K.G. Saur Verlag, München, 2008

Th. Gamer, Chr. Mayer, and M. Schöller

PktAnon – A Generic Framework for Profile-based Traffic Anonymization

Thomas Gamer studied Computer Sciences at the Univerity of Karlsruhe and received its diploma degree in 2005. Since then, he has been working as scientific assistant at the Institute of Telematics of the University of Karlsruhe. His main research interests include anomaly-based attack detection, network security, and WLAN meshes.

Christoph P. Mayer graduated from the University of Karlsruhe in 2007 with a diploma in computer science. In 2008 he joined the Institute of Telematics at the University of Karlsruhe as scientific staff. He is working on security and anonymity for future networks in the BW-FIT project SpoVNet.

Marcus Schöller has studied computer science at University of Erlangen-Nürnberg, University of Karlsruhe (TH), and University of Uppsalla, Sweden. In 2006 he received his doctoral degree from University of Karlsruhe (TH) for his studies on robustness of programmable networks. Thereafter, he joined the computing department of Lancaster University, UK, for his postdoctoral year investigating fundamental principles of resilient networking. In June 2006 Marcus joined the Next Generation Network Group at NEC Network Laboratories Heidelberg as a research scientist. His research covers areas about autonomic and resilient networking, security, and monitoring.

ABSTRACT

Computer network researchers, system engineers and network operators have an increasing need for network traces. These are necessary to build and evaluate communication systems. This ranges from developing intrusion detection systems over evaluating network protocols or system design decisions, up to education in network security. Unfortunately, availability of real-world traces is very scarce, mainly due to privacy and security concerns. Making recorded data anonymous helps to mitigate this problem. Available anonymization systems, however, do not provide sufficient flexibility, extensibility or ease of use. Therefore, we developed a generic framework for traffic anonymization that can easily be configured by anonymization profiles. Such profiles ensure an easy adaptation of the information actually being made anonymous to different environments or local legislation. Furthermore, our framework supports flexible application of arbitrary anonymization primitives to every protocol field. Due to its extensibility our framework provides an easy incorporation of new anonymity-enhancing techniques, too. Additionally, it prevents accidental disclosure of private data by applying a technique called defensive transformation. Finally, it can be used for online as well as offline anonymization of network traffic.

I INTRODUCTION

Anonymization of network traffic has been a challenging Problem for several years. Despite the work that has been done from that time on and the huge benefit that results from real-world network traces, sharing of recorded traffic today is still performed only in a very limited domain.

Network traces are necessary in very different fields of computer networking in order to evaluate new networking mechanisms, software systems or system design decisions. During deployment of a new network decisions about required services and their location have to be made. Therefore, assumptions are made on the number of users, on the usage of services, on traffic distribution, and many more network characteristics. Afterwards, the network behavior in different situations has to be evaluated with real-world network traffic. The development of network protocols or the design of server systems also relies on decisions about system parameters like buffer sizes or applied algorithms. These decisions, which are based on information that can be inferred from real-world traffic, highly affect system performance and robustness.

Manufacturers of signature-based attack detection systems need large amounts of real-world traffic, too, e.g. to create signatures of malicious packets. Additionally, researchers that work on anomaly-based detection systems need real-world traffic to evaluate effectiveness of their systems. If such systems are deployed in real networks their success heavily depends on the quality of the evaluation and therefore, on real-world network traffic.

However, in most cases simulated network traffic must be used which makes certain assumptions about traffic characteristics like burstiness or address distributions due to the lack of real-world traces. On the one hand, these simplifications ensure that specific characteristics of the developed system or protocol can

10.1515/piko.2008.0016

be evaluated in detail without side effects. On the other hand the simplifications may lead to inaccurate or even incorrect results compared to an evaluation with real-world network traffic. Real-world traffic shows some characteristics that cannot precisely be replicated by self-generated traffic. Examples encompass abnormal behavior due to misconfigured nodes or the occurrence of non protocol-conform packets due to faulty interfaces or links. In summary, an evaluation of newly designed network systems and protocols with real-world network traffic does not only show how a system works under real conditions but also how it reacts under adverse conditions. Thus, reliable results can only be achieved by using real-world network traffic.

