On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization

Jun Xu Jinliang Fan Mostafa Ammar
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{jx,jlfan,ammar}@cc.gatech.edu

Sue B. Moon
Sprint ATL
1 Adrian Court
Burlingame, CA 94010
sbmoon@sprintlabs.com

1. MOTIVATION

Real-world Internet traffic traces are crucial for network research such as workload characterization, traffic engineering, packet classification, web performance, and more generally network measurement and simulation. However, only a tiny percentage of traffic traces collected are made public (e.g., by NLANR [6] and ACM ITA project [1]) for research purposes. One major reason why ISPs or other traffic trace owners hesitate to make the traces publicly available is the concern that the confidential (commercial) and private (personal) information regarding the senders and receivers of packets may be inferred from the trace. In cases where a trace has been made publicly available, the trace is typically subjected to an anonymization process before being released.

In this work we focus on the problem of anonymizing IP addresses in a trace. A straightforward approach is one in which each distinct IP address appearing in the trace is mapped to a random 32-bit "address." The only requirement is that this mapping be one-to-one. Anonymity of the IP addresses in the original trace is achieved by not revealing the random one-to-one mapping used in anonymizing a trace. Such anonymization, however, results in the loss of the prefix relationships among the IP addresses and renders the trace unusable in situations where such relationship is important (e.g., understanding routing performance or clustering of end-systems [5]). It is, therefore, highly desirable for the address anonymization to be *prefix preserving*. That is if two original IP addresses share a k bit prefix, their anonymized mappings will also share a k bit prefix. One approach to such prefix preserving anonymization is adopted in *TCPdpriv* [8].

The goal of our work is two fold. First, we are interested in analyzing the security properties inherent in prefix-preserving IP address anonymization. We aim to understand the susceptibility of prefix-preserving anonymization to attacks that may reveal some IP address mappings (e.g., [11]). Through the analysis of publicly available traces, we show that prefix-preserving anonymization will, in general, not pose a serious threat to personal privacy. Second, we observe that TCPdpriv has some drawbacks that limit its use in a large-scale, distributed setting. We develop an alternative cryptography-based, prefix-preserving anonymization technique to address these drawbacks while maintaining the same level of anonymity as TCPdpriv.

The rest of the paper is organized as follows. In Section 2, we explore the security implications of the prefix-preserving property using real-world traffic data. In Section 3, we discuss TCPdpriv and our cryptography-based technique, which addresses the drawbacks of TCPdpriv. Section 4 concludes the paper.

2. IMPLICATIONS OF PREFIX-PRESERVING ANONYMIZATION

The security implication of prefix-preserving anonymization of traffic traces using TCPdpriv [8] is briefly studied in [11] and [3]. Here we offer a more formal characterization of the security level that a prefix-preserving anonymization can achieve. There are a number of possible ways certain anonymized IP addresses could be compromised in the trace. For example, it may be possible for IP addresses of popular web servers be inferred from their high frequency of occurrence in the trace, or for the IP addresses of the DNS servers (especially authoritative servers) be inferred from the hierarchical relationship among them. Suppose certain anonymized addresses have been compromised so that their corresponding raw addresses are known. Then given any anonymized address y, the first $k+1$ bits of the corresponding raw address is revealed if the longest prefix match between the compromised addresses and y is k bits long. So the highest level of security we could possibly achieve is to make sure that the remaining $32-k-1$ bits look completely random, i.e., indistinguishable from a (32-k-1)-bit random number.

However, this only characterizes how an individual address is affected when some anonymized addresses are compromised. When we study the security of a trace, we would like to measure the amount of information that is leaked from the whole trace as a consequence. We define two metrics that characterize this. Suppose we are given a trace file, which contains

N distinct anonymized source/destination IP addresses, and each address is 32 bits long. One way to count the number of unknown bits in the trace is to add up all the unknown bits in these addresses, which we refer to as *uncompressed* unknown bits, denoted as U (=$32 * N$ initially). However, due to the prefix-preserving nature of the anonymization algorithm, certain bits in different IP addresses must be equal. If such bits are counted only once, we get another count C, which we refer to as *compressed* unknown bits[1]. We have the following geometric interpretation of C. If we represent distinct IPv4 addresses in a trace as leaf nodes on a binary tree of height 32, then C is count of all their ancestors in the tree. If an address is compromised, then obviously all ancestors (prefixes) of the leaf node that correspond to the compromised address are compromised and should be taken away from C. The two metrics are to show respectively the number of uncompressed (U) and compressed (C) nodes that remain unknown when x randomly chosen IP addresses in the trace are compromised.

