

Anonymization of real data for IDS benchmarking

Vidar Evenrud Seeberg



Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2006

Institutt for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Abstract

Most IDS evaluation approaches use simulated network traffic as base for the test data sets used in the evaluation. Simulated network traffic lacks the diversities characteristic to a real world network. These diversities may be caused by non-standard implementations of protocols or abnormal protocol behavior, like unfinished threeway TCP handshakes and teardowns.

For realistic IDS evaluations, there is a need for test data sets based on real recorded network traffic. Such data sets must also be distributable since a valid test should be possible to reproduce by other evaluators. Due to legal concerns test data sets based on real recorded traffic must be anonymized.

This thesis presents a methodology for anonymization of real network data. The methodology focuses on information at the application layer, and HTTP/1.1 in particular. A prototype, called Anonymator, is implemented based on the methodology. A data set anonymized using such a methodology can be used in IDS evaluations, providing more realistic evaluations. It can also be distributed since identifying information is anonymized. This way evaluations can be validated by third parties.

The methodology and prototype are tested thoroughly through experiments using a data set consisting of HTTP traffic mixed with attacks. The prototype implements different anonymization strengths that can be chosen by the operator. The experiments show the differences between the anonymization schemes. The differences are carefully explained. Results show that the two strongest anonymization schemes give good level of anonymity without losing too much realism.

Contents

Abstract	ii
Contents	iii
List of Figures	v
List of Tables	v
List of Symbols	v
1 Introduction	1
1.1 Topic	1
1.2 Justification, motivation and benefits	1
1.3 Research methodology	2
2 Previous Work	3
2.1 IDS evaluations	3
2.2 Sensitivity, privacy and identity	3
2.3 Known methodologies for anonymizing real network data	4
3 The new anonymization methodology	7
3.1 Methodology introduction	7
3.1.1 Classification scheme	7
3.1.2 How to anonymize	8
3.2 Identifying information	8
3.3 Identifying information in network and transport protocols	9
3.3.1 IPv4 Header	9
3.3.2 TCP Header	12
3.3.3 UDP Header	13
3.4 Identifying information in application headers and payload	13
3.4.1 HTTP 1.1	13
3.5 Correlation of headers and other data	28
3.6 Methodology conclusion	29
4 Experimental Work	31
4.1 Introduction	31
4.2 Prototype	31
4.2.1 Architecture and implementation	32
4.2.2 Pseudocode	33
4.2.3 Using Anonymator	34
4.2.4 Testing Anonymator	35
4.3 Expectations	40
4.4 Infrastructure and resources	44
4.4.1 Preparations	44
4.5 Revised expectations	45
4.6 Experiments	46
4.6.1 Exp. 1: No anonymization	46
4.6.2 Exp. 2: Weak scheme	47

4.6.3	Exp. 3: Strong scheme	47
4.6.4	Exp. 4: Strongest scheme	47
4.6.5	Exp. 5: Customized: Weak URI, no headers anonymized	47
4.6.6	Exp. 6: Customized: Strong URI, no headers anonymized	47
4.6.7	Exp. 7: Customized: Strongest URI, no headers anonymized	48
4.6.8	Exp. 8: Customized: Weak URI, all headers anonymized	48
4.6.9	Exp. 9: Customized: Strong URI, all headers anonymized	48
4.6.10	Exp. 10: Customized: Strongest URI, all headers anonymized	48
4.7	Results	48
4.7.1	Weak related schemes	49
4.7.2	Strong related schemes	50
4.7.3	Strongest related schemes	54
4.8	Conclusions regarding experimental work	59
5	Conclusions	61
5.1	Methodology	61
5.2	Prototype	62
5.3	Experiments	63
5.4	Recommendations	64
6	Further Work	65
6.1	Methodology	65
6.2	Prototype	65
6.3	Experimental	66
	Bibliography	67
A	Classification table	71
B	Number of positives	73
C	Non-disclosure agreement	77

List of Figures

1	IPv4 header[1]	9
2	TCP header[1]	11
3	UDP header[1]	13
4	No anonymization	36
5	Weak anonymization	36
6	Strong anonymization	37
7	Strongest anonymization	37
8	Weak URI + no headers	38
9	Strongest URI + all headers	38
10	Strong URI + some headers	39
11	Experimental infrastructure	44

List of Tables

1	Fields used in Snort and Nessus	41
2	Fields in Snort and Nessus subject to anonymization	42
3	Nessus-generated URI-levels for revised expectations	46
4	Number of positives reported by Snort	49
5	Strong: drop in number of positives	51
6	Strongest: retention of attacks	55
7	Conclusions for experimental work	60
8	Classification	72
9	Number of positives	75

List of symbols

↔	Breaks long URL or filename in running text
#	Number/Count

1 Introduction

1.1 Topic

Benchmarking of Intrusion Detection Systems (IDSs) is necessary for determining how good IDSs are. Current IDS benchmarking efforts are mainly based on simulated test data sets. Traffic generators are used to generate traffic in which attacks are injected. For more realistic benchmarking, real recorded traffic is needed as a base for the test data set used in the evaluation.

There are at least two problems with the generation of test data sets using real prerecorded traffic. First, when recording real data, sensitive data for identifiable subjects may also be recorded. Due to legal concerns, test data sets containing this type of information cannot be distributed. If a data set used in testing cannot be distributed the evaluation will lose its validity. A main property of testing is reproducibility. It should be possible to validate an evaluation through a reevaluation using the same infrastructure, test data and so on. For this to be possible, test data sets *must* be distributable. This property may be accomplished through anonymization of the recorded traffic.

The second problem depends on the first one. When anonymizing a data set, important traffic characteristics may be lost. A main goal is therefore to keep the traffic as realistic as possible. The goals of anonymity and realism are unfortunately conflicting and a trade-off has to be made.

This thesis presents a methodology for anonymization of prerecorded network traffic in order to generate distributable test data sets for IDS benchmarking. The anonymization process will also be considered in regard to the goal of keeping the traffic as realistic as possible. There have been great work done in the area of link, network and transport layer anonymization. For anonymization of application layer protocols, however, much work has to be done. This thesis focuses on application layer anonymization, and HTTP/1.1 in particular. A software prototype, called Anonymator, based on this new anonymization methodology is implemented for anonymizing network data. Experiments using Anonymator have been conducted in order to explore the influence different levels of anonymization have on the recorded traffic.

1.2 Justification, motivation and benefits

When using simulated traffic generated by network traffic generators in IDS benchmarking, many characteristic properties of network behavior will be left out. Also, different networks have different characteristics. To find out which IDS suits a specific network, different IDSs must be tested with a network traffic pattern characteristic to that specific network.

Traffic generators cannot produce traffic that is close enough to the real world for realistic testing. Also, traffic generators are good at producing traffic conforming to protocol specifications. Due to nonstandard implementations of protocols in many applications, and other deviating protocol behaviors (like unfinished TCP handshakes and teardowns), network traffic may contain varying levels of abnormal traffic patterns. For realistic IDS

benchmarking, test data sets must be generated based on real traffic from the network type in question. This thesis will present a methodology and a prototype software able to produce anonymized test data sets for any environment. Network data may be recorded in any type of environment, for later to be anonymized using the system to be developed. This data set can be used in IDS benchmarking. Such a data set may also be distributed since it is anonymized. In this way it may also be possible to build a library of anonymized data sets to be used for different purposes. The methodology and prototype developed in this thesis focus for the current version on HTTP data.

More realistic IDS benchmarking may be helpful for vendors producing IDSs, for customers buying them and for administrators tuning them. Vendors may get better information of what their products are good at and not so good at. This may help vendors improve their products and even enable the vendors to tune their IDSs for specific network types. Customers may better be able to determine which IDS to purchase, and administrators may have more detailed information at hand when tuning the performance of the systems.

1.3 Research methodology

The research for this thesis is of quantitative nature. Literature study has been performed throughout the entire research period. However, literature study has been more emphasized in the first part of the project, which is common for most research processes[2]. The bibliography lists the bibliographic resources relevant for this thesis. Deduction based on premises found in literature and source code are made in order to devise the methodology.

Prototyping has been conducted, implementing the methodology devised. The prototype can be considered a "proof-of-concept" one, demonstrating how a methodology implementation can be made. Generally, a prototype can be further developed into a fully functional product or it can be discarded. If discarded, the knowledge derived from the prototype can be used to further enhance the methodology or to develop other software based on the knowledge.

Finally, experiments for evaluating the methodology and the prototype have been conducted. The goal of the experimental part was to explore what influence the prototype had on a data set regarding retention of attacks.

For a thorough description of the methodologies, see [2] and [3].

2 Previous Work

2.1 IDS evaluations

Most current IDS evaluations use artificially generated network traffic as basis for the test data sets. Examples of such evaluations are [4-10]. [11] criticises The DARPA evaluations [4, 5] among other things for lack of realism in the test data sets. [4, 5] simulated the network at an Air Force military base and used this simulated traffic as background in which attacks were injected. [11] questions the test network being used for generating the background traffic. The main concern was that the test network was not able to produce the diversity and quantity characterizing a military network.

[12] states clearly the need for test data sets based on sanitized network traffic. This will provide better and more reliable and valid results when testing IDSs. However, [12] does not provide any solution or methodology for the topic.

2.2 Sensitivity, privacy and identity

Research regarding anonymization of network traffic has been conducted for some time. However, most of the work has been done for network and transport protocols, like IPv4, TCP and UDP. Research in the area of application protocol anonymization is not so well established.

An important question to elaborate when conducting research in the area of anonymization is what information should be considered subject to anonymization. To answer this question some terms must be defined. When handling the topic of anonymity, and the need for an anonymization methodology, several terms have been used. "Sensitive" and "privacy"[12-15 and others] are some of them. The Norwegian privacy legislation[16, §2] defines "sensitive" as (translated by the author of this thesis):

Sensitive personal information Information regarding:

- a) racial or ethnic heritage, or political, philosophical or religious belief
- b) a person being suspect, accused, indicted or convicted for a legal offence
- c) personal health
- d) sexual preferences
- e) labor union membership

[17] has defined "Privacy":

Privacy is the interest that individuals have in sustaining a 'personal space', free from interference by other people and organisations.

One premise for these definitions is that the person associated with sensitive information and privacy must be positively and unmistakably identified[16, §2, no. 1]. The terms "sensitive" and "privacy" may therefore not be precise enough for defining what to

anonymize. A better term derived from [16, §2, no. 1] may be "identifiable information", a term, which more precisely characterizes information of interest for this thesis. Therefore, when terms like "sensitive" and "private" are used throughout the report, they are meant in the context of sensitive or private information for an identifiable person.

The problem of revealing identity regarding distribution of test data sets is clearly stated in [12, p. 17]. [13, 14, 18] provide good descriptions of what type of information should be subject to anonymization in regard to link, network and transport protocols. Some application protocols are also considered, however not as thoroughly as lower layer protocols. [13] states that information subject to anonymization fall into two categories: identities, including identity of users, hosts and data, and confidential attributes like passwords and specifics of sensitive user activity. [13] and [18] also give examples of how different types of information may be correlated to reveal a subject's identity.

There is, however, lack of a complete and comprehensive methodology for anonymization of real network data. A methodology should, in addition to a comprehensive description of fields in different protocols, contain considerations regarding the fields' significance to intrusion detection. Considerations regarding how to anonymize while retaining as much information as possible should therefore be included.