A Requirements

In order to get real-world traffic for evaluations and analysis network traffic of routers or hosts has to be recorded. But there are some requirements for recording traffic that is meant to be made available to others, e.g. to another department of a company or to other users of the Internet.

Recorded network traffic currently is scarcely publicly available mainly due to privacy concerns. Thus, the first requirement for recording traffic is to *protect the users of the network.* The anonymization that is necessary to fulfill this requirement usually is regulated by local legislation. In most countries all data which allows a mapping of recorded traffic to the real identity of the person that sent the data has to be made anonymous. This includes, for example, an anonymization of IP addresses and payload data since such data often contains private information. Furthermore, some data protection laws state that personal data may be recorded only if really necessary: principle of avoidance and thriftiness. If traffic, however, is recorded it has to be made anonymous *as fast as possible* [1]. In our opinion, this requires online anonymization in most cases, i.e., traffic has to be made anonymous before storing it on some medium.

Another aspect of privacy concerns which is often ignored is the *protection of companies and their networks.* This protection aims at making anonymous all data that might be useful for an attacker to compromise the network or to launch a highly sophisticated attack. Internal network structures, available services, and already compromised machines are just a part of the information an attacker can gain from network traces that are only made anonymous in regard to protection of users. Therefore, the second requirement is beyond legal requirements and includes e.g. an anonymization of port numbers or header fields of the application protocol. Such an anonymization may also be performed offline, i.e., before making previously recorded traffic available to others.

Additionally, companies may want to outsource talks like accounting or security analysis. Anonymization then needs special properties, e.g. certain internal structural elements like network prefixes or broadcast addresses must be preserved.

In order to address these requirements a framework for traffic anonymization must fulfill the following characteristics:

Flexibility means the possibility to apply the anonymization framework in different technical environments and with varying objectives. Especially autonomic networks [2] will have varying network characteristics and thus, require a high flexibility. Therefore, it must be possible to make any data field anonymous with an arbitrary anonymization primitive.

Extensibility to easily add new network protocols and new anonymization primitives. A huge range of different network protocols exists and each of them might be of interest to researchers. Thus, it is highly important to allow arbitrary protocols to be easily added to the anonymization framework. Therefore, the framework must be designed to be very generic. It must have a common protocol interface and support arbitrary protocol encapsulations like IP-in-IP or IP-in-ICMP transparently. Additionally, an anonymization framework must allow arbitrary anonymization primitives to be included into the framework easily. This ensures adaptability to different as well as future anonymization needs and furthermore, gives researchers the possibility to easily integrate newly developed primitives into the framework and e.g. to evaluate them in a real system with currently known attacks on anonymization [3].

Configurability for each user that records traffic using the anonymization framework to define his own *anonymization profile*, i.e. his own mapping between protocol attributes and anonymization primitives, according to his needs. Thus, profiles implement a specific *anonymization policy* that respects the trust relationship between the party that records network traffic and the party the recorded traffic is given to. In order to ensure usability policies should not be hard-coded but must be configurable outside of the program using e.g. a simple configuration file which can be edited easily. It is, however, beyond the scope of this paper to answer the question how to choose the right anonymization primitives and protocol attributes that should be made anonymous.

B Related Work

Tcpdpriv [4], the first anonymization tool for network traffic has been published in 1996. It provides anonymization for a few data fields like IP addresses, port numbers and payload data by applying some predefined anonymization primitives. A small degree of configurability is achieved by command line parameters. Extensibility and flexible mapping of arbitrary data fields to arbitrary anonymization primitives, however, is not available. Pang et al. [5] extended the policy-based intrusion detection system Bro to perform network trace anonymization. The focus of their work is on application level protocols like HTTP or FTP. The anonymization framework FLAIM [6] provides extensibility but lacks some flexibility as it can only map certain anonymization primitives to a protocol attribute, a hash function or HMAC anonymization, for example, can not be applied to IP addresses. Furthermore it does neither support arbitrary protocol encapsulations like IP-in-IP [7] nor checksum validation and recalculation after having made data anonymous.

