Network Log Anonymization: Application of Crypto-PAn to Cisco Netflows

Adam Slagell, Jun Wang, William Yurcik
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
{slagell, wangj, byurcik}@ncsa.uiuc.edu

***Abstract*— *Logs are one of the most fundamental resources to any security professional. It is widely recognized by the government and private industry that it is both beneficial and desirable to share logs for the purpose of security research and network measurements. Rapid growth of the Internet and its applications, especially financial and security related services, require a secure and efficient way to share network logs among security engineers and network operators. However, due to the risk of exposing sensitive information to potential attackers, the sharing is either not happening or not happening to the degree or magnitude that is desired. To eliminate the hurdle of sharing logs, strong and efficient anonymization techniques are needed. Crypto-PAn is a new anonymization tool which is gaining recognition. In this paper, we extend Crypto-PAn by designing and integrating it with an efficient passphrase-based key generation algorithm that requires no new libraries and little extra code. We also evaluate the performance of the extended Crypto-PAn on binary Cisco NetFlow log files.*

***Index Terms*—Anonymization, Audit Logs, NetFlow, Key Generation, Crypto-PAn**

I. Introduction

Because of the rapid growth of the Internet and the number of new network protocols and services it hosts, the number of attacks on critical network oriented services has increased dramatically. Consequently, network security has become one of the central and most active research topics in computer science and has even caught the attention of CxO's. Sharing network logs (e.g. TCP/IP traces or Cisco NetFlow logs) is essential to both network security operations and research, the value of which is recognized by both the government and industry. However, network logs contain much sensitive data.

The first step of any attack is reconnaissance. The attacker will try to map out the network and services running on particular hosts. Access to log files is like a treasure map for the attacker, reducing the amount of reconnaissance he must perform and often giving him more information than he could get on his own anyway. To reduce the exposure and still be able to share useful information, network and system administrators often wish to anonymize log files before releasing them.

There are dozens of common log types and several fields in each type. Consequently, there are many types of data to anonymize. While anonymizing a single field is often not enough even when it is the only *direct* identifier—because of relationships between other fields and logs—it is important to have building block functions to anonymize the different types of data. The most common identifier found in almost all network logs is the IP address, and it is on anonymization of the IP address that we focus in this paper. There are several ways just to anonymize the IP address data type, and we examine the most common methods and some of the tools to perform IP address anonymization.

In section II we examine IP address anonymization algorithms and introduce existing IP anonymization tools. In section III, we give a deeper treatment of a well-known algorithm called Crypto-PAn [1], [2]. Furthermore, we describe our passphrase-based key generation algorithm that we integrated into the original Crypto-PAn library. In section IV , we apply the extended tool to real NetFlow [3] log files and evaluate its performance, and we conclude in section V.

II. Existing Anonymization Algorithms/Tools

There are four basic types of IP address anonymization algorithms currently in use. The most trivial method is what we call "black-marker" anonymization. Here one simply replaces all IP addresses with a constant. It has the same affect as simply printing the log and blacking-out all IP addresses. This method loses all IP address information and is completely irreversible. The set of all IP addresses is essentially collapsed to one address. While this method is simple, it is quite undesirable because it does not allow correlation of events perpetrated against a single host.

This work is funded in part by a grant from the Office of Naval Research (ONR) under the umbrella of the National Center for Advanced Secure Systems Research (NCASSR).

Random permutations are another mechanism to anonymize IP addresses. This method creates a one-to-one correspondence between anonymized and unanonymized addresses that can be reversed by one who knows the permutation. This method loses less information as now actions against or by a particular host can be grouped together. However, to anyone who does not know the mapping, all information about subnets—or even what continent a host is on—is lost.

Truncation is probably still the most popular method of anonymizing IP addresses. Here a fixed number of bits is decided upon (typically 8, 16 or 24), and everything but those first bits are set to zero. For example, the Internet Storm Center (ISC) [4] reports source IP addresses of port scanners by truncating all but the first, 8 bits thus leaving one with only information about the class A network from which a scan originated. Again, a lot of information is lost here, and such mappings are not reversible because the mapping is not injective; many addresses map to a single anonymized address. This has the same problems as the "black-marker" approach (In fact, the "black-marker" method is just a degenerate case of truncation where all bits are truncated.), except one can choose how much information is retained with a rough granularity . For example, one may be able to tell from which subnet an attacker came without knowing an exact IP address. However, this may be a very large subnet depending on how many bits are truncated from the addresses.

