignment) in bioinformatics, is aligning similar application messages and identifying the constant and variant components in them. One of the MSA approaches is called *progressive alignment construction*, which uses the Needleman-Wunsch algorithm [11] to perform an alignment on two closest sequences derived from similarity comparisons on a global set of sequences.

Roleplayer [12] modifies the Needleman-Wunsch algorithm to improve its performance by adjusting the weight in aligning two application messages to find the most appropriate alignment. Discoverer [13] improves the alignment method by first breaking up a message into a sequence of tokens, and then inferring the fields by comparing the counterparts in another message. The alignment is based on the tokens rather than raw byte sequences. The work in [14] tracks the program binary implementing the protocol to further improve the precision of protocol reverse engineering, but tracking all the binaries for the protocols involved in large packet traces will be extremely complicated, if feasible. This work is inspired from

Discoverer for clustering and merging application messages. Rather than rely on the precision of tokenization and semantic inferences, this work simplifies the process by clustering based on similarity between constant parts (usually keywords) in the messages. The simplification is possible since we do not need the full functionality of protocol reverse engineering for packet anonymization.

III. DESIGN OF THE INFERENCE METHOD

Inferring sensitive application fields involves two main steps: (1) extracting the fields in the application messages, and (2) judging whether each field is sensitive or not. We first present an overview of the method, and then describe the details of both steps.

A. Overview of the method

- *Extracting application fields*: If the parser of an application protocol exists, we can directly rely on it for field extraction; otherwise, we group similar messages into a cluster to infer the fields in them. Note that a parser may not extract the deep fields in some application messages. For example, the parser in Wireshark just dissects the FTP response, 150 Opening ASCII mode data connection for W010.txt (3343 bytes), into the response code and the subsequent text, without extracting the fields of transfer mode, file name, and file size. Therefore, we may also need to group such messages for inferring the fields (e.g., the file name) in the text part. If inferring the fields is necessary, we align the messages in the same cluster for identifying the constant and variant parts in them. Since a constant part appears in every message of the same cluster, it is likely to be a keyword of the protocol, and should be insensitive. In contrast, a variant part is likely to be a field in the application messages. The field will be analyzed further to judge whether it should be anonymized or not.
- *Computing the measures of sensitivity in application fields*: The concept of *sensitivity* is subject, so we need to look for *objective* measures to judge whether a field should be anonymized or not. We use *entropy*, *diversity*, and *the mapping from the source IP address* of the field values as the measures. The reasons behind the measures will be explained later.

B. Extracting application fields

This work uses Wireshark for field extraction. Wireshark supports hundreds of protocol parsers (called dissectors) invoked based on the port numbers to extract the fields. For application messages that Wireshark does not parse into, we extract the message lines from the raw packets with tcpflow (www.circlemud.org/~jelson/software/tcpflow) for inferring the fields later. In the latter case, we sort the message lines of identical application protocols to facilitate the clustering based on the heuristics that the first few bytes of a message are usually commands (e.g., the GET command

```
1) Cookie:ystat_tw_ss376223=9_16940400_234398;
2) Cookie:zynga_toolbar_fb_uid=1018132522
3) GET /2009/visuels/Metaboli_120x600_UK.gif HTTP/1.1
4) GET /2010/07/15/ipad-3hk-smv-price-hk/ HTTP/1.1
5) GET /2.3.1/build/yahoo-dom-event/yahoo-dom-event.js
  HTTP/1.1
6) GET /2.5.1/build/assets/skins/sam/container.css
  HTTP/1.1
7) GET /album/insertPic.php?book=6 HTTP/1.1
8) GET /album/insertPoc.php HTTP/1.1
9) GET /album/select.php HTTP/1.1
```

Fig. 1. An example of messages lines of HTTP requests after the sorting.

in HTTP) or reply codes of the application protocols, and the messages of similar formats will be ranked together after the sorting. The heuristics is largely true with few exceptions in our experiment. Without the prior sorting, the clustering will have to seek throughout all the application messages for similar ones, and become inefficient. Figure 1 presents a trivial example of message lines after the sorting.

1) *Clustering*: The process then groups similar messages into clusters with the two steps: *initial clustering* and *merging*. The former attempts to group the messages with the same *multiple common substring* (MCS), which is a sequence of longest common substrings with at least 3 characters (to avoid too short substrings). The algorithm is described as follows. The derivation of the MCS in this algorithm is similar to that in Section III-B2, except that the algorithm operates only on two messages at a time.