Koukis et al. [8] propose a generic application programming interface (API) to perform network trace anonymization. The anonymization policy a user desires is created by using a series of API function calls which may be difficult especially for users without a technical background. Furthermore, it is not possible to quickly change or adapt the anonymization profile without rebuilding the whole project. *Tcpmkpub* [9] is considered the current state-of-the-art tool for making network traces anonymous. It is extensible and configurable by providing external configuration of protocol formats and mapping between data fields and anonymization primitives. Flexibility, however, is decreased by the fact that no generic anonymization primitives can be applied to different protocol attributes, e.g. a blackmarker must be implemented separately for each attribute it should be applied to. Furthermore, online anonymization is not possible since *Tcp-*

mkpub needs multiple passes over the original network trace for anonymization of data.

In addition to anonymization tools and frameworks several contributions have been made focusing only on certain anonymization primitives. Biskup et al. [10] and Verlier et al. [11], for example, designed pseudonymization methods that can be used in conjunction with intrusion detection systems. Primitives for a prefix-preserving IP address anonymization have been developed by Xu, Fan et al. [12], Harvan and Schonwalder [13], and Ramaswamy and Wolf [14]. Anonymization techniques regarding URLs and filenames have been evaluated by Kuenning and Miller [15]. A different anonymization approach by hiding the content of packets in a larger, scrambled content has been suggested by Ponce et al. [16]. All these primitives can be easily integrated into our anonymization framework.

Simultaneously to the development of new anonymization primitives attacks on known primitives or anonymous traces have been performed. Brekne et al. [17] evaluated attacks on prefix-preserving IP address anonymization as well as pseudonymization schemes. Kohno et al. [18] used the TCP timestamp option to fingerprint remote hosts in order to show that this option is – in contrast to former believe – highly sensitive. Coull et al. [3] used statistical methods in conjunction with publicly available information to infer sensitive information like server identities from a set of published traces that were made anonymous by *Tcpmkpub*. Allman and Paxson [19] discuss higher level issues on network data sharing and give a framework of guidelines for sharing and usage of measurement data.

II FRAMEWORK

Existing tools and frameworks that are able to record and make network traces anonymous are often applicable in certain situations but lack useful features in other ones. Therefore, we built a generic anonymization framework that overcomes these limitations by providing flexibility, extensibility, and configurability.

In section II-A we explain the design considerations that form the basis of our framework. Then, in section II-B implementation details are given before describing the integration of anonymization profiles in section II-C. Finally, section III details on an evaluation of our anonymization framework.

A Design considerations

Several design considerations have to be answered when building a network trace anonymization system. We will explain these considerations here to make clear why we designed our system called *PktAnon* the way we did.

1) Input and output handling: In order to achieve full flexibility we decided to clearly separate I/O handling from packet parsing using a minimal interface. This ensures that different formats can be applied, i.e., only packet data which is independent of the actual format is delivered to packet parsing of the framework. By encapsulating the actual I/O format online as well as offline anonymization is enabled. We consider online anonymization the best and most secure choice for usage in network trace anonymization since no original data is stored on local discs.

2) Packet parsing: In order to access the data fields which have to be made anonymous and to write back the transformed data the syntax of parsed packets must be implemented into protocol classes. This can be done either within the anonymization system itself or externally configured.