2.3 Known methodologies for anonymizing real network data

Earlier anonymization work have concentrated on anonymizing packet headers belonging to network and transport protocols. [19] presents a method based on cryptography for IP address anonymization while still preserving a common prefix of the address space. [20] presents a scheme for packet trace anonymization where the results are stored in a compressed format. Most such approaches have completely removed the payload [13, Sec. 1].

More recent approaches have also taken the payload into consideration. [13] describes an anonymizer developed as an extension to the Bro Intrusion Detection System [21]. The paper describes several techniques including

- constant substitution, e.g. any password may be substituted with the string `<password>`.
- sequential numbering, e.g. file names may be substituted, like `<file1>`, `<file2>` and so on.
- hashing, i.e. payload is replaced with its HMAC-MD5 hash value.
- prefix-preserving mapping, e.g. the first part of IP-addresses or directory components of file names are hashed, indicating common values.
- adding random noise to numeric values.

Another interesting approach is the *Network Dump* data *Displayer* and *Editor* (NetDude) [22]. NetDude is a framework for packet trace manipulation. The implementation of NetDude described in [23] includes an API¹ the author of NetDude claims could be used to develop an anonymization plugin to NetDude.

One of the main goals of this thesis is to describe how identifying information can be removed from recorded traffic in a way that

¹Application Programming Interface

1. no information needed for intrusion detection is removed and
2. assurance is attained that no private and sensitive information remain.

There will always be a possibility that information needed for intrusion detection is removed. This is especially true regarding unknown attacks. Anonymization may therefore make detection of such attacks even more difficult. [13] and [18] have made significant contributions to this topic in regard to known attacks.

Techniques used in data mining may also be used for anonymization. [24] gives a good overview over terminology in this field. [15] mentions several techniques for privacy preserving data mining, including heuristic-based, cryptography-based and reconstruction-based techniques. Heuristic-based techniques may use perturbation (adding noise) or blocking (substitution). Cryptographic techniques are mostly used in distributed data mining as techniques to partition data. Reconstruction-based techniques use reconstruction of objects (e.g. aggregation) in the data mining approach.

A problem regarding anonymization is to what extent assurance that no sensitive information remains in the data set can be made. It is not possible to be completely sure that all identifying information is anonymized without doing manual inspection. [13] approaches this using a fail-safe filter-in method where everything that need to be in the clear is explicitly stated and everything else is anonymized. This thesis' anonymization methodology will try to improve existing methodologies in such a way that manual inspection is kept at minimum.

3 The new anonymization methodology

3.1 Methodology introduction

The new methodology has to handle two conflicting goals. On one hand it is important to keep the anonymized data set as realistic as possible. On the other hand it is important to anonymize recorded data to ensure that identification of subjects is not possible in order to comply with privacy legislation. The new methodology is a trade-off between these goals.

The new methodology presented is intended to be the start of a comprehensive methodology encompassing most, if not all, protocols present in data networks. The analysis of protocols is a time consuming task. It would be impossible in a Master's Thesis to devise a complete methodology and at the same time ensure that every aspect is covered. This thesis will therefore focus on the HTTP/1.1 protocol and conduct a thorough analysis of this. Some network and transport layer protocols are also presented in this chapter. However, the analysis is not conducted as thoroughly as for the HTTP protocol. Among else, while HTTP header fields are classified according to the suggested classification system, this has not been done for the network and transport protocols. The considerations of these protocols are still included since they have significance to the prototype implementation. The presentation may also serve as a basis for a more thorough analysis.

The common way to operate when information is going to be changed, is to mark the information being subject to alterations and work on the marked information. The methodology presented here attacks the problem from the opposite direction by marking the information that should be *preserved*. All other information will be anonymized. This approach is presented in [13] as a "filter-in" method, and ensures that no identifying information is revealed by accident.

An important point is to decide what information to alter and what information to preserve. In a stringent filter-in methodology the operator of the anonymizing software will make all decisions about what to leave in clear. All information not marked will be anonymized. A not so stringent approach would be to leave some information in clear even though it is not marked by the operator. This information is of course considered not to endanger privacy in any way. Also, some information providing a high possibility of identification could be anonymized without the operator's interference. The methodology presented here will take such a semi-stringent approach.

3.1.1 Classification scheme

To classify what information should *automatically* be left in the clear and what information the operator should *decide* to be left in the clear, a classification scheme consisting of four classes is devised. The four classes are:

Must ... be anonymized. This is information with the highest potential to identify a person. This information is always anonymized by the anonymization software.

Should ... be anonymized. This is information with a limited potential to identify per-

sons. The decision for leaving in the clear or anonymizing is left to the operator of the anonymization software.

Could ... be anonymized. This is information most unlikely to identify a person. Under certain conditions identification may be possible. The decision for leaving in the clear or anonymizing is left to the operator of the anonymization software. The information placed in this class should be anonymized if complete assurance of anonymity is needed.

No ... anonymization necessary. This information has no potential to identify persons. The information is always left in the clear by the anonymization software.

The boundaries between the four classes are not strictly defined. This classification scheme is meant to support different legal requirements found in different countries. Clearly identifiable information is anonymized, and information not susceptible to identification of persons is left in the clear by default. The process of anonymization/non-anonymization is also carried out without human intervention. The operator must make decisions about not so obvious information based on the legislation of the country he or she resides in. The reason for leaving two classes for operator decision is to give a hint to the operator about the likelihood of identification. The class *Should* indicates a greater likelihood for identification than the class *Could*.

3.1.2 How to anonymize

There are two methods to choose between when performing the anonymization process. First, the information subject to anonymization may be entirely or partially removed. The drawback with this approach is that this will render the the data set more unrealistic than alterations to another possible value. The other approach is to alter the information to a value not having identifying properties. This is the approach used in the presented methodology. There are also different alteration possibilities, e.g. randomization, sequence, prefix-preserving. These are presented in Section 2.3. The methodology alters the values to new values not likely to occur in a real data set. Using a realistic value could lead to faulty identification of subjects. Instead, information is altered to values reflecting the type of value that should be present.

Altering information to other values makes it possible to retain the length of the original information. The methodology acknowledges the possibility that preserving the lengths may enable an attacker to deduce the original value. In most cases this will not be a big threat, since the information with the greatest potential of identification may represent a big number of possible values when altered.

3.2 Identifying information

Several parts of a network packet may contribute to the identification of a person. In some cases just a single field, like IP address, is enough to enable identification. In other cases a combination of fields may reveal identity. Identifying information in link, network, transport and application level *headers* are easier to consider than plain payload and body data. This is of course because the contents of header fields must follow a specific syntax and their semantics are well defined. In the present methodology, recognizable information, like header fields defined in RFCs¹, are thoroughly investigated for

¹Request For Comments

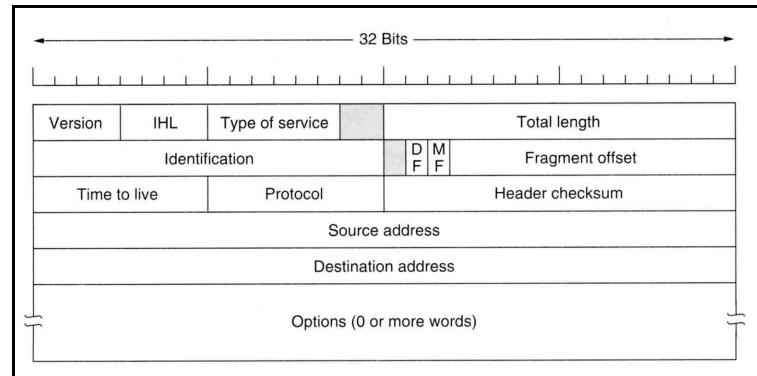


Figure 1: IPv4 header[1]

identifiable information and classified according to the methodology. Information not being recognized (e.g. body data) is placed in the *Must* class for automatic anonymization.

While some identifying information is easy to reveal, others may be hard to spot. Such information enables an attacker to conduct inference attacks. Examples are fingerprinting of operating systems, software, files, servers and clients etc. This has been accounted for in the present methodology. Such information is placed in appropriate classes according to its likelihood of identification.

The process of anonymization may lead to loss of information needed for attack detection. While an important goal in the anonymization process is to preserve as much information as possible, it is important to realise that the process in many cases is a trade-off between these goals. However, due to legal and ethical concerns the anonymization process should always prioritize anonymization in the case of a conflict between the two goals. At the implementation level, the decision of what to filter may be left to the user of the anonymizing software. [13] describes a filter-in methodology where all fields are anonymized except those explicitly stated by the operator. This thesis will also use this approach.

In this chapter the IP, TCP, UDP and HTTP headers will be examined for identifying information. While IP, TCP and UDP headers are considered more superficially, HTTP headers are subject to a thorough analysis. HTTP headers will also be classified according to the suggested classification system. How the different HTTP header values should be altered is also discussed. It is important to consider the importance of the fields in regard to intrusion detection. This knowledge is necessary for making adequate decisions regarding the trade-off between anonymity and realism. Snort[25] rules have been investigated to find out what kind of information it uses in the process of detecting attacks. This makes it possible to say something about how anonymization of the information influences the number of positives detected by Snort.

3.3 Identifying information in network and transport protocols

3.3.1 IPv4 Header

This subsection examines all fields of the IP header, of which several fields could reveal person's identity. Discussion of the importance of the fields for intrusion detection is also provided where a conflict between anonymization and intrusion detection is present.

Version

The 4-bits version number is either 0100 for IPv4 or 0110 for IPv6 and is needed for an IDS to correctly interpret a packet. This field is considered not necessary to anonymize.

Internet Header Length

The 4-bits Internet Header Length is necessary because the Option field can be of arbitrary length. In some cases specific header lengths may reveal specific applications in that these applications use specific options. This may identify both an application and an operating system. Header lengths shorter than 5 may trigger an IDS to alert for scanning attacks[26].

Type of Service

The 8-bits type of service (TOS) field consists of a 3-bit "Precedence" field, a 4-bit "Type-of-Service" field, and a 1-bit "MBZ" (must be zero) field. Due to bad implementations, the IP precedence field in ICMP error messages can reveal the operating system[27, 28]. The "Type-of-Service" field can help an attacker to identify possible applications[29]. In the case where the TOS value is unique to an application the application can be identified.

Total Datagram Length

The 16-bits total length field gives the total length of the datagram including headers, options and payload. As for the header length field, applications can be identified based on unique options and payloads. [30] states that the operating system can be identified due to miscalculations of this field. In rare cases this may lead to the identification of the user of the application. This is, though, considered so rare that anonymization of this field is not necessary.

The payload length, which can be calculated by subtracting header length from the total datagram length, can also indicate the application. However, the payload length alone is not enough for complete identification. An IDS may identify some attacks based on the payload size (e.g. Tiny Fragment, where the first fragment is so small that the complete TCP header does not fit into the packet, and nmap ping, where the payload is of size 0).

Identification

The 16-bits identification field is used in the reassembly of datagrams[31]. Some operating systems may echo this field incorrectly[30], making OS identification possible. An IDS may detect some flooding attacks where the identification field is part of the attack's signature.