The newest form of IP address anonymization is actually a type of pseudonymization. Pseudonymization is a type of anonymization that uses an injective mapping such as random permutations. The value to which an IP address maps is called a pseudonym. In the case of a permutation, both the input and output are real IP addresses. However, a pseudonym could be from a completely different set. For example, the IP address 10.3.33.4 could map to "Bob", and that is a perfectly valid pseudonym.

The newer form of anonymization to which we are referring is called *prefix-preserving pseudonymization*. In prefix-preserving anonymization, IP addresses are mapped to pseudo-random anonymized IP addresses by a function we will call τ. Let $P_n()$ be the function that truncates an IP address to n bits. Then τ is a *prefix preserving* permutation of IP addresses if $\forall\ 1 \leq n \leq 32$, $P_n(x) = P_n(y)$ if and only if $P_n(\tau(x)) = P_n(\tau(y))$.

Significant work has been accomplished on prefix-preserving anonymization of IP addresses [5], [6], [7], [8]. The benefit of this type of anonymization is that the structure of the networks and subnets are completely preserved while at the same time unknown. So while a recipient of such an anonymized log may be able to tell if the same subnet receives a lot of scans from the same attacker, neither the target nor the subnet it is on are revealed by real addresses. This type of anonymization is quickly replacing truncation techniques due to its structure preserving properties. It should be noted that while this property gives many benefits and without context provides perfect protection while retaining more information, there are attacks that make use of context around the IP address to exploit this structure. However, such attacks are outside the scope of this paper.

In Table I we list several prefix-preserving IP anonymization tools and indicate what sort of anonymization technique they utilize.

III. Extension to Crypto-PAn

Crypto-PAn is a prefix-preserving anonymizer provided by Xu et al [1], [2]. It is an improvement over TCPdpriv which must construct tables that are dependent upon the trace to do prefix preserving anonymization of IP addresses. Tables are determined by the trace structure and the pseudo-random generator being used. Thus different traces created by different machines will create distinct tables, and hence create different mappings between real IP addresses and anonymized addresses. As a result, TCPdpriv does not allow work to be done in parallel if it is required that the mappings be consistent across the different pieces.

Alternatively, Crypto-PAn does not create tables but instead mappings are uniquely determined by a passphrase and the symmetric block cipher used by Crypto-PAn. Any block cipher with a reasonable block size can be used. This makes it simple to do computations in parallel at different nodes. So a device could be responsible for anonymizing its own logs. Rather with TCPdpriv, every device would have to forward unanonymized logs to a central repository for anonymization.

One thing missing from the Crypto-PAn anonymizer class is a key generator. It takes a key in hex as input, but it does not generate keys. We found it useful to be able to specify a passphrase rather than distributing a large hex value to different people and devices. A similar limitation was in fact a big complaint about much of the older Wi-Fi hardware that supported WEP. Thus we integrated a key generation algorithm into the software which we then applied to Cisco NetFlow traces.

We setup the software to work on the binary data of the Cisco NetFlow logs directly. We could have simply compiled the code with few changes to take a dotted decimal IP address through STDIN with the key as an

TABLE I

DESCRIPTION AND COMPARISON OF DIFFERENT ANONYMIZATION TOOLS/ALGORITHMS

Tool/Algorithm	Description	Webpage	Reference
Crypto-PAn	A cryptography-based sanitization tool to anonymize IP addresses in network traces in a prefix-preserving manner.	http://www.cc.gatech.edu/computing/Telecomm/cryptopan	[1], [2]
TCPdpriv	A table-based prefix-preserving IP address anonymizer that operates on TCPdump files	http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html	[9], [10]
ip2anonip	A filter to turn IP addresses into host names or anonymous IPs based on TCPdpriv.	http://dave.plonka.us/ip2anonip	
ipsumdump	A program that summarizes TCP/IP dump files into a self-describing ASCII format easily readable by humans and programs. It has an option to use a prefix-preserving pseudo-nymization algorithm based on TCPdpriv.	http://www.icir.org/kohler/ipsumdump	