We decided not to use external configuration but to build the protocol definitions into our anonymization framework. This allows the complete protocol specific structures and semantics to be encapsulated into a protocol class and thereby, leads to a clear object-oriented design. External protocol definitions [9], on the other hand, use a generic stream reader and protocol specific configurations to parse packets. But these definitions are not flexible enough to capture the versatility of protocols. By using an external configuration it is, for example, hard to express that calculation of the TCP checksum is done using a pseudo header which includes the lower layer IP address.

3) Building the protocol chain: As today's network packets are often arbitrarily encapsulated we chose a loose coupling of network protocols which we call a *protocol chain.* Protocols of a packet are sequentially read and corresponding protocol classes are instantiated. These objects then are linked forming the protocol chain (see fig. 1). In this way, encapsulations like IP-in-IP or IP-in-ICMP can be handled transparently and thus, extensibility regarding support of arbitrary protocol encapsulations is provided by our framework.

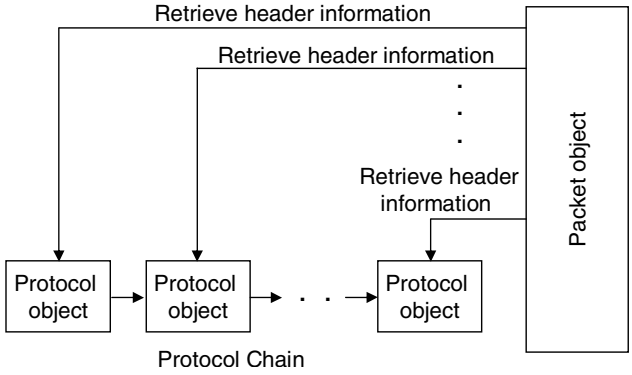

Fig. 1 Building the protocol chain

4) Protocol chain transformation: Having created the protocol chain any protocol field can be made anonymous. Therefore, a configuration is created that defines for each supported protocol which anonymization primitives are applied to the data fields of the protocol (see section II-C for further details). Flexibility of our framework is ensured by allowing an arbitrary mapping of anonymization primitives to protocol attributes.

In order to prevent accidental transfer of private data from the original packet to the transformed packet we use a transformation model we call *defensive transformation.* Instead of changing the value of a protocol attribute in the original protocol chain we perform a different approach: during the anonymization process a protocol chain duplicate is created. All fields are initially empty and must be filled by a transformation from the original packet chain (see fig. 2). Thus, no sensitive data can remain in the anonymous protocol chain by mistake. This means that protocol fields that should keep their original values must be explicitly copied into the protocol chain duplicate. Therefore, an additional anonymization primitive called *AnonIdentity* is defined for this task. We believe that by applying defensive transformation a higher level of security can be reached since original data is only transferred to the new chain if explicitly specified.

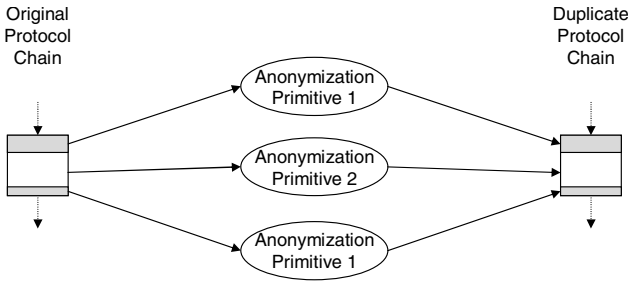

Fig. 2 Protocol chain transformation

Besides applying a single anonymization primitive to the value of a protocol field our framework additionally supports chaining of anonymization primitives. In this case the first anonymization primitive is called to transform the provided buffer during the transformation process. Depending on the result of this anonymization primitive a further primitive can be called. By allowing usage of consecutive anonymization primitives we provide special handling e.g. for protocol attributes like broadcast MAC addresses in case a user wants to preserve them. In this way, our framework provides a flexible method for handling special protocol attributes that is much easier to use than hard-coded exception mechanisms.

