Sharing Network Logs for Computer Forensics: A New Tool for the Anonymization of NetFlow Records

Adam J Slagell, Yifan Li and Katherine Luo
National Center for Supercomputing Applications
University of Illinois
{slagell, yifan, xluo1}@ncsa.uiuc.edu

I. Introduction

The world is becoming more interconnected every day. The Internet and related technologies are allowing us to work in new and more efficient ways. Consequently, new network aware applications are appearing constantly. Unfortunately, new methods of interconnection between systems introduce new attack vectors. Thus we are seeing a proportional growth in computer and network attacks. The Internet gives attackers more anonymity—at least a feeling of more anonymity—which in turn has made them bolder than traditional criminals. As a result, system and network administrators require better tools for digital forensics and log analysis as well as opportunities for training and education. While at first counter-intuitive, we argue that anonymization techniques are vital to create better tools for digital forensics and trace-back. This follows since log sharing is necessary for the development of forensic tools, and anonymization is necessary for wide-spread log sharing.

Academia, industry [7] and government [5], [10] have all recognized the importance of sharing logs. Both industry and academia have a keen interest in developing tools for digital forensics and log analysis. It is a straightforward conclusion that access to larger and more diverse log data sets enable the development of more refined tools. Educators in traditional academia, as well as those that train security professionals and network administrators in forensics (e.g. SANS (SysAdmin, Audit, Network, Security) Institute), also require access to logs. Again, diverse data sets are desired to create meaningful student projects and instructional materials. Lastly, there is a desire to create log repositories for network security research and network measurement studies.

The reluctance to share logs—which has resulted in fewer entities sharing—is understandable due to the sensitivity of the information contained within logs. The first phase of any digital attack is reconnaissance, and logs are like a treasure map for would-be attackers. Access to computer and network logs can provide intruders special views of a network not visible from the outside, even with scanning tools. Information gleaned from these logs could indicate potential bottlenecks for DoS attacks or could even contain passwords, especially if insecure logins (e.g. telnet and FTP) are allowed on LANs. Logs can even be used to identify machines infected by worms, which are most likely not well-maintained or monitored. Soft targets such as these can be used to get a foothold into a network.

While some would claim that the difficulties in sharing logs is social, we believe the problems are technical at the heart. To share logs and address these concerns about the sensitivity of the information within them, anonymization techniques must be employed. However, the field of log anonymization is still immature. There are very few tools, and the ones that exist are deficient in many ways. Further, current anonymization tools are one-size-fits-all, or better put one-size-tries-to-fit-all. Ideally, they would support multiple levels of anonymization that trade-off between security—of the anonymization scheme—and utility—the amount of useful information retained in anonymized logs. Thus far, no log anonymizers support multiple levels of anonymization, even though there are typically multiple levels of trust between parties who might wish to share logs and those with whom they would share their logs.

We have begun to address this problem with the development of a new prototype tool CANINE: Converter and ANonymizer for Investigating Netflow Events. Originally just a NetFlow converter, CANINE has been adapted to anonymize 8 of the most common fields found in all NetFlow formats. Most of these fields can be anonymized in multiple ways providing trade-offs between security and utility. This is the first tool we are aware of that supports many levels of anonymization and is the only NetFlow anonymizer of which we are aware—besides a previous, less advanced tool we developed [12].

The paper is organized as follows. Chapter 2 discusses related work in log anonymization. Chapter 3 discusses CANINE's anonymization algorithms and design decisions in depth. Chapter 4 concludes and presents future work on CANINE and the anonymization of other log types.

II. Related Work

While there has been some research into log anonymization, most work addresses only a small subset of all the available log sources (usually just network traces) and focuses exclusively on anonymizing IP addresses within a log. Because even the most advanced log anonymization solutions must still perform IP address anonymization—since it is a sensitive field common to most logs—we discuss IP address anonymization in detail.

A. IP Address Anonymization

There are four basic types of IP address anonymization algorithms currently being used—illustrated by an example in table 1. 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 particular host.

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 only by one who knows the permutation. This method loses less information as now actions against or by a particular entity 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) [6] 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, albeit not a very useful pseudonym.