Flags

The 3 flags are used in the reassembly of datagrams. Some operating systems may echo this field incorrectly, making OS and application identification possible. Invalid flags may be detected by IDSs because such packets can be used to fingerprint some operating systems and to get past firewalls[32].

Fragmentation Offset

The 13-bits offset field is used in the reassembly of datagrams. Some operating systems may echo this field incorrectly, making OS identification possible.

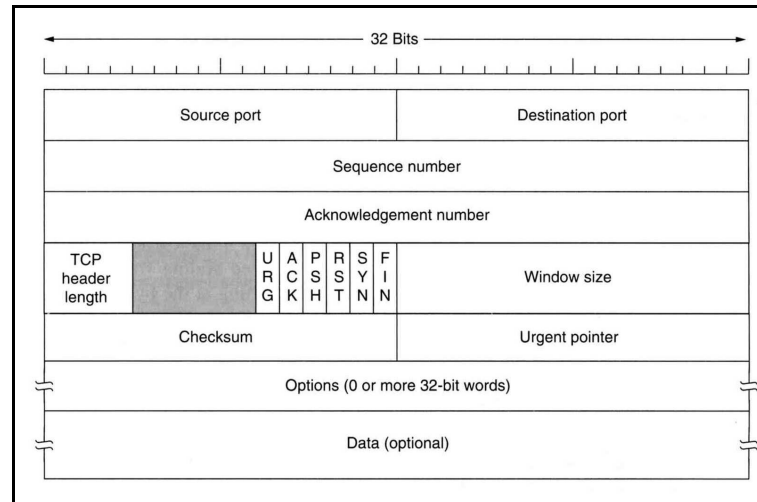


Figure 2: TCP header[1]

Time To Live

The 8-bits TTL field tells how many hops between nodes a packet is allowed to travel. At each node the value is decremented by one. An attacker can use this field to estimate how many hops the packet has travelled from the source host.

Upper Layer Protocol

This 8-bits field usually identifies the upper transport protocol. However, in the case of tunnelling, other level protocols may also be used (e.g. IP tunnelling). This may contribute to identify the application used.

Header Checksum

This 16-bits field is used to detect errors in the IP datagram. Some operating systems may calculate the checksum incorrectly. An IDS should trigger on packets with bad checksums since this may indicate an insertion attack[33].

Source IP Address

This 32-bits field may clearly identify a person, since the address may be linked to a specific host used by only one person. IDSs may also detect attacks based on source IP address.

Destination IP address

This 32-bits field may clearly identify a person, since the address may be linked to a specific host used by only one person. IDSs may also detect attacks based on destination IP address.

Options

There may be zero or more options. For example, the Bro anonymizer[13] anonymizes all options except maximum segment size, window scaling, SACK negotiation and time-stamps, which are preserved.

3.3.2 TCP Header

Source and Destination Ports

Port numbers may be used to identify a particular machine that runs a particular set of services, if the set of port numbers is in some way unique[14]. If that machine is only used by a single person, the person will be identified. IDSs use port numbers to detect many types of attacks.

Sequence Number

The sequence number may leak information in a way that the operating system may be fingerprinted[14, 34]. A person being the only one using this operating system may be identified. [35] states that an attack can exploit flaws in the interpretation of sequence numbers.

Acknowledgement Number

The acknowledgement number is the last received sequence number plus one, and is used to tell the receiving party that a packet is received and what the expected sequence number of the next segment is. Since the acknowledgement number may be derived from the sequence number, fingerprinting the operating system may occur. [36] states that the acknowledgement numbers may be used to create covert channels, which stateful IDSs should be able to detect.

TCP Header Length

The header length can indicate the application if the application uses unusual options. An IDS might trigger on TCP headers of unusual sizes.

Reserved

The reserved field is reserved for future use and should not be used. If some operating system or application use this field it could be identified, especially if unique values are used. IDSs could trigger on use of this field since it could be used for covert channels.

Flags

If an application uses a unique combination of flags, the application may be identified. Certain combinations of flags may be used in attacks such as Denial of Service and to bypass firewalls and IDSs.

Window Size

The windows size does not contribute to any kind of identification.

Checksum

Like for the IP header checksum, some operating systems may calculate the checksum incorrectly. This may lead to the identification of the operating system.

Urgent Pointer

If the URG flag is set to 1, the urgent pointer is used as an offset from the sequence number to indicate where urgent data can be found. If only certain applications use these fields, they may give an attacker an indication of the application used.

Options

The TCP options Loose Source Routing or Strict Source Routing gives routing information and may identify the communicating hosts. An attacker may use source routing as part of an attack, e.g. to get replies back to a known host if he spoofs an IP address.

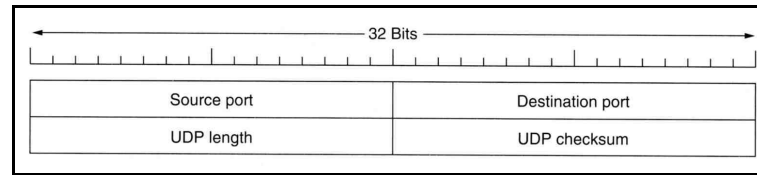


Figure 3: UDP header[1]

[14] rewrites the timestamp option due to clock drift manifestations. This may fingerprint a specific host, leading to eventual identification in the future[37]. An IDS use the timestamps for analyzing TCP dynamics in order to detect duplications and reordering of packets. [14] solves these conflicting goals by transforming the timestamp into a monotonically increasing counter without relation to time. This is done independently for each host. This way the uniqueness and transmission order of segments are preserved even though the actual timing information is lost.

Padding

Padding is used to align a header at a 32-bit boundary. An operating system or application could give the padding an unusual value. This may be used to fingerprint the operating system or application.

3.3.3 UDP Header

Source and Destination Ports

Like TCP port numbers, UDP port numbers may also be used to identify a particular machine that runs a particular set of services, if the set of port numbers is in some way unique[14]. If that machine is only used by a single person, the person will be identified. IDSs use port numbers to detect many types of attacks.

Length

UDP length may indicate the application level protocol, identifying an application.

Checksum

If all IP header fields are known, with the exception of the IP addresses, the checksum can be used to produce a possible list of IP addresses.

3.4 Identifying information in application headers and payload

3.4.1 HTTP 1.1

HTTP 1.1 is defined in RFC2616[38]. The following fields are part of this definition. Some additional header fields presented in other papers are also included due to their common appearance. The references for these will be given where appropriate. RFC3864[39] describes registration procedures for additional header fields. RFC4229[40] lists 81 more HTTP header fields, many being provisional. The inclusion of these is left to the next version of the methodology.

To devise the methodology, Snort[25] rules are inspected to see which headers Snort uses in the process of detecting attacks.

There are two types of HTTP messages: requests and responses. Some HTTP message headers can be general for both requests and responses. Other message headers belong to either requests or responses. Yet other headers belong to HTTP entities (see page 26).

General headers

Cache-Control

This header field prevents adverse interference of requests or responses by caches along the path between two communicating hosts. Two cache-directive values have a small potential to make an identification. These response directives, `private` and `no-cache`, can include some optional field names. Although such field names seldom contain a personal identifier, there is a slight possibility for this. Also, [38, p. 72] enables additional extensions to be written. For example, [41] presents an extension for group caching. This extension enables a server administrator to define groups for which caches will act differently. Usually such naming will not make it possible to identify a single individual. There may however be situations where identification is possible. Other not yet written extensions may also make it possible to identify users in the future. The conclusion for this field is that identifying information may occur, although the probability is very small. The header is not present in any Snort rule[42] and is not considered having any significance in the task of detecting intrusions. Anonymization of this field should therefore have no influence on the number of positives.

Class: *Could*.

Substitution: Iterations of the string "cache".

If an operator chooses to anonymize the `cache-control` directive, only those specific values mentioned will be anonymized in addition to all extensions added in the future.

Connection

This header field enables the sender to specify options needed for that specific connection. No options defined can compromise privacy. The header is not present in any Snort rule and is considered not to have significance regarding intrusion detection.

Class: *No*.

Date

The date represents the time at which the message was created. It is sent in RFC1123[43]-date format. [18] states that `Date` may reveal sensitive information although no explanation for this is mentioned. In the present methodology this header field is not considered endangering privacy. No indication has been found suggesting other classification than *No*. This header is not mentioned in the Snort rule set and is not considered significant for detecting intrusions.

Class: *No*.

Pragma

This field enables implementation-specific directives applicable to any recipient along the request/response chain. As for `Cache-Control` this field is also meant to be extensible. However, [38, p. 84] states that no new directives will be defined. This field is considered no danger to privacy. `Pragma` is not used in the current Snort rule set and is not considered significant for detecting intrusions.

Class: *No*.

Trailer

This header field says that the header fields listed in its option part are found in the trailer of a message, after the message body. `Trailer` is only used when `chunked`²

²The message body is transferred as a series of chunks, each with its own size indicator.

Transfer-Encoding is applied to the message. This field has no implication regarding privacy. However, it must be accounted for in an implementation of the anonymizer, since some header fields may be found after the entity body. The chunked encoding also implies that the message must be decoded before parsing. *Trailer* is not found in the Snort rule set and is not considered significant in intrusion detection.

Class: *No*.

Transfer-Encoding

This field indicates the type of transformation a message body is subject to. No options defined for this field are threatening privacy. However, as mentioned for the former header, it might have an influence on the implementation of the anonymizer. *Transfer-Encoding* is used in Snort to detect certain web-application attacks (e.g. sid³ 1618r17, 1806r11 and 1807r11), against Microsoft Internet Information Services.

Class: *No*.

Upgrade

The client can tell the server about the additional communication protocols it supports if the server will switch protocol. The server uses *Upgrade* to tell the client what protocol it switches to. The values for this field are not considered making positive identification possible. *Upgrade* does not occur in the Snort rule set and is not considered important in detecting intrusions.

Class: *No*.

Via

This field is meant to be used for tracking message forwards, avoiding request loops and identifying the protocol capabilities of all senders along the request/response chain. The option-part *received-by* annotates the receiving host, leading to a possible user identification. The rest of the values are not a threat to privacy. There may be several *Via* fields in a message. *Via* does not occur in the Snort rule set and is not considered important in detecting intrusions. Anonymization of this field should have no effect on the number of positives for an IDS.

Class: *Must*.

Substitution: `www.foo...foo.bar`

When anonymizing, only the *received-by* option needs to be altered.

Warning

The *Warning* header is used to carry additional information about the status or transformation of a message. The *warn-agent* value, declaring the host name, may lead to the identification of a user. The *warn-text* value consisting of a quoted string in natural language could also, if poorly designed, be the cause of identification. This is, however, such an unlikely situation that it is not considered a problem. The *Warning* header might appear several times in a message. *Warning* does not occur in the Snort rule set and is not considered important in detecting intrusions. Anonymization of this field should have no effect on the number of positives for an IDS.

Class: *Must*.

Substitution: `www.foo...foo.bar`

When anonymizing, only the *warn-agent* value need to be altered.