TABLE II

SAMPLE OF COMMA DELIMITED ASCII NETFLOW FILE

Version	Start time	End time	Src address	Src port	Dst address	Dst port	Bytes	Packets	IP	TCP
5	1036168575520	1036168575520	165.26.148.220	1352	141.142.106.54	4354	70	1	6	24
5	1036168575724	1036168575724	165.26.148.220	1352	141.142.106.34	1318	70	1	6	24
5	1036168581616	1036168581616	137.230.24.106	139	141.142.106.44	1235	93	1	6	24
...	...	...	...	...	...	...	...	...	...	...

argument. It would then send the anonymized IP address to STDOUT. A perl script could parse the log and call this C++ binary to do anonymization on every IP address. However, this is somewhat inefficient as the Crypto-PAn code works on IP addresses in a binary form. If one uses this method they have overhead from perl parsing the log and calling the external program. Additionally, they end up converting the log to ASCII, the IP addresses back to binary, and then back again to ASCII. It is simpler to operate on the binary log directly and then convert everything to ASCII just once. The only drawback is that such a solution is customized to the log format, whereas a perl script that finds dotted decimal IP addresses in ASCII logs can be written very generally to apply to many log types. Our code depends upon the IP address being at a specific place in the log entry.

A. Key Generation

We created a routine which takes as input a passphrase and creates a 32 byte key. We wanted to reuse as much code as possible, so we did not want to want to add a library and use a hash function. So we used Rijndael, which is already used by the anonymizer, to create a specialized hash function. Below we describe this algorithm.

1) The user input (passphrase) is read but not echoed back to the terminal.
2) The input is cut off after 256 bytes, or if it is shorter than that 256 bytes it is repeated an appropriate number of times to fill a 256 byte buffer. This buffer serves as a seed.
3) The 256 byte seed must next be combined to create an intermediate Rijndael key. A simple way to do this would be to take each 32-byte sub-block of the buffer and XOR them all together. But if the passphrase length is a divisor of 32, then each 32-byte block would be identical. XOR'ing the sub-blocks together an even number of times would create a key of all zeros! It is clear that this null key would occur much more commonly than many other possibilities.
   Another approach would be to take the last 32-byte block of the buffer as the key. But if the input is larger than 32 bytes, then we are really discarding some of the entropy entered by the user. Treating all of these sub-blocks of the buffers as integers and performing mathematical operations to combine them is another option.
   However, we chose to combine the data in a way certain to disturb any of the periodic behavior in the 256 byte buffer; we decided to CBC encrypt the buffer using a fixed Rijndael key of 32 bytes. Now the last 32 bytes of the buffer are affected by all previous blocks because of the chaining in CBC mode encryption. We take these last 32 bytes to form

an intermediate key.

4) It would be risky stopping here and simply using the intermediate key because the fixed key that is part of the source code would be known to adversaries. And thus every step performed is reversible except for the last operation which discarded 244 bytes of data. It makes us uneasy to have a mostly reversible algorithm since we do not want a compromised key to reveal the passphrase or anything about it because people will often use the same passphrase for many applications. So we use this intermediate key to CBC encrypt the original buffer once more. The last 32 bytes of this encrypted 256 byte buffer now forms the final key. This is irreversible even if we reveal the other 244 bytes of the buffer because the attacker who knows this final key, does *not* know the intermediate key which depends upon the input. And without this key, the encryption operation cannot be reversed.

To summarize, we apply a cryptography function to the input data to create an intermediate key. Then we use this intermediate key to encrypt the input data. The final key comes from this encrypted data.

IV. Experiments: Practice with Cisco NetFlows

In this section, we will first give a brief introduction to Cisco NetFlows. Then, we will describe our experiments of applying the extended Crypto-PAn anonymization method to example Cisco NetFlow traces. Finally, we will show the results.

A. Cisco NetFlow

Cisco NetFlow [3] logs contain records of unidirectional flows between computer/port pairs across an instrumentation point (e.g. router) on a network. Ideally, there is an entry per socket. These records can be exported from routers or software such as ARGUS [1] or NTOP [2]. NetFlows are a rich source of information for traffic analysis consisting of some or all of the following fields depending on version and configuration: IP address pairs (source/destination), port pairs (source/destination), protocol (TCP/UDP), packets per second, time-stamps (start/end and/or duration) and byte counts.