The newer form of anonymization to which we are referring is called *prefix-preserving pseudonymization*. In prefix-preserving pseudonymization, 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 [15], [16]. The benefit of this type of anonymization is that the structure of the networks and subnets are completely preserved while at the same time the actual subnets are 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. 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 strong protection while retaining more information, there are attacks that make use of context around the IP address to exploit this structure. As such, prefix-preserving anonymization is not appropriate in all circumstances.

TCPdpriv is a free program that performs prefix-preserving IP anonymization of TCPdump traces using tables. Because of the use of tables, it is difficult to process logs in parallel with this tool. In [15], [16], Xu et al. have created a prefix-preserving IP pseudonymizer that overcomes this limitation by eliminating the need for centralized tables to be shared and edited by multiple entities. Instead, with their algorithm Crypto-PAn, one only needs to distribute a short key between entities that wish to pseudonymize consistently with each other. We have reimplemented the Crypto-PAn algorithm in Java and added a key generator. This Java class is used in CANINE. Previously [12], we implemented a much simpler NetFlow anonymizer that anonymized just IP addresses of just one type of NetFlow and used the C++ class developed by Xu et al. with our own key generation algorithm.

B. Current Log Anonymization Attempts

Pseudonyms are an important building block to many anonymization schemes. One of the earliest uses of pseudonyms can be found in [3] where public keys are used as pseudonyms. We now recognize what Chaum described in [3] as a "digital pseudonym" to be a specific type of pseudonym called an authorization certificate. As noted in [9], pseudonyms help define middle ground in the zero-sum tradeoff between security and privacy of audit logs. In [13], Sobirey et al. first suggested privacy-enhanced intrusion detection using pseudonyms and provided the motivation for the work of Biskup et al. in [1], [2].

While the work in [9], [1], [2] does deal with log data and anonymization, their goals are significantly different than ours. All three works deal specifically with pseudonymization in Intrusion Detection Systems (IDSs). The adversary in their model is the system administrator, and the one requiring protection is the user of the system. In our case, we instead assume that the system/network administrators have access to raw logs, and we are trying to protect the systems from those who would see the shared logs. To contrast how this makes a difference, consider that in their scenario the server addresses and services running are not even sensitive—just information that could identify clients of the system. Furthermore, we do not have a need to support pseudonym reversals. They have this need since the system/network administrators do not have raw data in their case, and hence the privacy officer must help the system security officer reverse pseudonyms if alerts indicate suspicious behavior. In [1], [2], they take this further and try to support automatic re-identification if a certain threshold of events is met. In that way their pseudonymizer must be intelligent, like an IDS, predicting when re-identification may be necessary and thus altering how it pseudonymizes data. They also differ from us in that they create transactional pseudonyms, so a pseudonym this week might map to a different entity the next week. We, however, desire consistency with respect to time for data-mining across logs. Lastly, all of the anonymization solutions in these papers filter log entries and remove them if they are not relevant to the IDS; we endeavor to dispose of no entries because

TABLE I
EXAMPLE OF 4 COMMON METHODS TO ANONYMIZE IP ADDRESSES

IP	Black-marker	Random Permutation	Truncation	Prefix Preserving
141.142.96.167	c	141.142.132.37	141.142.0.0	12.131.102.67
141.142.96.18	c	141.142.96.167	141.142.0.0	12.131.102.197
141.142.132.37	c	12.72.8.5	141.142.0.0	12.131.201.29
12.161.3.3	c	212.3.4.1	12.161.0.0	187.192.32.51
12.72.8.5	c	141.142.96.18	12.72.0.0	187.78.201.97
212.3.4.1	c	12.161.3.3	212.3.0.0	31.197.3.82

completeness is very important for logs released to the security research community. If certain data cannot be anonymized, the user must be alerted to that fact.

In [4], Flegel takes his previous work in privacy preserving intrusion detection [1], [2] and changes the motivation slightly. Here, he imagines a scenario of web servers volunteering to protect the privacy of visitors from themselves, and he believes IP addresses of visitors need pseudonymization. However, to a web server IP addresses already act as a pseudonym protecting the client's identity since ISPs rarely volunteer IP address-to-person mappings. Though the motivation differs slightly, the system described is the same underlying threshold based pseudonymization system, and the focus of this paper is really about the implementation and performance of the system. As such, the results of [4] can be applied to [1], [2].