³Snort rule identifier

Request messages

A request message has this format[38, p. 24]:

```
Request = Request-Line
        *((general-header
        | request-header
        | entity-header ) CRLF)
        CRLF
        [ message-body ]
```

The request line has the format:

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

The method can be one of the following:

```
Method = "OPTIONS"
        | "GET"
        | "HEAD"
        | "POST"
        | "PUT"
        | "DELETE"
        | "TRACE"
        | "CONNECT"
        | extension-method
extension-method = token
```

Altogether this means that an HTTP request consists of a method followed by a space (SP), followed by a URI (Uniform Resource Identifier, e.g. www.hig.no), followed by HTTP-version (e.g. HTTP/1.1), followed by CRLF (Carriage Return - Line Feed). The subsequent lines are headers with their corresponding values. The character "|" means "or". Several headers are divided by CRLF. After the last header with its corresponding values and CRLF comes another CRLF indicating the end of headers and the start of the message body. This format makes it fairly easy to parse an HTTP message. [38] also opens for the addition of other methods (*extension-method*).

Method

Method is a directive the client uses to instruct the server to give a certain type of response. No method values are considered sensitive.

Some request methods may in conjunction with a specific *content* or *uricontent*⁴ be used in Snort rules to detect suspicious activity. Examples of this are *DELETE* and *TRACE* used in Snort rules sid 1603r7 and 2056r4 respectively. *GET*, *HEAD* and *POST* are methods used in conjunction with specific values in the *content* or *uricontent* parts of some Snort rules to detect possible attacks. Here are some examples:

GET is in Snort used in conjunction with "/" (sid 306r10 and 1881r6) and "x" (sid 1375r6) as *Request-URI* to detect suspicious activity.

⁴*content* and *uricontent* are Snort rule options

HEAD is used in Snort rule sid 1139r7 in conjunction with the Request-URI `"/./"`.

POST is used in sid 939r11 with the `uricontent="/author.dll"` and in sid 3629r3 in combination with `content:"/search/results.stm"`.

PUT and CONNECT are not associated with any Snort rule.

Class: No.

Request-URI

The Request-URI is the address of the resource requested by the client. The generic syntax and semantics for URIs are defined in RFC2396[44]. Parameters of the URI are also included in the Request-URI. A URI may identify a user both by itself or if a specific SQL⁵ query string is part of the URI. This speaks for anonymization of this field.

Different anonymization strengths of the Request-URI are classified this way:

Must: The domain part (if present, e.g. `www.mydomain.com`) of the Request-URI is always anonymized.

Should: The Request-URI is anonymized until the second to last `"/"` (e.g. until `/etc/passwd`).

Could: The entire Request-URI is anonymized.

The domain part should, according to [38], only be part of the Request-URI in requests to proxies and between proxies. Servers should also understand a URI with a leading domain part, even though no client or proxy should issue request including the domain to a server. This means that in most cases no anonymization of the Request-URI will occur when implementing *Must* Request-URI anonymization.

The reason for anonymizing the path until the second to last `"/"`, as suggested for *Should* Request-URI anonymization, is that many attack signatures use the last 2 levels of the path to detect attacks. Table 1 in page 41 shows the distribution of path levels used in Snort signatures.

Chapter 4 describes experiments conducted for evaluating the methodology and the prototype. A data set consisting of real network traffic mixed with Nessus generated attacks was prepared for the experiments. After analysing the Nessus attacks it was found that many attacks were looking for `/etc/passwd` on the server. Also, many attacks were cross site scripting attacks, including a `<script>` tag in the URI. The script tag was in some cases also given in hexadecimal form as `"%3cscript%3e"`, capitalized or not. Because of the large occurrence of these attacks, the methodology treats these attacks specifically. In an implementation of the *Could* Request-URI anonymization, being the strongest anonymization provided, these patterns will be searched for. When found, the Request-URI will be anonymized until the occurrence of these patterns. For *Should* Request-URI anonymization only the script-tag is treated specifically since `/etc/passwd` will be preserved by the scheme itself. It could be argued that the two types of attacks could also be treated specifically for *Must* Request-URI anonymization. This anonymization scheme is however invented to preserve as many attacks as possible. In case a Snort rule would trigger on parts of the URI coming before the script- or passwd-parts, these attacks would be rendered ineffective.

⁵Structured Query Language

A URI can also be formed in a way that an attacker may be able to execute code on the server remotely. An example is a number of "../" sequences followed by "/winnt/system32/cmd.exe?...", followed by a command to be executed by cmd.exe. Signatures for such attacks are present in most signature-based IDSs. Snort rules use the option `uricontent` to detect such suspicious values. An example is Snort rule sid 1002r8 using `uricontent:"cmd.exe"` to warn for "WEB-IIS cmd.exe access".

Alterations of the Request-URI will influence on the number of positives since the attack will be removed when anonymizing the URI.

Class: *Must*.

Substitution: Iterations of the string "n".

The anonymization should retain the path levels using "/" or "\".

HTTP-Version

`HTTP-Version` is used to tell the server how to interpret the packet. It is not considered sensitive and is therefore not candidate for anonymization. `HTTP-version` is used in some Snort rules in combination with other `content` values to detect suspicious activity. Examples on this are sid 2090r11 ("WEB-IIS WEBDAV exploit attempt") and sid 1881r6 ("WEB-MISC bad HTTP/1.1 request, Potentially worm attack").

Class: *No*.

The following is a list of headers bound to request messages:

Accept

This header field specifies which media types are acceptable to receive in the response. No values for this header field are expected to contribute to the identification of a user. There may be several `Accept` header fields in a message. Snort rule 2090r11 use `Accept` as part of `content` to detect suspicious activity.

Class: *No*.

Accept-Charset

This header indicates which character sets are acceptable for the response. In some circumstances character sets rare for the environment may contribute to the identification of a user. An example might be the presence of ISO-8859-11 in an environment typical for a Sami environment, which normally would use ISO-8859-1 or Windows-Sami-2. This is maybe a far-fetched situation, although possible. `Accept-Charset` is not found in any standard Snort rule and is therefore not considered significant for detecting intrusions. Anonymization of this field should have no influence on the number of positives.

Class: *Should*.

Substitution: Iterations of the string "charset".

Accept-Encoding

The `Accept-Encoding` header states which content encodings are acceptable to be present in a response. No values belonging to this header are of any concern regarding privacy. This field is not found in the standard Snort rule set and is therefore not considered significant for detecting intrusions.

Class: *No*.

Accept-Language

This header states the languages being acceptable for the response to the request. The header is explicitly stated in [38, p. 94] as a header field subject to privacy issues. With the same arguments as `Accept-Charset` this field should also be anonymized. `Accept-Language` is not found in any standard Snort rule and is therefore not considered significant for detecting intrusions. Anonymization of this field should have no influence on the number of positives.

Class: *Should*.

Substitution: Iterations of the string "l".

Authorization

This header is used in a request to authenticate the client to the server. The credentials following this header contain among else username, password and the URI copied from the `Request-URI` [45, p. 12]. It is necessary to anonymize this header. `Authorization` occur in several content and `pcrc`⁶ options in Snort rules, often along with other values. Anonymization of this field may therefore have influence on the number of positives.

Class: *Must*.

Substitution: Iterations of the string "credentials".

Cookie

The `Cookie` header field is defined in RFC2965 [46]. It is used to maintain and handle state in HTTP. `Cookie` has several options, many of them mirroring the `Set-Cookie2` response header. The options defined in [46] are `cookie-version`, `path`, `domain` and `port`. Of these, `path` and `domain` must be anonymized. `Cookie` can also be extended with other options. One extension, `login=0`, is also used in Snort rule 2441r4. This particular value should not be anonymized since it represents no danger to privacy.

Class: *Must*.

Substitution: Iterations of the string "cookie".

Options `cookie-version`, `port` and `login=0` are kept in clear. Other options are anonymized.

Cookie2

This header field is defined in [46]. If the client does not support the cookie version set by a server, the client sends a request with the `Cookie2` header set to the highest version the client understands. There is only one option defined, being the cookie version. This header is not subject to anonymization.

Class: *No*

Expect

This header indicates that particular server modes of behavior are required by the client. Although this header is extensible, new extension will hardly enable identification of users. No value-parts of this field are considered being a danger to privacy. `Expect` is not part of any Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

⁶Regular expressions library

From

This header contains, if used, an e-mail address of the user controlling the user agent. This field is clearly subject to anonymization. `From` is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions. However, customized rules may be defined to include specific addresses. In such cases anonymization will affect detection based on these rules.

Class: *Must*.

Substitution: Iterations of the string "email".

Host

The `Host` field gives the host and port number of the resource requested. This may on its own give positive identification of a user and is therefore subject to anonymization. `Host` is used in Snort rule 2091r9 to alert for an attempted-admin classtype attack. If anonymized this rule will not be triggered.

Class: *Must*.

Substitution: www.foo...foo.bar

If-Match

This field is used by a client to verify that one or more of its previously received entities are current. The value for this field includes the entity tag (`Etag`), showing which entity the client wants to verify. In most cases entity tags are not designed in a way that makes identification of a user possible. However, some HTTP implementations may include some identifying information. One example is [47], implementing usernames as part of the `Etag`:

... the current implementation adds the remote user name to the `Etag`[48].

Although such a situation is not common, this methodology acknowledges the threat and suggests anonymization of this field. The final decision is though left to the operator. `If-Match` is not part of any Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Should*.

Substitution: Iterations of the string "ifmatch".

If-Modified-Since

This field causes an entity update only if the entity is updated on the server after the date present as value in the header field. This field is, as the general `Date` header field, considered not a threat to privacy. `If-Modified-Since` is not part of any Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

If-None-Match

`If-None-Match` has the same syntax as `If-Match` and is used for the verification that none of the provided entities given by the entity tag values are current. This header ends up in the same class as `If-Match` with the same reason. `If-None-Match` is not part of any Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Should*.

Substitution: Iterations of the string "ifnonematch".

If-Range

This header is used to complete an entity if the client has just a part of the entity in its cache. Its syntax includes the entity tag associated with the entity. As for *If-None-Match* and *If-Match* this option could reveal identity if the HTTP implementation includes sensitive information. *If-Range* is not part of any Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Should*.

Substitution: Iterations of the string "ifrange".

If-Unmodified-Since

A server will, if this header is present, perform the requested operation only if the requested resource has not been modified since the date provided in the value part. This field is not considered a problem regarding privacy. *If-Unmodified-Since* is not part of any Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Max-Forwards

Max-Forwards defines the number of proxies or gateways that can forward the request. This header presents no danger to the identification of a user. *Max-Forwards* is not part of any Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Proxy-Authorization

This field has the same functionality as *Authorization*, but for a client to authenticate to a proxy or for authentication between proxies. The values include also here username, password and URI. *Proxy-Authorization* is not part of any Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Must*.

Substitution: Iterations of the string "credentials".

Range

The *Range* header enables a client to request parts of the entity in question. The values include no information being a threat to privacy. *Range* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Referer

The *Referer* header enables the client to specify the URI of the resource where the request-URI was obtained. This field is subject to anonymization because it contains an absolute or relative URI to a resource. *Referer* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions. One might think, though, that customised rules may be created containing specific *referer* values. In such cases, anonymization will affect the number of positives.

Class: *Must*.

Substitution: `www.foo...foo.bar`".

In some cases the *Referer* value might contain scripts, indicated by a `<script>` tag. In such cases the value will be anonymized until the tag occurrence.