Algorithm 1 The psuedo code of initial clustering.

```
MCS( $S_i, S_{i+1}$ ): finding the MCS of two consecutive message lines:  $S_i$ 
and  $S_{i+1}$ , returning TRUE for success.
CHECK_MCS( $S_i, L$ ): checking whether  $L$  is an MCS of  $S_i$  or not,
returning TRUE for success.
 $i \leftarrow 1; j \leftarrow 1$ 
read a line  $S_i$ ;
 $i \leftarrow i + 1$ ;
while read a line  $S_i$  do
  if MCS( $S_i, S_{i-1}$ ) then
     $C_j \leftarrow C_j \cup \{S_i\} \cup \{S_{i-1}\}$ ; //  $C_j = \phi$  initially.
     $L \leftarrow \text{MCS}(S_i, S_{i-1})$ ;
     $i \leftarrow i + 1$ ;
    while read a line  $S_i$  and CHECK_MCS( $S_i, L$ ) do
       $C_j \leftarrow C_j \cup \{S_i\}$ ;
       $i \leftarrow i + 1$ ;
    end while
  else
     $C_j \leftarrow C_j \cup \{S_{i-1}\}$  //  $\{S_{i-1}\}$  is a standalone cluster.
  end if
   $j \leftarrow j + 1$ ; // Start with a new cluster.
   $i \leftarrow i + 1$ ;
end while
```

We use the example in Figure 1 for demonstration.

- *Initial clustering*: Let S_i be the application message in line i . We first find a common substring $L = \text{Cookie:}$ in S_1 and S_2 , i.e., $\text{MCS}(S_1, S_2)$, implying that S_1 and S_2 are in the same cluster C_1 . The step then proceeds to check whether S_3 belongs to the same cluster using the CHECK_MCS function to see whether L is a substring of S_3 . Because L is not a substring of S_3 , S_3 is considered belonging to another cluster, and the step proceeds to

- After initial clustering
 - * $C_1 = \{S_1, S_2\}$ with MCS = <Cookie:>
 - * $C_2 = \{S_3, S_4\}$ with MCS = <GET /20, HTTP/1.1>
 - * $C_3 = \{S_5, S_6\}$ with MCS = <GET /2., .1/build/, HTTP/1.1>
 - * $C_4 = \{S_7, S_8, S_9\}$, with MCS = <GET /album, .php HTTP/1.1>
- After merging
 - * new cluster C'_1 from C_1
 - * new cluster C'_2 from C_2, C_3 , and C_4

Fig. 2. Result of initial clustering and merging.

check whether MCS(S_3, S_4) is a substring of the subsequent messages. Repeating the above steps, the initial clustering will derive the result shown in Figure 2.

- *Merging*: The initial clustering may generate small clusters. In other words, similar application messages may be scattered over multiple clusters. In the preceding example, $S_3 \cdots S_9$ are all HTTP GET requests, and should be in the same cluster. Merging can mitigate the problem. Let m_j be the string concatenated from the MCS in C_j . The similarity between m_i and m_j is defined by

$$\text{Similarity}(m_i, m_j) = \frac{2 * |LCS(m_i, m_j)|}{|m_i| + |m_j|}, \quad (1)$$

where $LCS(m_i, m_j)$ is the longest common substring of m_i and m_j . Due to low similarity, this step does not merge C_1 and C_2 , but C_2, C_3 and C_4 will be eventually merged into a cluster because the MCSs between them are similar. Figure 2 presents the merging result.

2) *Alignment*: The alignment algorithm is inspired by the work in [16]. For a cluster C_j , we first arbitrarily choose an application message in it, and move a sliding window starting from the prefix of the message. The sliding window is a substring to be tested with the other messages in the same cluster, and its initial length is 3 characters.

Let w_j be the substring in the sliding window. If w_j is also a substring of all the other messages in the same cluster, the sliding window will be expanded one more character to find a longer common substring. The sliding window will keep expanded as such, until the substring in it is no longer a common substring in the cluster. The common substring found is therefore the longest in the iteration. The sliding window is then shrunk to the original length of 3 characters, and shifted to the next character, starting a new iteration to find the next longest common substring. The common substrings found in the iterations are constant parts of the messages (e.g., GET and HTTP/1.1 in C'_2). The remaining substrings are the variant parts, and are likely to be application fields.

We are aware of the work named *Protocol Informatics (PI) alignment* in [15], which leverages the multiple sequence alignment (MSA) algorithm in bioinformatics to find specific fields and constant parts in the protocol messages. We do use the PI alignment algorithm because the similarity calculation in the first stage of PI is time-consuming.