In [11], Pang et al. developed a new packet anonymizer that anonymizes packet payloads as well as header information, though their methodology only works with the few application level protocols that their anonymizer understands (HTTP, FTP, Finger, Ident and SMTP). The process also alters logs significantly, losing fragmentation information, the size and number of packets and information about retransmissions, thus skewing time-stamps, sequence numbers and checksums. While their anonymizer is limited in its capabilities, it is fail-safe because it only leaves information in packets that it can parse and understand. Further, they create a classification of anonymization techniques and a classification of attacks against anonymization schemes. We use a similar classification based partly off of their work.

Waters et al. [14] address the tension between data access control and searchability of audit logs through a new method they developed to search asymmetrically encrypted logs. In this way the encrypted log can be made public for search, and the owner distributes private keys corresponding to keywords. Thus, instead of the data owner decrypting the log and running the search, he can simply give the query maker the ability to perform the query with a set of keywords he deems acceptable.

Most recently, Lincoln et al. in [8] proposed a log repository framework that enables community alert aggregation and correlation, while maintaining privacy for alert contributors. However, the anonymization scheme in the paper is partially based on hashing IP addresses. Such a scheme is always vulnerable to dictionary attacks. Moreover, their scheme mixes the hashes with HMACs and truncates the hashes/MACs to 32 bits, both actions which result in more hash collisions an inconsistent mappings. Finally, their suggested use of re-keying by the repository destroys the correlation between repositories and therefore limits the view to a single repository.

III. A NEW TOOL FOR ANONYMIZING NETFLOW RECORDS

CANINE (Converter and ANonymizer for Investigating Netflow Events) is a unique, new tool that anonymizes NetFlows to multiple levels. It handles Argus, NCSA Unified Format, Cisco 5, Cisco 7 and Cisco 5/7 mixed logs. Multiple levels of anonymization that trade-off between the security of the anonymization scheme and the utility of the anonymized logs is supported through multiple algorithms that anonymize the 8 most common NetFlow fields (Source IP Address, Destination IP Address, Source Port, Destination Port, Start Timestamp, End Timestamp, Protocol and Byte Count) in different ways. This is the only log anonymizer that we are aware of that supports multiple levels of anonymization, and it is the only NetFlow anonymizer we know of besides a previous tool we created that just anonymized IP addresses in one NetFlow format [12].

A. Source and Destination IP Addresses

We have implemented all of the IP address anonymization methods discussed in Section II.A. Truncation is the most basic type of IP address anonymization we support. Here the user chooses the number of least significant bits to truncate from an IP address by use of a sliding bar. For example, truncating 8 bits would simply replace an IP address with the corresponding class C network address. Truncating all 32 bits would replace every IP with the constant address of "0.0.0.0"—what we have also referred to as "black-marker" anonymization. Truncation is probably the most common type of log anonymization currently employed, and its use on a log is readily apparent. It is also a very safe option, but this security comes at a high cost. Truncation can lose a lot of information and makes it impossible to correlate attacks against a particular host within a log.

We also support anonymization by creating random permutations on the set of possible IP addresses. This permutation is then applied to each IP address in the log. We implement this algorithm through use of two hash tables for efficient lookup. One hash table is used to store mappings from unanonymized to anonymized IP addresses. The other hash table is used to store all of the anonymized addresses so we can check if an address has already been used when creating a new mapping. Because the first table is indexed by unanonymized addresses, the whole

table would have to be searched for a free anonymized address if we did not use a second hash table. In this way, we trade a little storage for a large computational speed-up.

The main drawback to this implementation of a random permutation is that these tables make it difficult to anonymize in parallel or to anonymize consistently between program executions. Tables would need to be stored securely and accessed the next time CANINE is run. This is hardly a smooth process, and so we do not support such a feature. Thus one cannot data mine between logs anonymized in this way if they are indexing the IP address field. Alternatively, we could have used a 32 bit block cipher where the key deterministically defines the permutation. Then only a key would need to be stored or remembered. However, there are no strong 32 bit block ciphers, and none have been used in at least 30 years. For those requiring consistency between pseudonyms on different runs, we recommend prefix-preserving anonymization which is weaker from an information theoretic and cryptographic point of view.