TE

This field states which extension transfer encodings the client will accept and if it accepts trailer fields in a chunked transfer encoding. TE has no values being a danger to privacy. TE is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Translate

The `Translate` header is added by Microsoft for IIS⁷ websites enabling WebDAV⁸. The header is described in [49] and [50]. Snort uses `Translate` to detect attacks directed at IIS servers. The header has a boolean value of either "T" or "F".

Class: *No*

User-Agent

The `User-Agent` field states which user agent originates the request. This field may in rare circumstances identify a user if the user uses an uncommon [version of a] user agent. This situation is not a big problem due to huge IDS test data sets, although it is a possibility. Attackers can use this field to determine if a host uses a vulnerable user agent. `User-Agent` is used in some Snort rules to detect suspicious activity. One example on this is sid 1436r5 (`content: "User-Agent |3A|Quicktime"`). Anonymization of this field may have influence on the number of positives.

Class: *Could*.

Substitution: Iterations of the string "browser".

Response messages

A response message has this format[38, p. 26]:

```
Response = Status-Line
          *((general-header
            | response-header
            | entity-header ) CRLF)
          CRLF
          [ message-body ]
```

The response line has the format:

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

The status code is a three digit code where the first digit defines the class of response.

The classes are:

```
1xx: Informational
2xx: Success
3xx: Redirection
4xx: Client Error
5xx: Server Error
```

⁷Internet Information Services

⁸Web-based Distributed Authoring and Versioning

The next two digits give the exact status message. The reason phrases defined in RFC2616[38, p. 27] are only recommendations, so the phrases might differ from those stated in the RFC.

Altogether this means that an HTTP response consists of the HTTP version, followed by a space, followed by a status code, followed by a space, followed by a reason phrase, followed by CRLF. The subsequent lines are headers with their corresponding options. Several headers are divided by CRLF. After the last header with its corresponding values and CRLF, another CRLF comes indicating the end of headers and the start of the message body. This format makes it fairly easy to parse an HTTP message. The RFC also opens for the addition of other status codes and reason phrases.

HTTP-Version

`HTTP-Version` is used to tell the client how to interpret the packet. It is not considered sensitive and is therefore not candidate for anonymization. `HTTP-version` is not used in any Snort rule for `from_server` activity, and is therefore considered not important for intrusion detection.

Class: *No*.

Status-Code

The `Status-Code` field is used to tell the client the status of the response. This field is not considered sensitive. Snort rule sid 1045r9 uses the `Status-Code` 403 to detect attacks leading to a "Forbidden" reply from the server.

Class: *No*.

Reason-Phrase

The `Reason-Phrase` is used to give a textual explanation in addition to the `status-code`. This field is not considered sensitive. Although these values are only recommendations, reason-phrases constructed in a way that could endanger privacy are considered a far-fetched situation. The only `Reason-Phrase` used in the default Snort rule set is `Forbidden` (sid 1045r9). This field is not considered important to intrusion detection.

Class: *No*.

The following is a list of headers bound to response messages:

Accept-Ranges

`Accept-Ranges` is used by a server to tell what byte ranges it accepts in a request. This field represents no problems regarding privacy. `Accept-Ranges` is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Age

This field represents the sender's time estimate of the time passed since the response was generated. Normally this field would not represent any danger regarding privacy. A far-fetched situation could occur if this field would be compared with the `Date` field of other messages with the aim to pinpoint a server. This situation is not considered a significant threat. However, the decision will be left to the operator. `Age` is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Could*.

Substitution: Iterations of the string "age".

ETag

This header defines the value of the entity tags (Etags) used in the headers *If-Match*, *If-None-Match* and *Vary*. In most cases entity tags are not constructed in a way that makes identification of a user possible. However, some HTTP implementations may include some identifying information. One example is [47], implementing usernames as part of ETag. Although such a situation is not common, this methodology acknowledges the threat and suggests anonymization of this field. However, the final decision is left to the operator. ETag is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Should*.

Substitution: Iterations of the string "etag".

Location

Location is used for redirection to another resource necessary to fulfil the request. The value is an absolute URI and is therefore necessary to anonymize. *Location* is used in the default Snort rule 2577r4. It is also highly possible that customizable rules include specific values. Anonymization of this field may for the default rule set have no significant consequences.

Class: *Must*.

Substitution: www.foo...foo.bar

Proxy-Authenticate

If a client must authenticate to a proxy, or a proxy to another proxy, but the receiving proxy does not receive credentials, the proxy sends a 407 *Proxy Authentication Required* with a *Proxy-Authenticate* header back to the client. This header includes the realm value with the name of the host performing the authentication and also the domain value, listing URIs related to the resource. Clearly this header must be anonymized. *Proxy-Authenticate* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Must*.

Substitution: www.foo...foo.bar

Retry-After

This field gives a client either a date and time or a number-of-seconds value to indicate when the service again should be available. It should be no problem regarding privacy. *Retry-After* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Server

The value of the *Server* header field contains information about the software used by the server to handle the request. In a test data set with a low number of servers this might pinpoint a particular server. However, the threat is not considered as dangerous as the *Must* class, and the decision is left to the operator. *Server* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Should*.

Substitution: Iterations of the string "server".

Set-Cookie

Set-Cookie is defined in RFC2109[51]. Of the options defined, *Max-Age*, *Secure* and *Version* have no possibilities to identify a subject. The *Domain* option is the header with highest identifying properties. Also, it is possible to define other options for *Set-Cookie*. The *Set-Cookie* header is not present in any Snort default rule.

Class: *Must*

Substitution: Iterations of the string "setcookie".

All options, with the exceptions of *Max-Age*, *Secure* and *Version*, must be anonymized.

Set-Cookie2

Set-Cookie2 is defined in RFC2965[46]. [46, p. 5] allows the option values to be "anything the origin server chooses to send". However, the options *Discard*, *Max-Age*, *Port*, *Secure* (having no value) and *Version* are information not needed to anonymize. Other options may be defined and should be anonymized. The default Snort rule set does not have any rules containing *Set-Cookie2*.

Class: *Must*

Substitution: Iterations of the string "setcookie".

All options, with the exceptions of *Discard*, *Max-Age*, *Port*, *Secure* and *Version*, must be anonymized.

Vary

The *Vary* field gives the set of request-header fields that fully determines if a cache is allowed to use this response to reply to a subsequent request without revalidation. Since the values are only header field names this header is not subject to anonymization. *Vary* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

WWW-Authenticate

If a client should authenticate to a server, but the server does not receive credentials, the server sends a 401 *Unauthorized* with a *WWW-Authenticate* back to the client. This header includes the realm value with the name of the host performing the authentication and also the domain value, listing URIs related to the resource. Clearly this header must be anonymized. *WWW-Authenticate* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *Must*.

Substitution: www.foo...foo.bar

Authentication-Info

This header field is defined in RFC2617[45, p. 15] and is used by the server to convey information about the successful authentication to a client. The values for this field are not considered subject to anonymization. *Authentication-Info* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Proxy-Authentication-Info

This header field is defined in RFC2617[45, p. 19] and is used by the server to convey information about the successful authentication to a proxy. The values for this field is not considered subject to anonymization. `Proxy-Authentication-Info` is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Entity headers

The entity is defined as the information transferred as the payload of a request or response[38, p. 8]. An HTTP entity consists of entity-headers and the entity-body. Some entities will only contain the entity headers.

The following is a list of headers bound to an entity:

Allow

This header lists the request methods supported by the resource given in the request-URI. Since the value only lists methods, this header has no privacy implications. `Allow` is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Content-Disposition

This header field is defined in RFC2183[52] and is not a part of the HTTP/1.1 protocol. Although a lot of security implications are associated with this header, it is often found in HTTP messages. The header is used for telling the client how the content should be presented. The only value-part being subject to anonymization is `Filename`. Even though the user agent should not respect any path information given as part of the `Filename` option, [52, p. 4] at the same time it implies that such information may be present. A filename by itself should be anonymized and this decision is strengthened by the possible inclusion of a path. `Content-Disposition` appears in some default Snort rules, e.g. sid 2589r4 ("`WEB-CLIENT Content-Disposition CLSID command attempt`"). Anonymization of this field may have some influence on the number of positives.

Class: *Must*.

Substitution: Iterations of the string "file".

Only the "Filename" option value should be anonymized.

Content-Encoding

`Content-Encoding` tells the receiving party which encoding is used for the entity. Encoding is mostly used to compress the document to be sent. The values will not reveal identifying information. `Content-Encoding` is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Content-Language

This field states the natural language(s) of the intended audience for the entity. As for the headers `Accept-Charset` and `Accept-Language` this header may contribute to the identification of a user. `Content-Language` is not part of any default Snort rule, and is

therefore not considered significant for detecting intrusions.

Class: *Should*.

Substitution: Iterations of the string "I".

Content-Length

This field gives the size of the entity-body. It is not considered a danger to privacy. *Content-Length* is part of some Snort default rules, e.g. sid 2090r11 ("WEB-IIS WEBDAV exploit attempt") and sid 2278r9 ("WEB-MISC client negative *Content-Length* attempt"). Since *Content-Length* is significant for detecting intrusions this methodology will preserve the payload lengths of the packets. This implies that when altering information, the substitution must be of the same length as the original data.

Class: *No*.

Content-Location

This field is used when the entity is found at another location than given in the URI. The value of this field is an absolute or a relative URI. As stated before, a URI might reveal identifying information. *Content-Location* is not part of any default Snort rule. However, it is easy to imagine situations where a system administrator wants to implement customized rules for this header. Employees fetching unwanted web page contents via this header may be detected by such a rule. Anonymization of this field is not considered to have a significant influence on the number of positives when using the standard rule set. The influence may be bigger if the rule set contains customized rules.

Class: *Must*.

Substitution: www.foo...foo.bar

Content-MD5

The value for the *Content-MD5* is an MD5 message digest of the entity body. The value poses no problems regarding privacy. If values in an entity are altered the MD5 must be recalculated. *Content-MD5* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Content-Range

This header specifies where the partial entity body should be applied to the body it belongs. No values belonging to this header are considered to be a danger to privacy. *Content-Range* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Content-Type

The *Content-Type* header indicates the media type of the entity body. The value for this header includes also the character set used for the body. With the same arguments as *Accept-Charset* and *Accept-Language* this field should also be anonymized. There are several Snort rules using *Content-Type*, e.g. sid 2925r3 ("INFO web bug 0x0 gif attempt") and sid 3473r3 ("WEB-CLIENT RealPlayer SMIL file overflow attempt"). Anonymization of this field may influence on the number of positives.

Class: *Should*.

Substitution: Iterations of the string "type".

Expires

This header gives the date and time after which the response is considered invalid. The value is in HTTP date format and is not considered a threat to privacy. *Expires* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Last-Modified

This header indicates the time when the server believes the entity was last modified. The value is a HTTP date and is not considered a threat to privacy. *Last-Modified* is not part of any default Snort rule, and is therefore not considered significant for detecting intrusions.

Class: *No*.

Message body

The message body carries the entity body of a request or response message. The message may in some circumstances be empty, e.g. when the request method is "HEAD".

It would be hard for an anonymizing program to intelligently parse and understand the content of the message body. For this reason this methodology will anonymize the message body completely. It is, however, important to remember that there may be a trailer containing headers following the message body. This is indicated by the *Trailer* header. In such cases chunked *Transfer-Encoding* is used, implying that the message must be decoded before parsing.