B Implementation

We will now describe theiImplementation of *PktAnon* in detail to make clear how the design considerations named in section II-A are realized. *PktAnon* has been written in C++ and tested with Microsoft Visual C++ 2005 under Windows XP and GNU gcc 4.0.2 under Linux 2.6.13.

1) Protocol classes: pcap frames are sequentially read from the input source one at a time. The packet buffer portion of a frame is then given to the `Packet` class that starts the main parsing routine. In order to enable the `Packet` class to build the protocol chain all *protocol classes* must implement a basic interface that allows the parsing process for each protocol to be started from the `Packet` class. After parsing a protocol all protocol classes return which is the subsequent upper layer protocol. According to this value the `Packet` class creates a suitable protocol class object and calls the parsing routine. This way the protocol chain is built iteratively until the end of a packet is reached.

So far *PktAnon* implements the following protocol classes: Ethernet, ARP, IPv4, IPv6, ICMPv4, UDP, TCP, and PayloadPacket. A protocol class must implement parsing and assembling functionality and provide access to protocol attributes in a convenient way. Furthermore the protocol class is responsible to guarantee a well formed binary data after assembling. This includes adjusting length fields and recalculating checksums. If a protocol does not know its next upper layer protocol it tells the framework to use the *PayloadPacket* protocol class which simply encapsulates a data buffer containing arbitrary data.

2) Anonymization primitive: To provide an arbitrary mapping of protocol attributes to anonymization primitives we define a minimal interface for anonymization primitives.

During the transformation process of the protocol chain the assigned anonymization primitive is called automatically from within the framework for each protocol attribute. The actual attribute is handled transparently as a generic data buffer with a specific length. An anonymization primitive returns two values:

– A boolean value whether the anonymization process continues or is aborted in case multiple anonymization primitives should be applied consecutively
– The new length of the data buffer in case the buffer length changed

Table 1 Overview of anonymization primitives and their parameters

Name	Parameters	Description
AnonBroadcast-Handler	–	Preserve broadcast MAC addresses.
AnonConstOverwrite	byte value	Overwrite every byte with provided value.
AnonCryptoPan	key for Rijndael	Prefix-preserving anonymization [20].
AnonHashSha1	–	Hash complete buffer with SHA1.
AnonHashHmacSha1	key for HMAC	Hash complete buffer with HMAC SHA1.
AnonIdentity	–	Do not change buffer.
AnonRandomize	–	Overwrite each byte with a random value.
AnonShorten	new length	Cut buffer to given length.

Table 1 shows some exemplary anonymization primitives that currently are implemented into *PktAnon*. The AnonShorten primitive, for example, can be used to shorten PayloadPacket buffers and option fields like in IP and TCP. This way the protocol chain can be kept up to the point of the last known protocol and payload data can be shortened to a new length of 0 if storage space optimization shall be achieved. The AnonBroadcastHandler primitive currently is the only conditional anonymization primitive that can be used to apply chaining of anonymization primitives.

C Anonymization profiles

As packet traces are recorded by one party and used by another one the trust relationship between the two must be reflected by the anonymization policies. It is easily imaginable that lots of different trust relationships exist and therefore, a profile-based anonymization framework must allow to implement all possible policy sets in order to ensure full configurability.

```
<submodule name=TcpPacket>
 <item name=TcpSourceport anon=AnonHashSha1/>
 <item name=TcpDestport   anon=AnonHashSha1/>
 <item name=TcpSegnum     anon=AnonIdentity/>
 <item name=TcpAcknum      anon=AnonIdentity/>
 <item name=TcpFlags      anon=AnonIdentity/>
 <item name=TcpWindowsize  anon=AnonIdentity/>
 <item name=TcpUrgent     anon=AnonConstOverwrite val=0x00/>
 <item name=TcpOpt        anon=AnonShorten newlen=0/>
</submodule>
```

Fig. 3 Extract from an anonymization profile

PktAnon uses XML-based anonymization profiles to configure the mapping of anonymization primitives to protocol attributes. Furthermore, additional parameters of anonymization primitives (see table I) are configured in these profiles, too. Fig. 3 shows an exemplary XML configuration for the TCP protocol.