Also, note that it is more difficult to infer the type of anonymization used by just looking at the anonymized log in this case, where as it should be quite obvious from looking at a log with truncated IP addresses. With many types of anonymization, it is not apparent that anonymization was even used. For this reason it is important to create meta-data to indicate how the logs were anonymized. We are currently considering how to represent types of anonymization in XML.

Prefix-preserving pseudonymization uses a special type of permutation that has a unique structure preserving property. The property is that two anonymized IP addresses match on a prefix of n bits if and only if the unanonymized addresses match on n bits. This preserves subnet structures and is often preferred to random permutations, but the simple fact that there are many times fewer permutations of this type makes it weaker. As a general principle, cryptographic algorithms that preserve structure are more open to attack, and this algorithm is not an exception. For example, an adversary that injects traffic to be recognized later not only gleans information about the addresses she specifically attacked, but she also learns many of the unanonymized bits of addresses that share prefixes with the addresses she attacks. For this type of anonymization we implemented the Crypto-PAn [15], [16] algorithm and generate keys by hashing a passphrase the user provides. In this way, tables are not used, and logs can more easily be anonymized in parallel across different locations. We recommend this type of anonymization when the adversary that is likely to attack the logs is weak, and the utility of the log and its structure is very important.

B. Source and Destination Port Numbers

We support 2 methods of anonymization for port numbers. First we implement "black-marker" anonymization, which is the same, from an information theoretic point of view, as printing the log and blacking out all port numbers. In a digital form, we just replace all port numbers with a constant. Port 0 is a good candidate for the constant. However, we must use a 16 bit representation for 0 so that programs which process unanonymized logs can still process anonymized logs. We take this same approach with all anonymization algorithms, ensuring that anonymization does not break other forensic and log analysis tools that process these NetFlows.

The second method of port number anonymization we support is called bilateral classification. Usually the port number is useless unless one knows the exact port number to correlate with a service. However, there is one important piece of information that does not require one to know the actual port number: whether or not the port is ephemeral. In this way ports can be classified as being below 1024 or above 1023. To make the output the same format as the input, a representative of the set, such as port 0, can replace all non-ephemeral ports, and 65535 can replace all ephemeral port numbers. This is very similar to IP address truncation in that we are collapsing a sets of numbers down to single representatives within those sets. Again, we do this so that anonymized logs do not break any log analysis tools.

C. Timestamps

Time stamps can be broken down into the units of *Year, Month, Day, Hour, Minute and Second*. We currently support 3 algorithms to anonymize timestamps. In a method we call time unit annihilation, we support the annihilation or "zeroing-out" of any subset of the afore mentioned units. If one wishes to remove the hour, minute and second information, they can do so (This is similar to truncation in IP address anonymization). Likewise, if someone wishes to obfuscate the date, they can remove the year, month and day information. If they want to completely eliminate time information, i.e. "black-marker" anonymization, they can select all of the time units for annihilation. Starting times are adjusted so that the duration of the flow is kept the same.

Notice that depending on the units annihilated, records can become quite disordered. For example, if the date information is removed, and the log spans multiple days, then one will see records start around midnight and keep increasing in time. When the next day is encountered, things will reset to midnight so that it appears the chronology moves back 24 hours. In this way, one may not be able to discern the day an event occurred, but they could tell if one event occurred 1 or two days later in the log than another event. This may be a desired property, but if an adversary can determine the date for one record, they can figure it out for all records. If this is giving away too much information, the log could be sorted after this anonymization method is used.