The message body may contain values used in an attack. An example of this is Snort rule 3552r4:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any
(msg:"WEB-CLIENT OLE32 MSHTA masquerade attempt"; flow:to_client,established;
flowbits:isnotset,http.hta; content:"R|00|o|00|o|00|t|00| |00|E|00|n|00|t|00|←
r|00|y|00|"; nocase; content:"|D8 F4|P0|B5 98 CF 11 BB 82 00 AA 00 BD CE 0B|";
within:16; distance:60; reference:bugtraq,13132; reference:cve,2005-0063;
reference:url,www.microsoft.com/technet/security/bulletin/ms05-016.mspx;
classtype:attempted-user; sid:3552; rev:4;)
```

Since the message body has to be anonymized, this may influence the number of positives.

Class: *Must*.

Substitution: Iterations of the string "x".

3.5 Correlation of headers and other data

The identification of users based on correlations of header fields not being anonymized as described in the methodology is not considered a problem. No remaining headers kept in clear can contribute to user identification when correlated to other headers kept in the clear. If, however, user behavior patterns could be analysed, these could be correlated with fields containing HTTP-dates (e.g. the *Date* and *Last-Modified* header fields). However, such positive user identity deduction should not be possible to conduct, since

IP-addresses and other identifying information are altered. The threat will increase if a prefix-preserving IP-address mapping is used, and if the part of the IP address not being part of the prefix is always mapped to the same value.

Another threat to privacy may occur if the user uses an uncommon operating system or application and these may be pinpointed due to non-standard protocol implementations. Such bad implementations could lead to certain values erroneously calculated (e.g. IP header `Content-Length` and HTTP `Content-Length`).

As a conclusion, correlating headers and other data *may* contribute to the identification of a user. There are however so many conditions that must hold for this to happen that this methodology does not consider this a problem.

3.6 Methodology conclusion

A new methodology for anonymization of network traffic is presented in this chapter. The methodology focuses on application layer anonymization. The HTTP protocol is thoroughly analyzed and the recognizable information is classified according to the devised classification system. The classes used are *Must*, *Should*, *Could* and *No*. Appendix A gives an overview over the classification of HTTP/1.1 fields presented in this chapter.

The methodology also analyzes the known header fields belonging to IPv4, TCP and UDP. Since this thesis focuses on application layer information, the analysis for network and transport protocols are not so thorough. Also, the information in network and transport layers are not classified. This is handed over to further research. The information regarding these protocols is provided as a basis for future research.

4 Experimental Work

4.1 Introduction

The idea for the experiments was to find out what influence differently anonymized data sets have on intrusion detection in comparison with a non-anonymized data set. Different anonymization schemes were applied to the non-anonymized data set in order to produce data sets with different anonymization strengths. The number of attacks retained in the different anonymized data sets were estimated using Snort. This way an approximation could be made of how large percentage of attacks were retained. Note that the number of positives detected by Snort was measured, not the true positive or false positive rates. This way no ROC¹ curves have been calculated.

Using Snort for counting attacks in the data set may be considered a weak measurement. The best way would of course be to inspect the data set manually. This would be a much too laborious task for a data set of 85 megabytes. This is the reason for using Snort to give an estimate of the level of attack retention. See Section 4.4.1 on how Snort was configured.

4.2 Prototype

The prototype is called Anonymator. It is "terminating" (or rather altering) information having the potential to identify users. The prototype is built to demonstrate how this type of software might work. It could be further developed into a fully functional product or it could be discarded. If discarded, the knowledge derived from developing and using the prototype could be used in building a functional model from scratch or devising a theoretical model.

The goal was to implement a prototype proving the concept of the methodology. Since this methodology concerns payload anonymization, only payload data is anonymized in Anonymator. Link, network and transport level information is copied to the new packet, with the exception of those fields being dependent on the payload, e.g. the TCP checksum. Such fields are recalculated to contain the correct values for the packet. When not implementing anonymization of the lower layers, conducting experiments will make it easier to interpret the results of the experiments. Examples are the interpretations of differences in the number of positives for testing IDSs using a raw recorded data set in one experiment and using an anonymized data set in another experiment. The anonymizer should in such a case optimally produce the same number of positives. However, it is expected that the experiment using the anonymized data set will produce a lower number of positives since information needed for detecting suspicious activity is altered. If lower layer protocols are *not* anonymized it is more likely that the difference in the number of positives is caused by alterations of the payload. If lower layer protocols *are* anonymized the difference could be caused by alterations made to this information as well as alterations done to the payload.

¹Receiver Operating Characteristic

The prototype does not implement IP defragmentation and TCP reassembly. This should be done in a real anonymization product in order to provide a more intelligent anonymization. If for instance an attack consists of a specific payload and this payload is scattered over two or more datagrams, the anonymizer could reassemble the payload, detect that the payload contains an attack and then preserve the payload. In Anonymator the scattered payload will be anonymized.

4.2.1 Architecture and implementation

Anonymator is a console application providing four anonymization schemes:

1. **Strongest** implements the classes *Must*, *Should* and *Could* with no other user interaction than giving input and output file names.
2. **Strong** implements the classes *Must* and *Should* with no other user interaction than giving input and output file names.
3. **Weak** implements the class *Must* with no other user interaction than giving input and output file names.
4. **Customized** implements the class *Must*. The decision of whether to anonymize or not the header fields in the classes *Should* and *Could* is left to the operator of Anonymator. Compliant to the methodology, the operator chooses which *Should* and *Could* fields should be *left in clear*. The other *Should* and *Could* fields will be anonymized. The operator also chooses which anonymization strength to apply to the Request-URI.

Header fields classified as *No* will not be altered in any scheme.

Anonymator is coded in C and can be run on any Linux system having the `libpcap`[53] library installed. A port to Microsoft Windows is also implemented, requiring the `wincap`[54] library installed. C is chosen because `libpcap/wincap` includes functions, which can be used directly in C. There are Java and C# wrappers available for `wincap`. However, none of these provide the functionality needed for the prototype. The `libpcap/wincap` libraries provide an application interface to the physical layer. Through `*pcap`, applications can sniff packets from the network and send packets into the network. `*pcap` can also read from and write to files using the `libpcap` format. The current version of Anonymator uses this approach, taking a non-anonymized data set as input, and producing an anonymized data set as output.

The source consists of these files:

AnonMain.c is the driver for the application. It provides the main user interface, iterates through packets fetched from the input file and saves the anonymized packets to the output file.

LoadSave.c is responsible for loading the input file. As the file name suggests it should also be responsible for saving the output file. This specific functionality was moved to `AnonMain.c`.

HeaderChoices.c provides a user interface used when the Customizable anonymization scheme is chosen. The file lists all headers subject to choosing, with explanations and hints. Choices and hints regarding the Request-URI are also given.

AnonEngine.c is the main engine for anonymization. Here each packet will be parsed

and inspected, and sent to appropriate anonymization functions residing in other files, such as `AnonHttp.c`. Link, Network and Transport layer information is kept as they are, with the exception of information dependent on the application layer information. An example is the TCP checksum, which must be recalculated due to alterations in the payload.

`AnonHttp.c` is responsible for anonymization of HTTP traffic. The anonymization is done according to the choices made by the operator.

`anonheaders.h` provides access to functions in other files than the present.

`Printing.c` is implemented to display on screen the contents of the packets. This file is provided as a testing facility. To enable the printing function, the line `//printPayload(pkt_data);` in `AnonMain.c` must be uncommented.

The header fields `Cookie2`, `Set-Cookie`, `Set-Cookie2` and `Translate` are not implemented in Anonymator because they were discovered after the experiments were conducted. They are put on the to-do list for further development. The headers are, though, included in the methodology, Section 3.4.1.

Acknowledging that Anonymator is just a prototype, some parts of the methodology have not been implemented. An example is anonymization of the `Request-URI`, where path-level separation using `"/` is not retained. The anonymized part is anonymized using iterations of the letter `"n"`, regardless of how many levels the URI encompasses.

4.2.2 Pseudocode

This thesis will not include the entire source code for Anonymator. The source code will, though be provided to anyone who requests it. Instead of listing the source code, a shorter pseudocode is presented here:

```
begin
list anonymization schemes
  1. Strongest
  2. Strong
  3. Weak
  4. Customized
operator makes selection
ask for inputfile
operator gives input file

if(Customized anonymization)
{
  list URI—anonymization schemes
operator makes selection of URI—anonymization
  list header anonymization choices
operator selects headers to keep in clear
}

ask for output file
operator gives output file
```

```
Open input file
loop(for every package)
{
  if(payload is HTTP)
  {
    loop(for each line (parse payload based on "\r\n"))
    {
      if(Strongest URI—anonymization)
      {
        if("<", "%3c", "%3C" or "/etc/passwd" is recognized)
          Anonymize until these occurrences
        else
          Anonymize entire URI with exception of leading "/"
      }
      else if(Strong URI)
      {
        if("<", "%3c" or "%3C" is recognized)
          Anonymize until these occurrences
        else
          Anonymize until second to last "/" with exception of leading "/"
      }
      else(Weak URI)
      {
        Anonymize only domain part, if present
      }
    }

    if(Strongest header anonymization)
      Anonymize Must, Should and Could headers
    else if(Strong header anonymization)
      Anonymize Must and Should headers
    else if(Weak header anonymization)
      Anonymize Must headers
  }
}
else(payload is not HTTP)
{
  leave packet alone
}

recalculate TCP checksum
assemble package
save packet in new file
}
End
```

4.2.3 Using Anonymator

Anonymator is started from the command line without arguments. The program will ask which scheme to use for the anonymization. The first three schemes, Strongest, Strong and Weak implements varying levels of anonymization according to the methodology.

Section 4.2.1 explains how the classification is implemented with respect to the anonymization schemes.

If the Customized scheme is chosen, Anonymator will first ask which type of Request-URI-anonymization to use. These types are presented in Section 3.4.1. Next, the operator must choose which headers should be kept unaltered. Only the headers classified as *Should* and *Could* can be chosen. The *Must* headers are automatically anonymized because of their sensitive nature. The *No* headers are always left in the clear. The *Should* and *Could* headers not chosen will automatically be anonymized. The operator is given hints on the severity of the different choices through the interface.

Anonymator also asks for the input and output file names to be used. The input file must be in libpcap format. The output format is also libpcap.

4.2.4 Testing Anonymator

To investigate if Anonymator works as expected, several tests have been conducted. The following screenshots will show some examples of the influence Anonymator has on one particular packet in a data set when different anonymization schemes are applied. The screenshots are taken from Ethereal[55] representations of the packet in question. For the record, the packet is not part of the data set provided by the IT department of Gjøvik University College[56]. It is part of a data set recorded in non-promiscuous mode, recording the authors web requests to this thesis' project web page. The package and its relevant URIs have been inspected and found not to endanger privacy for any person.

Figure 4 shows the packet without any anonymization applied. Figure 5 shows the same packet when the Weak anonymization scheme has been applied. Note that only the *Host* and *Referer* header fields are anonymized. According to the policy and implementation, it is only *Must* headers that are anonymized under the Weak anonymization scheme. Weak Request-URI anonymization is applied through anonymizing only the domain part, if present, of the URI.

Figure 6 shows the packet after the Strong anonymization scheme has been applied. In addition to the *Must* headers mentioned above, the *Should* headers *Accept-Language* and *If-None-Match* are anonymized. Since the Request-URI consists of only two path levels, no anonymization is applied to the present URI. If the URI has three or more levels, the parts preceding the last two levels would have been anonymized.

Figure 7 shows the influence the Strongest anonymization scheme has on the packet. In addition to the above mentioned headers, the *User-Agent* header is also anonymized. This header is classified as *Could* and implemented in the Strongest anonymization scheme. It can also be seen how the Strongest Request-URI-anonymization influences the packet. This scheme anonymizes the whole URI, with the exception of URIs containing a script tag or the string `/etc/passwd`, where the URI is anonymized until these occurrences. Note that the current version of Anonymator does not retain path-separations.

Figure 8 shows an example of anonymization applying the Customized anonymization scheme using Weak Request-URI anonymization and no anonymization of *Should* and *Could* headers. As expected, the URI is presented as the original. Also, only the *Must* headers are anonymized. In fact, this is the same anonymization as the plain Weak anonymization presented in Figure 5.

Figure 9 shows the result after a Customized anonymization scheme using Strongest Request-URI anonymization and all *Must*, *Should*, and *Could* headers anonymized. If