Table II shows sample entries of a comma delimited ASCII NetFlow file.

[1] ARGUS <http://www.qosient.com/argus/>
[2] NTOP <http://www.ntop.org>

B. Experiments and Results

We applied the extended version of Crypto-PAn (with our new key generation algorithm) to NetFlow trace files obtained within the NCSA at the University of Illinois. We generate on the order of two Gigabytes of flow data every day, and hence need a fast anonymization system. We tested our code on three different machines: Machine A - a dual processor 2.4 GHz Xeon, Machine B - a single processor 2.4 GHz Xeon, and machine C - an IBM T41 notebook with a 1.7 GHz Pentium - M. Our results include both the processing time and file I/O. For the dual processor machine we split the data in half and ran two concurrent processes to perform timings. The Xeon machines are both Dell N series workstations, and all machines have 1 GB of main memory.

Clearly machine B was slower than machine A with rates of 42686.279 records per second compared to 75015.342 records per second. Machine C, the laptop, was the slowest with a rate of 40113.674 records per second. The total time required to process a 2 GB log was still only 19 minutes and 33 seconds on the slowest machine. Even the workstations are not that fast or uncommon in our environment, and thus we conclude that anonymization of all our NetFlows is very feasible.

While we may not share our anonymized NetFlows with any party (because there is more sensitive data in a NetFlow than just IP addresses and attacks can exploit other unanonymized fields to attack the prefix-preserving anonymization scheme), we will share them with students doing research at the NCSA or the computer science department of the University of Illinois. Other potential partners for sharing include other supercomputing centers (e.g. San Diego Supercomputing Center) and Department of Energy Labs (e.g. Argonne National Labs).

V. Conclusion

We at the NCSA are beginning to share network and security logs with researchers and students. We have started by sharing our Cisco NetFlow logs. However, like most network logs, NetFlows are abundant with sensitive information, the most sensitive fields being incoming and outgoing IP addresses. As such we have analyzed different methods of anonymizing IP addresses. In this paper we discuss four basic types and focus on tools that provide prefix-preserving pseudonymization. We found Crypto-PAn to be the most superior due to its ability to be parallelized. However, the source code provided does not include a key generator. We developed a novel key generator that required no additional cryptographic libraries to extend the Crypto-PAn tool. Using this extended tool we created a NetFlow anonymizer that

operates on binary traces. We have shown that a typical workstation can easily handle the anonymization of our daily flow records which order around two Gigabytes a day.

VI. Acknowledgments

We would like to thank Kiran Lakkaraju for help in coding some of the timers to run benchmarks. We also thank Jim Barlow for providing sample netflows.

References

[1] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon, "On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization," in *ACM SIGCOMM Internet Measurement Workshop*, 2001.

[2] 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 *IEEE International Conference on Network Protocols (ICNP)*, 2002.

[3] Cisco Systems White Paper, "NetFlow Services and Applications," http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps-wp.htm.

[4] Port Report, "Internet Storm Center - ISC," http://isc.incidents.org/port_report.php.

[5] R. Pang and V. Paxson, "A High-Level Programming Environment for Packet Trace Anonymization and Transformation," in *ACM SIGCOMM*, 2003.

[6] M. Peuhkuri, "A Method to Compress and Anonymize Packet Traces," in *ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2001.

[7] M. Allman, E. Blanton, and W.M. Eddy, "A Scalable System for Sharing Internet Measurements," Passive and Active Measurement (PAM) Workshop, 2002.

[8] C.J. Antonelli, M. Undy, and P. Honeyman, "The Packet Vault: Secure Storage of Network Data," USENIX Workshop on Design Issues in Anonymity and Unobservability, 2000.

[9] G. Minshall, "TCPdpriv: Program for Eliminating Confidential Information from Traces," Ipsilon Networks, Inc.

[10] G. Kuenning and E. Miller, "Anonymization Techniques for URLs and Filenames," Technical Report UCSC-CRL-03-05, University of California, Santa Cruz, Sep. 2003.