In some cases it may be important to know how far apart two events are temporally without knowing exactly when they happened. For this reason a log or set of logs can be anonymized at once such that all timestamps are shifted by the same random number—this number represented in seconds. In CANINE we call this method of anonymization a random time shift. If one does this to two different sets of logs at different times, then this random number will be different between the data sets. This means that data mining can not be done by indexing the time field between the data sets. The solution requires the ability to choose the number by which to shift. However, it seems cumbersome and impractical for data owners to save and keep track of shifting amounts used on different logs. That is why we do not support that ability but instead warn users to be aware of

the troubles with data mining (by indexing the time stamp) between sets anonymized at different times when using this specific method of anonymization.

In implementing this method we allow the user to select a range from which to have the random shift chosen. A number will not be chosen such that the random shift will overflow the timestamp field and cause wrap-around. Also, notice that this method would be hard to detect without metadata. If the random shift takes the log far into the future its use could be inferred, but metadata would really be needed to indicate its use in a general. Lastly, this method can still fall prey to a simple data injection attack on the timestamp field. One could scan a network with a special sequence determining the length of times between probes. The length of times between probes will still be apparent in the anonymized log since the relative time between events remains unchanged. For those requiring more security, we recommend the next method.

Alternatively, all information could be removed except the order in which the events occurred. In this method, which we simply call enumeration, a random time for the first record is chosen. All other ending times are equidistantly spaced apart from each other and in chronological order. Corresponding starting times are calculated from the original flow duration. Implementation of this method can be troublesome, especially when dealing with streamed data. The problem arises because entries in the logs are not presorted by ending time. They are close, but not in perfect order. Sorting cannot work perfectly on streamed data, and it would be extremely slow on large log files. The solution we use is to buffer events to sort locally. Since events are never terribly disordered, this sorts records with great accuracy. If data is from multiple routers, there will likely be small errors in this regard anyway, due to time skews between routers.

D. Protocol Number and Byte Count

We only support one method of anonymization for both the protocol number and byte count: "black-marker" anonymization. We simply replace the field with a value that would be impossible in practice. For example, when anonymizing the protocol number, we replace all values with the unused, but IANA reserved, number of 255. When anonymizing byte counts for the flow we replace all byte counts with 0, an impossible value in practice since headers will at least account for some bytes in the flow.

IV. CANINE Performance

CANINE scales extremely well. Every anonymization operation is bounded in constant time, and thus CANINE scales linearly with respect to the number of records—consequently also with respect to the size of the log since the size of the log is directly proportional to the number of records. We have verified the linear scalability empirically, as well. Furthermore, all but one anonymization option adds less than a 25% increase—most actually less than 5%—to the running time as compared to simply copying the file with no anonymization. This means file I/O dominates CANINE's running time.

Prefix-preserving anonymization is the one exception because of the complex computations involved. Each record contains 2 IP addresses, and the prefix-preserving anonymization makes 32 calls to an AES function for each IP address. These 64 calls to AES-128 are equivalent to encrypting a 1 KB block of data. So for this method, anonymization is about 22 times as slow as using no anonymization. However, this is still not slow in absolute terms. On a 2.4 GHz Xeon with 1 GB of memory, CANINE can perform prefix-preserving IP address anonymization at a rate of 1.66 MB/min. Just using an older Dell 650 N workstation such as this, we could handle all of traffic for the thousands of cluster nodes and workstations at the NCSA in real-time (We generate about 2 GB of NetFlows a day). Thus a modern workstation could anonymize logs in real-time for a very large organization even using the slowest anonymization algorithm. The next slowest algorithm is 17.5 times as fast.

V. Conclusions and Future Work

We have argued that log anonymization is essential to the advancement of network forensics. This follows since anonymization is necessary to promote widespread sharing of sensitive log data, and log sharing is necessary for both testing forensic tools and creating educational materials for instructors of digital forensics.

Tools for log anonymization are currently scarce and the field immature. Most only handle one type of log and often just the IP address field within that log. Anonymization solutions should support multiple levels of anonymization to trade-off between security (of the anonymization scheme) and utility (of the anonymized log data). Multiple levels are necessary to customize anonymization solutions to best fit the multiple levels of trust shared between those who would share their logs and those with whom they would share.