```
GET /img/progress.jpg HTTP/1.1\r\n
User-Agent: Opera/9.00 (X11; Linux i686; U; en)\r\n
Host: www.infosikring.no\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0\r\n
Referer: http://www.infosikring.no/progress.html\r\n
If-Modified-Since: Mon, 27 Feb 2006 14:50:11 GMT\r\n
If-None-Match: "3b24d7-1181-d3dddec0"\r\n
Connection: Keep-Alive, TE\r\n
TE: deflate, gzip, chunked, identity, trailers\r\n
\r\n
```

Figure 4: No anonymization

```
GET /img/progress.jpg HTTP/1.1\r\n
User-Agent: Opera/9.00 (X11; Linux i686; U; en)\r\n
Host: www.foofoofoo.bar\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0\r\n
Referer: http://www.foofoofoofoofoofoofoofoofoofoo.bar\r\n
If-Modified-Since: Mon, 27 Feb 2006 14:50:11 GMT\r\n
If-None-Match: "3b24d7-1181-d3dddec0"\r\n
Connection: Keep-Alive, TE\r\n
TE: deflate, gzip, chunked, identity, trailers\r\n
\r\n
```

Figure 5: Weak anonymization

```
GET /img/progress.jpg HTTP/1.1\r\n
User-Agent: Opera/9.00 (X11; Linux i686; U; en)\r\n
Host: www.foofoofoof.bar\r\n
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1\r\n
Accept-Language: lllllllllllll\r\n
Accept-Encoding: deflate, gzip, x-gzip, identity, *,q=0\r\n
Referer: http://www.foofoofoofoofoofoofoofoo.bar\r\n
If-Modified-Since: Mon, 27 Feb 2006 14:50:11 GMT\r\n
If-None-Match: ifnonematchifnonematch\r\n
Connection: Keep-Alive, TE\r\n
TE: deflate, gzip, chunked, identity, trailers\r\n
\r\n
```

Figure 6: Strong anonymization

```
GET /nnnnnnnnnnnnnnnnnnnn HTTP/1.1\r\n
User-Agent: browserbrowserbrowserbrowserbrowser\r\n
Host: www.foofoofoof.bar\r\n
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1\r\n
Accept-Language: lllllllllllll\r\n
Accept-Encoding: deflate, gzip, x-gzip, identity, *,q=0\r\n
Referer: http://www.foofoofoofoofoofoofoofoo.bar\r\n
If-Modified-Since: Mon, 27 Feb 2006 14:50:11 GMT\r\n
If-None-Match: ifnonematchifnonematch\r\n
Connection: Keep-Alive, TE\r\n
TE: deflate, gzip, chunked, identity, trailers\r\n
\r\n
```

Figure 7: Strongest anonymization