We generated curves U and C using three publicly available IP address traces from NLANR [6]. Here, we only show the curves generated from the biggest trace (about 800 MB) that also contains the largest number of distinct IP addresses (130163). The curves from the other two traces are very similar to those that will be shown. Fig. 1(a) and 1(b) shows how U and C decrease as the number of compromised IP addresses (denoted as x) increases. Fig. 1(b) magnifies the portion of 1(a) when x ranges from 0 to 1000. The curve C drops almost linearly with respect to x, which is not surprising because it does not reflect how other addresses are affected by the compromised addresses. The curve U drops much faster than C because many other "innocent" addresses have their prefixes revealed which they share with the compromised addresses (x of them).

Each IP address can be broken down into following four groups of bits: bits 1 to 8, 9 to 16, 17 to 24, and 25 to 32. We denote the accounting of U on these four groups as U_1, U_2, U_3, U_4 respectively. As shown in Fig. 2(a) and 2(b), U_4, U_3, U_2, and U_1 drop at increasingly faster rate with respect to x (Fig. 2(b) zooms the beginning part of Fig. 2(a).). This is not surprising because lower order bits are much more likely to be shared by a large number of distinct IP addresses than higher-order bits. Fig. 2(a) and 2(b) show that U_3 and U_4 drop very slowly with respect to the number of addresses compromised, which means that the chance for the last 16 bits of an IP address to be revealed due to the prefix-preserving nature of the anonymization scheme is not high. This is indeed good news for privacy-conscious network users: the last 16 bits (typically host ID) in general are much more important for personal privacy than the first 16 bits (typically network ID). For commercial confidentiality, the first 16 bits could be more important, but that could be addressed in some other ways (e.g., releasing the trace after a year).

3. PREFIX-PRESERVING ANONYMIZATION SCHEMES

In the following, we describe TCPdpriv, an existing traffic anonymization tool that, among other things, allows the prefix-preserving anonymization of IP addresses. We describe how TCPdpriv is implemented and identify its drawbacks. We then discuss our cryptography-based prefix-preserving anonymization algorithm that does not have these drawbacks.

3.1 TCPdpriv

The implementation of the prefix-preserving translation of IP addresses is table based: it stores a set of <raw, anonymized> binding pairs of IP addresses to maintain the consistency of the anonymization. When a new raw IP address x needs to be anonymized, it will first be compared with the all the raw IP addresses inside the stored binding pairs for the longest prefix match. Let n be the number of bits in an IP address (32 in IPv4). Suppose the binding pair whose raw address has longest prefix match with $x = x_1x_2...x_n$ is $< x', y' >$ (let $x' = x'_1x'_2 \cdots x'_n$ and $y' = y'_1y'_2 \cdots y'_n$), in which $x_1x_2 \cdots x_k = x'_1x'_2 \cdots x'_k$ and $x_{k+1} = \overline{x'_{k+1}}$. Suppose x is anonymized to $y = y_1y_2 \cdots y_n$. Then $y_1y_2 \cdots y_ky_{k+1} := y'_1y'_2 \cdots y'_k\overline{y_{k+1}}$ and $y_{k+2}y_{k+3} \cdots y_n := RAND(0, 2^{n-k-1} - 1)$, where RAND is a pseudorandom (not necessarily cryptographically strong) number generator. If x is not identical to x', a new binding $< x, y >$ will be added to the binding table. It seems that it would take N comparisons to anonymize a new IP address where there are N binding pairs in the table. However, a data structure that is a compressed binary trie in nature is used to reduce the search cost to O(n). The memory requirement of the algorithm is to store $2 * N - 1$ trie nodes, where each node occupies 16 bytes. We refer readers to the source code of TCPdpriv [8] for the actual data structure and algorithm.

There are three major drawbacks of this implementation with respect to three different ways traffic traces need to be collected/anonymized for network research. First, for researches that involve large volume of network trace data (e.g., several days' worth), it is highly desirable for an operating router to pump out anonymized packet trace in real-time, thus saving time and effort for offline anonymization. However, the memory requirement can be high for a long trace with a lot of distinct IP addresses. For example, to anonymize a trace with 10 million different IP addresses in it, it would require approximately 320 MB of main memory space. Though this is affordable for state-of-the-art computers, such a requirement makes it unsuitable for high-speed hardware implementation inside a router. Second, TCPdpriv does not allow distributed processing of different traces simultaneously. The reason is that the translation between the raw and anonymized IP addresses is dependent on the sequence they appear in the trace. So when two different traces are being anonymized, the same raw IP address in general will not be translated into the same anonymized address. However, there is a real need for simultaneous (but consistent!) anonymization of traffic traces in different sites, e.g., for taking a snapshot of the Internet. It would be very cumbersome if hundreds of traces have to first be gathered and then anonymized in serial. Third, a large trace (e.g., terabytes) may be collected for a high-speed link for a long period of time. For the same reason discussed above, TCPdpriv does not allow a large trace file to be broken into pieces and be processed in parallel. Note that these drawbacks of TCPdpriv remain true even when prefix-preservation is not a requirement.