Our tool CANINE (Converter and ANonymizer for Investigating Netflow Events) is a new anonymization tool that is unique in 2 ways. First, it is the only anonymization tool we are aware of that handles NetFlows, an important network log collected by most routers. Second, our tool is the first to provide so many options for anonymizing many fields in many different ways. This provides the user greater control and customization of the anonymization process, providing multiple levels of anonymization. In this paper we have discussed the different anonymization options of CANINE, their implementations and trade-offs in depth. Further, we have shown that CANINE is extremely scalable, linear with respect to the size of the log for all anonymization options. Furthermore, this overhead is often little more than simply copying the log without anonymizing.

There is a lot of work to still be done in the field of log anonymization. Specifically for CANINE, we could add support for a streaming mode that would listen for NetFlows on a socket and write them to disk already anonymized. We could also add support for a few more formats such as NFDump and Cisco 9 which are both less commonly used than Cisco 5 and 7. Currently, anonymization tools only handle a single type of log. We are building a new anonymizer for syslogs that separates the anonymization engine from the log parsing routines. In this way, anonymization will just be based upon the types of fields in the log and not the log itself. It will then be easier

to add support for new log types by simply adding new parsing routines. Changes will only be made to the anonymization engine infrequently since there are many times fewer types of fields than logs themselves.

Lastly, this field of log anonymization needs standards to define multiple levels of anonymization more specifically. On the road to developing standards, metrics should be created to measure the utility of an anonymized log as well as the strength of an anonymization scheme. The first metric will likely depend upon the kinds of attacks detectable from a specific log type and the latter will require an analysis of the types of attacks against anonymization schemes. We have begun this work, and when completed, we will have a more quantitative way to create multiple levels of anonymization. Currently, the task of choosing an appropriate level of anonymization—which is essentially a set of algorithms and the corresponding fields to which they are applied—is somewhat manual and as much an art as a science. Quantitative metrics will make it more of a science.

VI. Acknowledgments

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).

References

[1] J. Biskup and U. Flegel, "On Pseudonymization of Audit Data for Intrusion Detection", USENIX Workshop on Design Issues in Anonymity and Unobservability, 2000.

[2] J. Biskup and U. Flegel, "Transaction-Based Pseudonyms in Audit Data for Privacy Respecting Intrusion Detection", Recent Advances in Intrusion Detection (RAID), 2000.

[3] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", Communications of the ACM, Vol. 24 No. 2, 1981, pp. 84-88.

[4] U. Flegel, "Pseudonymizing UNIX Log Files", Infrastructure Security Conference, 2002.

[5] Department of Homeland Security, "Information Sharing & Analysis Centers", Jun. 2005; http://www.dhs.gov/dhspublic/display?theme=73&content=1375

[6] ISC – Internet Storm Center, Jun. 2005; http://isc.incidents.org

[7] Iain Thomson, "IT Industry's 12-point Cyber-security Plan", Dec. 2004; http://www.vnunet.com/news/1160087

[8] P. Lincoln, P. Porras and V. Shmatikov, "Privacy-Preserving Sharing and Correlation of Security Alerts", USENIX Security Symposium, San Diego, CA, 2004.

[9] E. Lundin and E. Jonsson, "Privacy vs Intrusion Detection Analysis", Recent advances in Intrusion Detection (RAID), 1999.

[10] The White House, "The National Strategy to Secure Cyberspace", Jun. 2005; http://www.securecyberspace.gov

[11] R. Pang and V. Paxson, "A High-Level Programming Environment for Packet Trace Anonymization and Transformation", ACM SIGSOMM, 2003.

[12] Adam Slagell, Jun Wang, and William Yurcik "Network Log Anonymization: Application of Crypto-PAn to Cisco Netflows", Workshop on Secure Knowledge Management (SKM), 2004.

[13] M. Sobirey, S. Fischer-Hubner and K. Rannenburg, "Pseudonymous Audit for Privacy Enhanced Intrusion Detection", IFIP TC11 13^{th} International Conference on Information Security (SEC), 1997.

[14] B. Waters, D. Balfanz, G. Durfee and D.K. Smetters, "Building an Encrypted and Searchable Audit Log", Internet Society Network Distributed Systems Symposium (NDSS), 2004.

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

[16] 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", IEEE International Conference on Network Protocols (ICNP), 2002.