C. Computing the measures of sensitivity

Since the degree of sensitivity of an application field is usually *subject*, some *objective* measures of sensitivity are required to infer how sensitive a field may be. In this subsection, we will introduce the measures that help to identify sensitive information in the application messages. We first present how the source IP addresses and application fields are extracted for sensitivity analysis, and then explain the rationale behind the measures to infer sensitivity.

1) Extracting source IP addresses and application fields:

Let f be an application field either derived by the protocol parser or inferred by the procedure introduced in Section III-B. Suppose the field f appears n times in the packet trace, and the value of each instance is $f_1, f_2, \dots, f_n$. We associate each instance with its source IP address, $s_1, s_2, \dots, s_n$. The association can be a hint of whether the value of a field is common for multiple sources or not. We represent the association with the instance f_i as the pair (s_i, f_i) .

The pairs may be redundant if a source host sends an application message for multiple times. For example, the value in the User-agent field of the HTTP requests (denoting the browser program) from the same user is very likely to be identical. We do not want the redundancy affects the sensitivity analysis, so if two pairs (s_i, f_i) and (s_j, f_j) are the same, only one pair is considered. We finally have the set $\{(s_1, f_1), (s_2, f_2), \dots, (s_m, f_m)\}$ after the redundancy elimination, where $m \leq n$. The set of distinct field values in the instances is denoted as $\{v_1, v_2, \dots, v_k\}$, where $k \leq m$.

2) *Inferring sensitivity of application fields*: The analysis uses *entropy*, *diversity*, and *one-to-one mapping with the source IP address* as the measures to infer the sensitivity of application fields. The implications of the measures to the sensitivity are discussed below.

- *Entropy*: The entropy of a field f is defined as $-\sum p_i \log p_i / \log k$, where $p_i = |v_i| / m$ and $|v_i|$ denotes the number of occurrences in which the value of f is v_i . Entropy implies the degree of unpredictability. If the entropy is close to zero, it means a field's value is nearly predictable, and one can easily guess the value. It is therefore ineffective to anonymize a predictable field; otherwise, if a field's value is unpredictable, leaving them unanonymized means revealing the information that is supposed to be unknown, and the analysis therefore suggests to anonymize such a field.
- *Diversity*: We define the diversity of a field as k/m , where k , as defined above, is the number of different values that may appear in a field. If the possible values are few, it will be easy to make a guess from the few values, and it is ineffective to anonymize such a field.
- *One-to-one mapping*: Consider the mapping from the source IP addresses to a field's values in the above association. If the mapping is one-to-one, only one source IP address ever sends a certain field's value, e.g., a user's account, implying that the field value is likely to be personal information. Otherwise, multiple source IP

addresses can generate a value. In other words, the value is shared and known by multiple users. The measure computes the rate of field values mapped from only one source IP address. If the value is high, a field should be sensitive and be anonymized.

The presence of network address translation (NAT) or dynamic host configuration protocol (DHCP) may change the source IP addresses and distort the measures of entropy and one-to-one mapping. To solve this problem, we can investigate the use of NAT or DHCP in a network environment in advance, and avoid calculating the measures from the problematic source IP addresses.

IV. EXPERIMENTAL RESULTS

In this section, we evaluate the accuracy of anonymization and clustering on the application messages. Before discussing the experimental results, we first describe the data sets and evaluation methods.

A. Accuracy of Selecting Sensitive Fields for Anonymization

1) *Evaluation methods*: The packet traces for the evaluation are real network traffic from multiple sources, involving seven application protocols, HTTP, SMTP, FTP, SSH, DNS, WOW (World of Warcraft), and YMSG (Yahoo messenger), totally 2.6GB in size. The payloads in the protocols include ASCII characters and binary encoded information for testing with various types of messages. We manually label the sensitivity of 117 fields (extractable in Wireshark) in the protocols according to the field's value and the interpretation in the RFC documents, and 47 out of the fields are labelled sensitive after the manual inspection.

We select the C4.5 decision tree algorithm in Weka (www.cs.waikato.ac.nz/ml/weka) for classification because the other algorithms do not perform better after our trials. Table I lists the performance of the classification with 10-fold cross-validation, where the accuracy is measured with *precision* and *recall* defined as follows:

$$precision = \frac{|I_s \cap I'_s|}{|I'_s|}, \quad recall = \frac{|I_s \cap I'_s|}{|I_s|}, \quad (2)$$

where I_s denote the set of sensitive fields, and I'_s denote the set of fields that are classified to be sensitive.