Additionally, general parameters are configured by such XML-based profiles, e.g. input and output as well as flags that control tasks like the inbuilt runtime measurement engine or ASCII-based debug output of original and anonymous data.

At startup, *PktAnon* reads the given XML profile, creates and configures anonymization primitives and then statically assigns them to the according protocol attributes. Each protocol attribute that is implemented in *PktAnon* must have an anonymization primitive assigned in the XML profile – otherwise an error is reported. Protocol attributes that are not to be made anonymous must be assigned the `AnonIdentity` primitive. Additionally, further checks, e.g. if parameters are within valid boundaries, are performed during startup in order to prevent configuration errors.

III EVALUATION

We have shown in section II-A.1 that our system supports several input/output sources and thus, enables offline as well as online anonymization. Especially for online anonymization where traffic from e.g. *tcpdump* is directly piped into *PktAnon*, processing speed is very important. A processing that is too slow will make the packet capture entity drive into dropping packets as they cannot be delivered through standard I/O.

Our evaluation system has the following configuration: GNU/Linux 2.6.13 i386, Intel(R) Pentium(R) 4 CPU 2.80 GHz with 2 GB RAM. In order to determine the maximum processing speed of our anonymization framework we use offline anonymization. Therefore, a trace file is used that has been recorded on a gateway router of a large stub network. The trace file contains about 8.2M full-length packets which add up to a file size of 4.7 GByte including *pcap* headers.

A PktAnon *runtime*

First we evaluate the performance of each stage of our framework successively. The profile we use for measurement in this evaluation is an identity profile, i.e., the `AnonIdentity` primitive is attached to each protocol attribute. Runtime of each single anonymization primitive will be evaluated separately in section III-B. The different stages of *PktAnon* are measured in the following sequential order:

- **File input processing** *PktAnon* reads packets one at a time from the source file. With this test we can measure how file input affects processing speed.
- **Packet parsing and building the protocol chain** This test focuses on parsing of data read from the source file and on building of original protocol chains which is done after reading a packet.
- **Defensive protocol chain transformation** This test measures the time *PktAnon* needs to create the duplicate protocol chain and apply anonymization. As performance of the anonymization primitives is evaluated later we use the identity profile for transformation. After anonymization the new protocol chain is converted into the appropriate input/output format.
- **File output** The last test evaluates the performance of the file output operation.

Optional tasks that can be additionally applied are **checksum recalculation** and **checksum validation**. Checksum recalculation is done after protocol chain transformation if required. Additionally, *PktAnon* allows checksums to be validated. This is useful to keep bad checksums in the anonymous data for protocols that originally had bad checksums.

Table II(a) shows the average speed of each of the serialized tasks. It can be seen that file output takes up a lot of processing time and that normal protocol handling and the anonymization process are fast compared to I/O. The optional tasks checksum recalculation and validation were measured after evaluating the serialized tasks. The result of this measurement – about 1 s additional time per task – shows that these optional tasks are negligible.

TABLE II Average Throughput of PktAnon

(a) Different anonymization tasks

Sequential tasks	Throughput [Mbit/s]	Duration [s]
File input	406.8	95.1
Packet parsing	404.2	95.6
Chain transformation	337.3	113.5
File output	150.6	257.2
tcpdpriv (no file output)	392.4	99.8

(b) Different anonymization primitives

Anonymization primitive	Throughput [Mbit/s]
AnonIdentity	337.3
AnonConstOverwrite	309.8
AnonHashSha1	39.0
AnonHashHmacSha1	17.9
AnonRandomize	57.1
Mixed primitives	95.6