[1]This corresponds to the usual sense of entropy.

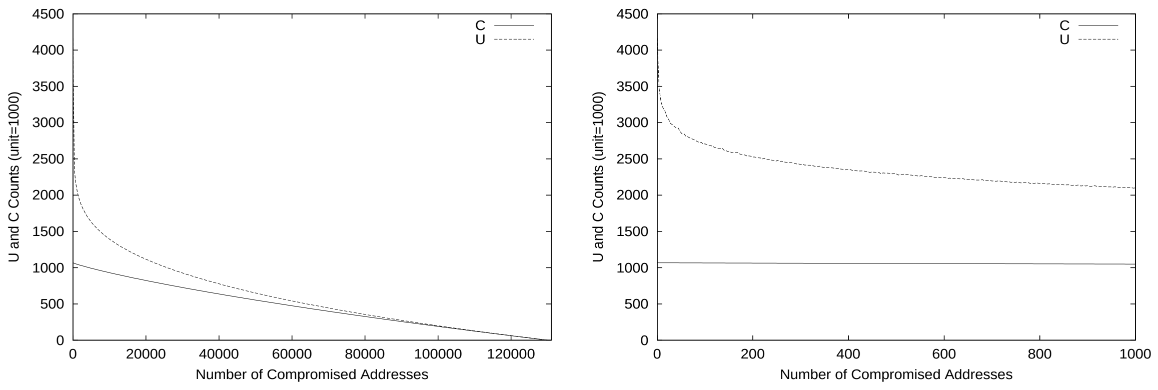

Figure 1: Measurements of U and C on an NLANR IP Address Trace (left is (a))

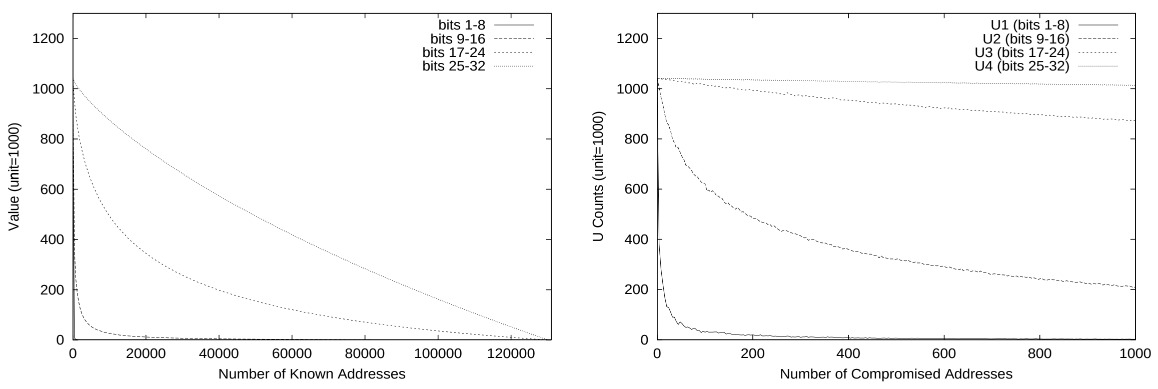

Figure 2: Measurements of U(1-4) on an NLANR IP Address Trace (left is (a))

3.2 Cryptography-based Approach

We designed an algorithm that addressed all the drawbacks of TCPdpriv. Our algorithm "computes" the binding between raw and anonymized addresses on the fly, so that its memory requirement is small enough to be fit into an on-chip cache. This makes it very amenable for hardware implementation (keyed hash functions such as HMAC-MD5 in general can be made quite hardware-friendly [9]). Also, since our algorithm is deterministic, it allows distributed and parallel anonymization of traffic traces. We will show that the algorithm is provably secure up to the level of security a prefix-preserving anonymization could possibly deliver. Before we describe our algorithm, we state the following proposition, the proof of which can be found in [10].