The recall is high in the evaluation, implying that almost all sensitive fields can be found with the inference. In comparison, the precision is lower, implying some insensitive instances are incorrectly classified to be sensitive. Note that the false positives does not hurt the privacy, but may affect the utility for network security analysis. We will discuss this issue in the case studies in the next subsection.

2) *Case studies on false positive/negative assessment*: There are totally 12 misclassified instances of fields in the classification — two are false-negative (FN) instances and ten are false-positive (FP) instances. Table II lists the two FN instances and their measures.

The first FN instance is the `server` field in HTTP. According to RFC 2616, the field stands for the software version

TABLE I
THE ACCURACY FROM THE MODEL OF C4.5 DECISION TREE.

protocol (# of fields)	precision	recall
HTTP (31)	0.73	0.92
FTP (13)	0.92	1.00
DNS (9)	1.00	1.00
SMTP (21)	1.00	1.00
SSH (23)	0.20	0.50
WOW (11)	0.80	1.00
YAHOO (9)	1.00	1.00
All (117)	0.81	0.96

TABLE II
THE MEASURES OF THE FN INSTANCES.

protocol	field	entropy	diversity	one-to-one	sensitive
SSH	protocol	0.4559	0.1183	0.06	yes
HTTP	server	0.6785	0.3147	0.20	yes

of the server. The revelation may expose the vulnerability of the server program and incur attacks. The second FN instance is the `protocol` field of SSH, and this field usually contains the version of the operating system, and so it should be also anonymized. The reason of the false negatives is that multiple users have common server versions and the diversity is low in an objective perspective. An attacker can make a guess from popular server names and versions to exploit possible vulnerabilities, making the benefits of anonymizing the two fields somewhat controversial. Nonetheless, the false negatives can be easily eliminated by searching for server names and versions in the application payloads.

In contrast, the fields of FP instances contain the information of multiple digits such as timestamps, download rates, and sequence numbers. However, such fields may contain sensitive information such as telephone numbers and social security number (SSN), so we prefer to anonymize them even though false positives may occur. In practice, these fields seldom carry attack signatures, and affect little to the security analysis. The false positives are therefore still acceptable.

B. Accuracy of clustering

1) *Evaluation Methodology*: We also evaluate the accuracy of clustering for inferring the application fields. This work focuses on this stage because the alignment is largely correct according to our manual inspection, if the application messages can be correctly grouped in a cluster. This data set is composed of 900 SMTP responses with various response codes, which are rather complicated among real cases because multiple messages may be associated with the same response code, and they should be further divided into multiple clusters. We can adjust the similarity threshold for merge to achieve different performance.

We apply the methods in [17] to evaluate our clustering technique. Let $C = \{C_i\}_{1 \leq i \leq c}$ and $R = \{R_i\}_{1 \leq i \leq r}$ be two partitions of our data set, where C is the partition after the clustering, and R is the reference partition of the ground truth.

TABLE III
THE ACCURACY OF CLUSTERING FOR VARIOUS THRESHOLDS.

similarity threshold	precision	recall	F-measure
0.6	0.611	0.879	0.714
0.7	0.678	0.866	0.760
0.8	0.731	0.831	0.778
0.9	0.741	0.814	0.776

The precision and recall are calculated as:

$$prec(C, R) = \frac{1}{m} \sum_{i=1}^c |C_i \cap R_{f(i)}| \quad (3)$$

$$recall(C, R) = \frac{1}{m} \sum_{i=1}^r |C_{g(i)} \cap R_i|, \quad (4)$$

where

$$f(i) = \underset{i'}{argmax} |C_i \cap R_{i'}|, \quad g(i) = \underset{i'}{argmax} |C_{i'} \cap R_i|. \quad (5)$$

The precision and recall can be subsumed to the F-measure, which is the harmonic mean of the precision and the recall.

2) *Discussions*: Table III presents the accuracy of clustering. The threshold of 0.8 has better performance than the others. Increasing the threshold will result in many small clusters, whereas decreasing the threshold will increase the chances of unnecessary merges, and can significantly influence the precision value. The cluster sizes in the clustering can vary wildly, ranging from only two messages to more than 100 in large clusters. If the cluster size is too small after merging, we prefer to anonymize the entire messages directly in the clusters because they provide few hints for locating the fields with the alignment; otherwise, we proceed to locate the fields.