Additionally, we compared *PktAnon* with *tcpdpriv* [4]. The last row of table II(a) shows the duration and throughput of *tcpdpriv* if the same input file is used. In order to enable a correct comparison between both tools, *tcpdpriv* is configured not to make IP addresses and port numbers anonymous. Payload, however, is always deleted by *tcpdpriv*, i.e., packets are written to an output file without payload data. Therefore, we only measured the time *tcpdpriv* took to read the input file and to perform the identity anonymization. Moreover, checksum recalculation is done during the measured time. A comparison between both tools – *tcpdpriv* took about 100 s, *PktAnon* about 114 s – shows that our anonymization framework took only a little more time for offline anonymization than *tcpdpriv* but offers a more secure anonymization by defensive transformation. Additionally, higher configurability and usability are provided by usage of anonymization profiles as well as higher flexibility by allowing an anonymization of each protocol attribute. Finally, extensibility ensures anonymization of arbitrary protocols, application of arbitrary anonymization primitives, and easy incorporation of to-be developed anonymization primitives and protocols.

Furthermore, we did an evaluation of online anonymization based on the identity profile. Therefore, we generated CBR network traffic. A throughput of up to 120 Mbit/s can be achieved without packets being dropped by our framework. Real network traffic, however, has no constant bitrate but, on the contrary, shows self-similar behavior [21]. This leads to the situation that no universally valid statement about achievable processing speed is possible but throughput and number of dropped pack-

ets heavily depends on the characteristics of the recorded network traffic.

B Anonymization primitive runtime

During the previous evaluation we only used the anonymization primitive AnonIdentity for anonymization. We have seen that an average throughput of about 337 Mbit/s can be achieved using this primitive if file output is not regarded. Table II(b) lists the average throughput – not regarding file output – if each protocol field of a packet is made anonymous with the measured anonymization primitive, i.e., the primitive is applied to every protocol field of a packet. AnonShorten cannot be applied to each protocol field and thus, was not measured but this primitive has a complexity of $O(1)$. AnonCryptoPan can only be applied to IPv4 addresses and has been evaluated in [22]. The last row of table 11(b) additionally shows the resulting throughput if an anonymization profile is applied that, in our opinion, protects users and the network. In this case different primitives are used depending on the protocol field.

The first two primitives have a complexity of $O(N)$ to make N bytes anonymous and do not reduce the average throughput much. Primitives that must calculate a hash value or create a random number for each protocol attribute, however, significantly reduce the average throughput. Especially AnonHashHmacSha1 which has to calculate a cryptographic hash value is expensive in comparison to other primitives if applied to each field. The value in the last row, however, shows that if a more realistic anonymization profile is applied an average throughput of about 100 Mbit/s can be achieved with offline anonymization.

IV SUMMARY AND OUTLOOK

The importance of recorded network traces increases with the increasing complexity of today's networks. Researchers, system engineers, and network operators are in need of recorded traffic for their daily work. On the other hand, privacy and security issues as well as local legislation limit the way network data may be recorded. Specialized solutions have been proposed for many different application domains but no universal framework is available today which can be adapted to different scenarios, requirements, and trust relations. Our work depicts such a framework that can be configured easily using anonymization profiles. It prevents accidental information leakage by defensive transformation, and supports chaining of anonymization primitives. Furthermore, it is extensible and allows arbitrary anonymization of every known protocol attribute. Compared to existing solutions the performance of our tool looks promising. *PktAnon* software is publicly available at [23].

Future work has to be done on defining anonymization profiles that meet the basic legal issues. This must be achieved in collaboration with lawyers specialized in this area. *PktAnon* enables an easy creation and deployment of these profiles. Additionally, further work has to be done to improve processing speed of our framework, e.g. by applying parallel processing or hardware-based cryptographic acceleration. Finally, achievable throughput and packet drop rates of an online anonymization have to be evaluated in more detail using real network traffic with real-world characteristics like self-similarity.

REFERENCES

[1] Haibl, Dressler: Anonymization of measurement and monitoring data: Requirements and solutions. In: Praxis der Informationsverarbeitung und Kommunikation (PIK) 29 (4) (2006) 208-213.