Proposition I. Let f_i be a function from $\{0,1\}^i$ to $\{0,1\}$, $i = 0, 1, \cdots, n-1$. Let F be the function from $\{0,1\}^n$ to $\{0,1\}^n$ defined as follows. Given $x = x_1 x_2 \cdots x_n$, then

$$F(x) := y_1 y_2 \cdots y_n \tag{1}$$

where $y_n = x_n \oplus z_n$ and $z_{i+1} = f_i(x_1, x_2, \cdots, x_i)$ for $i = 0, 1, \cdots, n-1$. We claim that (a) F is a prefix-preserving anonymization function, and (b) a prefix-preserving anonymization function necessarily takes this form.

In our algorithm

$$f_i(x_1 x_2 \cdots x_i) := LSB(KH(PAD(x_1 x_2 \cdots x_i), key_i)) \tag{2}$$

where $i = 0, 1, \cdots, n-1$. Here KH is a keyed hash function such as HMAC-MD5 [7, 4]. The security of our scheme hinges upon these keys (key_i) being kept secret. Obviously, for parallel and distributed anonymization, identical copies of the keys will be used by different processors/processes. PAD expands $x_1 x_2 \cdots x_i$ into a 512-bit string as follows:

$$PAD(x_1 x_2 \cdots x_i) := x_1 x_2 \cdots x_i || 10^d || i \tag{3}$$

Here d = 512 - $\lceil log(n) \rceil$ - 1 - i and 0^d means the repetition of 0 for d times. We set aside $\lceil log(n) \rceil$ bits for storing i. So the total length of the data after padding is 512 bits. This padding scheme is standard in using cryptographically strong hash function functions (e.g., MD5 [7]). It guarantees that for two strings x and x', if $x \neq x'$, then $PAD(x) \neq PAD(x')$. Without loss of generality, we also assume that the result of KH is 128 bits as in MD5 (i.e., $KH : \{0,1\}^{512} \rightarrow \{0,1\}^{128}$). This length of the result does not affect the security in *this application*.

Achieved Level of Security: Here we define the level of the security that is achieved by our anonymization function F as defined in (1)-(3). Suppose that a set of N anonymized addresses S have been compromised. Given an **arbitrary** anonymized address b (fixed after it is chosen), suppose k is the longest prefix match between b and the elements in S. Then, due the prefix-preserving nature of the anonymization algorithm, the first $k+1$ bits of the corresponding raw address, referred to as a, are revealed as mentioned before. However, our algorithm guarantees that the remaining $n-k-1$ bits are indistinguishable from random bits to computationally constrained adversaries, provided the underlying keyed hash functions (KH in (2)) are cryptographically strong. This result is made precise and proved in [10]. According to standard notions from the provable security literature [2], this is actually equivalent to saying that the algorithm F is indistinguishable from a **random prefix-preserving function**, a function uniformly chosen from the set of all prefix-preserving functions (characterized in Proposition I) from $\{0,1\}^n$ to $\{0,1\}^n$, to any computationally constrained adversary (also shown in [10]). This means that our algorithm achieves the highest level of security achievable by prefix-preserving algorithms when the adversary is computationally bounded.

4. CONCLUSION

In this paper, we motivate the need for anonymizing packet traces in a prefix-preserving manner and analyze its security implications using real-world traffic data. We show that TCPdpriv, the existing tool that allows for prefix-preserving anonymization, has drawbacks that make it unsuitable for parallel and distributed traffic anonymization. We propose a provably secure cryptography-based scheme that addresses these drawbacks. We expect that this work will help improve the availability of traffic traces for network research.

5. REFERENCES

[1] The Internet traffic archive. http://ita.ee.lbl.gov/, April 2000.

[2] M. Bellare. Practice-oriented provable-security. In *First International Workshop on Information Security(ISW97)*, Boston, Massachusetts, 1998. Springer-Verlag.

[3] K. Cho, K. Mitsuya, and A. Kato. Traffic data repository at the wide project. In *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, San Diego, CA, June 2000.

[4] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication.* Network Working Group, February 1997.

[5] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proc. ACM Sigcomm 2000*, pages 97–110, September 2000.

[6] Tony McGregor. NLANR active measurement project. In *Proceedings of Measurement and Analysis Collaborations Workshop*, June 1999.

[7] R. Rivest. The MD5 message-digest algorithm. Request for Comments 1321, Internet Engineering Task Force, April 1992.

[8] *TCPdpriv Command Manual*, 1996.

[9] J Touch. Performance analysis of MD5. In *sigcomm*, Boston, Massachusetts, 1995. ACM.

[10] J. Xu, J. Fan, M. Ammar, and S. Moon. On the design and performance of prefix-preserving IP traffic trace anonymization. Technical report, College of Computing, Georgia Tech, GIT-CC-01-22, August 2001.

[11] T. Ylonen. Thoughts on how to mount an attack on TCPdpriv's "-a50" option ... In *http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html*, 1996.