The F-measure is limited by two factors. First, some substrings in the application messages are rather short, and may also appear in the other messages, making the MCS and merging computations sometimes inaccurate. Second, the sorting prior to the clustering is based on the heuristic that the messages of similar semantics will be ranked together after sorting, but the heuristic may be inaccurate if the variant parts are in the first few bytes of the messages. For example, the first, the third and the fourth ones in the following messages have similar semantics, but they are separated after sorting because the variant parts (i.e., the domain names), rather than the command, are in the first few bytes of the messages.

```
221 2.0.0 cannoli.messagescreen.com closing connection
221 2.0.0 closing connection 14si3295638icg.127
221 2.0.0 fbmail003.sncl.facebook.com closing connection
221 2.0.0 fmgw.net.mexline.com closing connection
```

The solution is searching for similar messages throughout the entire set of messages during the clustering, but the clustering will become inefficient in the search process, and it is a trade-off between efficiency and accuracy.

V. CONCLUSION

Due to the diversity of application protocols in real packet traces and their rich semantics, packet anonymization is still a highly manual and error-prone process so far. The innovative

methods presented in this work can infer the sensitivity of application fields, and optionally infer the fields in the application messages without a parser. The evaluation demonstrates the recall of inferring sensitive application fields is as high as 96%, while the precision is acceptable. The inference can save a great amount of time for the network administrators to determine which fields are potentially sensitive and should be anonymized. For those application messages without a parser, this work provides a clustering and alignment method on application messages to identify the fields. The accuracy of clustering can be nearly 78%, but is still not high enough due to the complexity of the SMTP reply messages from a diversity of mail servers. We will keep on improving the clustering algorithm to increase the accuracy in the future work.

ACKNOWLEDGMENT

This work was supported by National Science Council (NSC) Project 99-2221-E-194-052.

REFERENCES

- [1] J. Sommers and P. Barford, "Self-configuring network traffic generation," In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement (IMC)*, Oct. 2004.
- [2] Know your enemy: passive fingerprinting, *HoneyPot Project*, Mar. 2002.
- [3] R. Pang, M. Allman, V. Paxson and J. Lee, "The devil and packet trace anonymization," *Computer Communication Review*, 36(1), pp. 29-38, Jan. 2006.
- [4] R. Ramaswamy and T. Wolf, "High-speed prefix-preserving IP address anonymization for passive measurement systems," *IEEE/ACM Trans. Networking*, 15(1), pp. 26-39, Feb. 2007.
- [5] 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 Intl. Conf. Network Protocols (ICNP)*, Nov. 2002.
- [6] Tcpanon, available at <http://www.ing.unibs.it/ntw/tools/tcpanon>.
- [7] R. Pang and V. Paxson, "A high-level programming environment for packet trace anonymization and transformation," In *Proceedings of ACM SIGCOMM*, Aug. 2003.
- [8] D. Koukis, S. Antonato, D. Antoniadis, E. P. Markatos and P. Trimintzios, "A generic anonymization framework for network traffic," In *Proceedings of IEEE Intl. Conf. on Communications (ICC)*, Jun. 2006.
- [9] Y. D. Lin, I. W. Chen, P. C. Lin, C. S. Chen and C. H. Hsu, "On campus beta site: architecture designs, operational experience, and top product defects," *IEEE Communications Magazine*, Vol. 48, Issue 12, Dec. 2010.
- [10] V. E. Seeberg and S. Petrovic, "A new classification scheme for anonymization for real data used in IDS benchmarking," In *Proceedings of the Intl. Conf. Availability, Reliability and Security (ARES)*, Apr. 2007.
- [11] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, 48(3):443-453, 1970.
- [12] W. Cui, V. Paxson, N. C. Weaver and R. H. Katz, "Protocol-independent adaptive replay of application dialog," In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS)*, Feb. 2006.
- [13] W. Cui, J. Kannan and H. J. Wang, "Discoverer: automatic protocol reverse engineering from network traces," In *Proceedings USENIX Security Symp.*, Aug. 2007.
- [14] J. Caballero, H. Yin, Z. Liang and D. Song, "Polyglot: automatic extraction of protocol message format using dynamic binary analysis," In *ACM Computer and Communications Security (CCS)*, Oct.-Nov. 2007.
- [15] M. A. Beddoe, "Network protocol analysis using bioinformatics algorithms," <http://www.baselineserach.net/PI>.
- [16] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. Voelker, V. Paxson, N. Weaver and S. Savage, "Botnet judo: fighting spam with itself," *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS)*, Feb. 2010.
- [17] P. Li, L. Lu, D. Gao and M. Reiter, "On challenges in evaluating malware clustering," In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sept. 2010.