[2] Schmid, Sifalakis, Hutchison: Towards autonomic networks. In: Proceedings of 3rd International Annual Conference on Autonomic Networking, Autonomic Communication Workshop (IFIP). Lecture Notes in Computer Science, Springer Verlag, Heidelberg (2006) 1-11.

[3] Coull, Wright, Monrose, Collins, Reiter: Playing devil's advocate: inferring sensitive information from anonymized network traces. In: Proceedings of the ISOC Network and Distributed Systems Symposium. (2007) 35-47.

[4] Minshall: tcpdpriv (1996) http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.

[5] Pang, Paxson: A high-level programming environment for packet trace anonymization and transformation. In: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, ACM Press (2003) 339-351.

[6] Slagell, Lakkaraju, Luo: Flaim: A multi-level anonymization framework for computer and network logs. In: Proceedings of 20th USENIX Large Installation System Administration Conference (LISA '06). (2006) 101-115.

[7] Perkins: IP Encapsulation within IP. RFC 2003, IETF (1996).

[8] Koukis, Antonatos, Antoniades, Markatos, Trimintzios: A generic anonymization framework for network traffic. In: Proceedings of IEEE International Conference on Communications (ICC), IEEE (2006) 2302-2309.

[9] Pang, Allman, Paxson, Lee: The devil and packet trace anonymization. In: ACM SIGCOMM Computer Communication Review. (2006) 29-38

[10] Biskup, Flegel: Transaction-based pseudonyms in audit data for privacy respecting intrusion detection. In: Third International Symposium on Recent Advances in Intrusion Detection (RAID). Lecture Notes in Computer Science, Springer Verlag, Heidelberg (2000) 28-48.

[11] Verlier, Brekne, Eres: Non-expanding transaction specific pseudonymization for ip traffic monitoring. In: Proceedings of Cryptology and network security (GANS), GANS (2005) 261-273.

[12] Xu, Fan, Ammar, Moon: Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In: Proceedings of IEEE International Conference on Network Protocols, IEEE (2002) 280-289.

[13] Harvan, Schonwalder: Prefix- and lexicographical-order-preserving ip address anonymization. In: Proceedings of Network Operations and Management Symposium (NOMS). (2006) 519-526.

[14] Ramaswamy, Wolf: High-speed prefix-preserving ip address anonymization for passive measurement systems. IEEE/ACM Transactions on Networking 15 (1) (2007) 26-39.

[15] Kuenning, Miller: Anonymization techniques for urls and filenames. Technical report, University of California, Santa Cruz (2003).

[16] Ponce, Loebl, Kencl: Packet content anonymization by hiding words. In: Demo at IEEE Infocom. (2006).

[17] Brekne, Arnes: Circumventing ip-address pseudonymization. In: Proceedings of the Third IASTED International Conference on Communications and Computer Networks, IASTED/ACTA Press (2005) 43-48.

[18] Kohno, Broido, Claffy: Remote physical device fingerprinting. IEEE Transactions on Dependable and Secure Computing 2 (2) (2005) 93-108.

[19] Alllman, Paxson: Issues and etiquette concerning use of shared measurement data. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, ACM New York, NY, USA (2007) 135-140.

[20] Fan, Xu, Ammar, Moon: Crypto-pan – cryptography-based prefix-preserving anonymization. http://www.cc.gatech.edu/Computing/Networking/Projects/cryptopan/ (2004).

[21] Park, Willinger: Self-similar network traffic: An overview. In: Self-Similar Network Traffic and Performance Evaluation, Wiley Interscience (1999) 3-49.

[22] Xu, Fan, Ammar, Moon: On the design and performance of prefix-preserving ip traffic trace anonymization. In: Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW). (2001) 263-266.

[23] Gamer, Mayer, Schöller: Pktanon project (2007). http://tm.uka.de/~mayer/pktanon.