```
GET /img/progress.jpg HTTP/1.1\r\n
User-Agent: Opera/9.00 (X11; Linux i686; U; en)\r\n
Host: www.foofoofoof.bar\r\n
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1\r\n
Accept-Language: no-bok,en;q=0.9\r\n
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0\r\n
Referer: http://www.foofoofoofoofoofoofoofoo.bar\r\n
If-Modified-Since: Mon, 27 Feb 2006 14:50:11 GMT\r\n
If-None-Match: ifnonematchifnonematch\r\n
Connection: Keep-Alive, TE\r\n
TE: deflate, gzip, chunked, identity, trailers\r\n
\r\n
```

Figure 10: Strong URI-anonymization + all customizable headers, except Accept-Language and User-Agent anonymized

compared with Figure 7, these figures show, as expected, exactly the same. The two schemes are anonymizing the same fields.

The last example is depicted in Figure 10. This is the result when the Customized anonymization scheme using the Strong Request-URI anonymization scheme and all *Must*, *Should*, and *Could* headers with the exception of *Accept-Language* and *User-Agent* are anonymized. This is applied by the operator by selecting fields 2 and 11 when choosing fields to retain in clear. When comparing Figure 10 to Figure 7, the differing fields are the *Request-URI*, *Accept-Language* and *User-Agent*, which is exactly what to expect according to the methodology and implementation.

4.3 Expectations

As mentioned, Snort is used to detect attacks during the experiments. During devising the methodology, Snort rules[42] were inspected to find out what information triggered an alarm. The distribution of header fields used in HTTP-related Snort rules are listed in Table 1. The majority of HTTP-related Snort rules trigger on parts of the URI. Table 1 also shows the distribution of URI-levels in Snort rules. Here will for instance "URI 2-level" mean that Snort uses the last two levels of the directory path to detect attacks (e.g. /etc/passwd).

The test data set provided by the IT department at Gjøvik University College includes only three HTTP-related attacks. These attacks were all portscans on port 80. Because of this Nessus was used to generate attacks. These attacks were inserted into the data set.

Note that when deriving the numbers predicted in this subsection, a precondition is made that Snort detects all attacks generated by Nessus. The numbers may not hold if Snort fails to detect some attacks.

HTTP header fields

Table 1 shows that HTTP header fields are not heavily used in detecting attacks. For a total of 1206 rules, only 43 header fields are present in the signatures. Only some of these fields (superscript g) are subject to anonymization according to the methodology. Table 2 summarizes which fields are both present in Snort rules and at the same time subject to anonymization.

It is expected that if these fields originally contain information triggering an alert, and the fields are anonymized, the attacks will not be retained in the anonymized data set. This will cause a drop in the number of positives reported by Snort when anonymization is applied to these fields.

The HTTP header fields present in the Nessus attacks are enumerated in the last column of Table 1. Here we see that only the header fields *Authorization*, *Content-Type* and *User-Agent* are used both by Snort and Nessus and are subject to anonymization (superscript h). To find out if these attacks would trigger any Snort rules the following questions had to be answered:

1. Can the *Authorization* values generated by Nessus trigger an alert based on a Snort rule? The answer to this question is no. The Snort rules containing *Authorization* in *web-iis.rules* and *web-misc.rules* do not match any *Authorization* value generated by Nessus.
2. Can the *Content-Type* values generated by Nessus trigger an alert based on a Snort rule? The answer to this question is no. The Snort rules containing *Content-Type* in

Header fields ^a	Snort rule files																	Nessus
	<i>attack-responses</i>	<i>backdoor</i>	<i>dos</i>	<i>info</i>	<i>multimedia</i>	<i>p2p</i>	<i>porn</i>	<i>scan</i>	<i>virus</i>	<i>web-cgi</i>	<i>web-client</i>	<i>web-coldfusion</i>	<i>web-frontpage</i>	<i>web-iis</i>	<i>web-misc</i>	<i>web-php</i>	<i>Total</i>	
# of related rules	6	1	1	1	5	3	21	1	2	353	173	35	35	119	323	127	1206	-
URI 1-level ^{b,f}		1			5	3			2	324	5	4	13	62	183	111	713	254
URI 2-level ^{c,f}										23		3	21	20	61	8	136	1177
URI 3-level ^{d,f}										2		11	1	13	14	4	45	132
>URI 4-level ^{e,f}										4		6		2	4		16	188
Accept														1			1	1736
Accept-Charset ^f																		1726
Accept-Encoding																		3
Accept-Language ^t																		1729
Authorization ^f														2 ^g	5 ^g		7 ^g	2 ^h
Connection																		1733
Content-Disposition ^f											2 ^g					1 ^g	3 ^g	
Content-Length													1	2			3	8
Content-Type ^f				1 ^g							11 ^g		2 ^g				14 ^g	8 ^h
Cookie														1			1	10
Host ^f													2				2	1747
Location ^f											1 ^g						1 ^g	
Pragma																		1728
Referer																		4
Transfer-Encoding														2	1		3	
Translate														2			2	1
User-Agent ^f						3 ^g									3 ^g		6 ^g	1742 ^h

^a The table shows only those fields occurring in Snort rule files and/or Nessus generated attacks.

^b Attack signature occurs after last "/"

^c Attack signature occurs after second to last "/"

^d Attack signature occurs after third to last "/"

^e Attack signature occurs after fourth to last "/" or more

^f These fields are anonymized in one or more of the anonymization schemes

^g Used by Snort AND subject to anonymization

^h Used by Snort AND subject to anonymization AND used by Nessus

Table 1: Header fields and part of URI used in Snort rules and Nessus attacks

Header field	Occurrences in Snort rules	Anonymization	Snort + Nessus + Anonymization
Authorization:	7	<i>Must</i>	*
Content-Disposition:	3	<i>Must</i>	
Content-Type:	14	<i>Should</i>	*
Location:	1	<i>Must</i>	
User-Agent:	6	<i>Could</i>	*
Total:	31		3 header fields

Table 2: Fields subject to anonymization and occurring in Snort rules and Nessus attacks

`info.rules`, `web-client.rules` and `web-iis.rules` do not match any `Content-Type` value generated by Nessus.

- Can the `User-Agent` values generated by Nessus trigger an alert based on a Snort rule? The answer to this question is no. The Snort rules containing `User-Agent` in `p2p.rules` and `web-misc.rules` do not match any `User-Agent` value generated by Nessus.

Since the answer is no to these questions, anonymization of the fields will not affect the number of positives. Note that these answers are related to this particular data set. For instance, the attacks generated by Nessus contain mostly the value "Mozilla/4.75 [en] (X11; U; Nessus)", a value, which Snort through the default rule set does not trigger on. If the value were recognized as an attack by Snort, the attack would be rendered ineffective by the anonymization process, leading to a lower number of positives.

Conclusions

The conclusions regarding anonymization of HTTP headers are:

- For the current data set, anonymization of any field will not result in lower number of positives.
- The number of header fields used in Snort rules is so low that the influence on the number of positives should be insignificant for any data set recorded from any network (at least in comparison to anonymizing the request-URI).

HTTP Request-URI

Table 1 also shows that most Snort rules trigger on the `Request-URI`. Of 1206 HTTP-related rules, 910 rules trigger on the `Request-URI`. Note that some of the rules trigger both on `Request-URI` and one or several header fields. However, this is not shown in the table. 1-level URIs, where the signature is present in the last part of the URI-path, has a significantly higher representation than URIs with more levels. E.g. 1-level URIs are used in 713 rules in comparison to 136 rules containing 2-level URIs.

The numbers for each URI-level present in the Nessus attacks are listed in the last column of Table 1. Contrary to the Snort rules, the 2-level URIs have a significantly higher representation in the Nessus attacks.

To analyze the influence anonymization has on the attacks when looking at the `Request-URI`, a summary of the methodology for anonymization of the `Request-URI` is given:

Must: The domain part (if present, e.g. `www.mydomain.com`) of the `Request-URI` is always anonymized. *Must* is implemented in Anonymator as Weak `Request-URI`

anonymization.

Should: The Request-URI is anonymized until the second to last "/" (e.g. until /etc/passwd). *Should+Must* is implemented in Anonymator as Strong Request-URI anonymization.

Could: The entire Request-URI is anonymized. *Could+Should+Must* is implemented in Anonymator as Strongest Request-URI anonymization.

For a more thorough discussion of Request-URI anonymization, see Section 3.4.1.

When applying the Weak anonymization scheme, only the domain part of the Request-URI is anonymized. No specific domains are used in the default Snort ruleset. Because of this, the number of positives should be the same for a dataset anonymized with Weak URI anonymization as for the non-anonymized dataset.

Based on the URI-level counts in Table 1 Nessus generated 1751 URIs. Of these 1431 are of level 1 or 2. This is about 81% of the complete URI set. In cases where a script tag occurs in a URI, it appears after the path-part of the Request-URI. This means that also cross site scripting attacks are preserved.

Still based on the URI-level counts in Table 1, one should believe that no attacks would be preserved using the Strongest scheme for Request-URI-anonymization. Note however that the methodology treats cross site scripting and /etc/passwd attacks specifically. When counting the amount of such attacks, Snort should detect 221 attacks, which represents about 13% of the attacks. This might seem like a very low number. However, due to the strict legal requirements for handling sensitive information for identifiable subjects in many countries, a scheme where the entire URI is anonymized is necessary to be included in anonymizing software.

As mentioned in the former subsection, these numbers relate only to the present data set. If, however, another data set has the same distribution of attacks (however an unlikely situation) the numbers should be about the same. It could also occur that a data set has no script- or /etc/passwd-attacks. In such a case the number of positives could be significantly lower.

Conclusions

The conclusions regarding anonymization of the Request-URI are:

- For the current data set, using the Weak Request-URI anonymization scheme should retain about 100% of the attacks.
- For the current data set, using the Strong Request-URI anonymization scheme should retain about 81% of the attacks.
- For the current data set, using the Strongest Request-URI anonymization scheme should retain about 13% of the attacks.
- For other data sets with other attack distributions, using the Weak anonymization scheme, should retain about 100% of the attacks.
- For other data sets with other attack distributions, using the Strong or Strongest anonymization schemes could produce other numbers.

4.4 Infrastructure and resources

4.4.1 Preparations

Four machines were set up with RFC1918[57] addresses. All machines had Ubuntu 5.10 Breezy installed with kernel 2-6-12-10. Ethereal 0.10.12 was also installed. Additional programs are listed as follows:

- PC1, IP:192.168.0.2, tcpreplay, Nessus 2.2.4-2 with updated plugins as of 28th of March 2006. This PC was used for creating HTTP attacks to mix with real network traffic. It was also used for replaying tcpdump files.
- PC2, IP:192.168.0.3. This PC was used for Ethereal sniffing.
- PC3, IP:192.168.0.4, Snort 2.3.2-5 with rule set 2.3 as of 8th of March 2006, containing 4409 rules. This PC was used for intrusion detection to record the number of positives.
- PC4, IP:192.168.0.5, Apache httpd 2.0.54-5ubuntu4 webserver, Mysql 4.1.12-1↔ ubuntu3.1 database and PHP 5.0.5-2ubuntu1.2 and tcpreplay. This PC was set up as a web server implementing PHP and MySQL functionality. It was also used for replaying tcpdump files. The attacks generated by Nessus at PC1 were directed at PC4.

The infrastructure can be viewed in Figure 11.

Real test data was gathered in the format of a tcpdump file from the main router of the wireless network of Gjøvik University College[56] (for Non Disclosure Agreement, see Appendix C). Attacks were collected by running Nessus with a collection of web related attacks. The plugin categories used in Nessus were:

- CGI Abuses
- CGI Abuses: XSS
- Gain a shell remotely
- Gain root remotely
- General

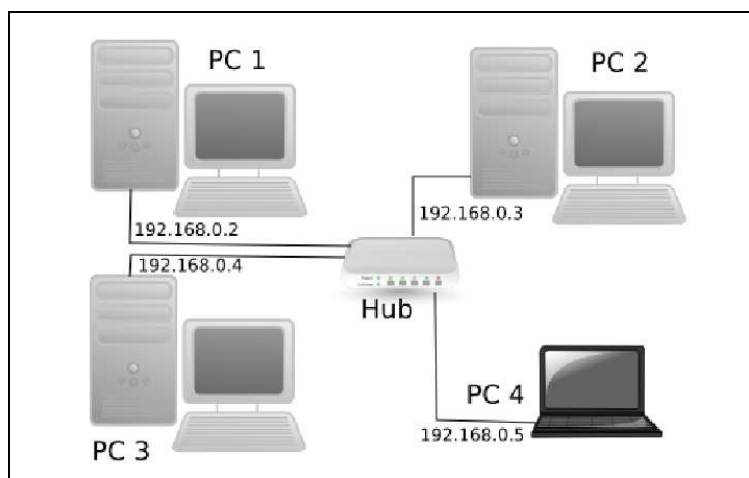


Figure 11: Experimental infrastructure

- Misc
- Web Servers

Altogether 1751 attacks were generated. Using Nessus to generate attacks for use in a thesis handling the topic of realistic data sets might seem contradictory. The author acknowledges this, but had to choose such a solution because the data set provided by Gjøvik University College contained only 3 attacks, all of them being portscans on port 80. More attacks were needed to conduct the experiments, and Nessus was chosen to provide the attacks.

Since Anonymator is a prototype handling only HTTP traffic, this type of traffic was extracted from the routerdump and attackdump. The two tcpdump files were replayed simultaneously to generate a testfile containing both HTTP traffic and HTTP related attacks. It could be argued that for the experiments it would be necessary just to replay the Nessus attacks. However, it is also important to see how the prototype affects real network traffic in order to evaluate if the traffic retains its reality.

The open source IDS Snort was used to give an indication of how many attacks were retained in the data sets when different anonymization schemes were applied. A preparation experiment showed that Snort reported 587 positives when replaying the non-anonymized attackdump. The positives could without further ado be classified as true positives since these were generated by Nessus. However, a preliminary test showed fewer true positives. A basic tuning of Snort, among else enabling the HTTP-inspect pre-processor, resulted in the number of 587 true positives. It is noted that this number is significantly lower than the actual number of attacks generated by Nessus. Because of this, the expectations discussed in Section 4.3 had to be adjusted. See Section 4.5 for this. Due to time constraints, Snort was not tuned further to produce a higher number of true positives. The HTTP routerdump was also replayed to see if Snort detected any attacks in this data set. Snort reported 3 alerts. After inspecting the logs, these alerts could be classified as true positives, being portscans on port 80.

Note that the the number of positives reported by Snort gives just an indication of how Anonymator affects the number of attacks in a data set. If time had not been a factor, the data sets should have been manually examined to count the real number of attacks retained after anonymization. With 10 experiments, each consisting of 1751 attacks, this is considered a much too laborious undertaking. Snort is therefore used as an indicator of the number of retained attacks in the different anonymized data sets.

4.5 Revised expectations

As noted, Snort reports only 587 true positives when replaying the Nessus attacks. In Section 4.3 it was concluded that for the present attacks, anonymization of header fields should have no influence on the number of positives. The conclusions presented on page 42 still hold. The difference, represented by the 1164 missing attacks, is caused by non-matching Request-URIs. If the same distribution between the different URI-levels as in the original attacks are used, the numbers presented in Table 3 should be used when deducing the expected experimental results.

Revised expectations for Weak-related anonymization schemes

The expectations from Section 4.3 still hold with the same arguments. About 100% of the attacks should be retained in the anonymized data sets. The cause for this percentage is

URI-level	Original	Distribution	Revised
1	254	14.51%	85
2	1177	67.22%	395
4	132	7.54%	44
>4	188	10.74%	63
Total	1751	≈ 100%	587

Table 3: Nessus-generated URI-levels for revised expectations

that only the domain part of the Request-URI is anonymized, leaving the parts matching Snort rules in clear.

Revised expectations for Strong-related anonymization schemes

Recall from Section 3.4.1 and 4.2.1 that the Strong Request-URI anonymization anonymizes the URI until the second to last "/". Also, if Anonymator finds signs of a cross site scripting attempt, it anonymizes the URI until the script tag. According to Table 3 the total number of URIs causing true positives is reduced to 587. Adding the 3 port scans gives a total of 590 attacks. When keeping the distribution between attacks, level 1 and 2 count for 480 attacks, representing 81.4% of the attacks. This means that 81.4% of the attacks should be preserved using Strong-related anonymization schemes.

Revised expectations for Strongest-related anonymization schemes

In the Strongest-related anonymization schemes, the whole URI is anonymized. However, cross site scripting- and /etc/passwd-attacks are still preserved. As mentioned in Section 4.3 these attacks count 221, now representing about 37,5% of the attacks. This means that 37.5% of the attacks should be preserved using Strongest-related anonymization schemes.

Conclusions for revised expectations

The revised conclusions regarding anonymization of the Request-URI are:

- For the current data set, using the Weak Request-URI anonymization scheme should retain about 100% of the attacks.
- For the current data set, using the Strong Request-URI anonymization scheme should retain about 81.4% of the attacks.
- For the current data set, using the Strongest Request-URI anonymization scheme should retain about 37.5% of the attacks.
- For other data sets with other attack distributions, using the Weak anonymization scheme, should retain about 100% of the attacks.
- For other data sets with other attack distributions, using the Strong or Strongest anonymization schemes could produce other numbers.

4.6 Experiments

Ten experiments have been conducted to provide material for studying the operation of the methodology and the prototype. A summary of the results is given in Table 4.

4.6.1 Exp. 1: No anonymization

The result from this experiment will serve as a basis for interpreting the results of the subsequent experiments. The non-anonymized test file was replayed. Snort reported 590

positives for this experiment. This shows that all 587 positives from the preliminary experiment replaying the attackdump and the 3 positives from the preliminary experiment replaying the routerdump were detected by Snort.

4.6.2 Exp. 2: Weak scheme

This experiment was conducted to see how the Weak anonymization scheme influenced the number of positives. The Weak anonymization scheme implements the *Must* class of the methodology, that is, all headers classified as *Must*, and the domain part of the Request-URI are anonymized. The non-anonymized test file was fed to Anonymator and the Weak anonymization scheme was applied. The resulting anonymized file was replayed. In this experiment Snort reported 590 positives.

4.6.3 Exp. 3: Strong scheme

This experiment was conducted to see how the Strong anonymization scheme influenced the number of positives. The Strong anonymization scheme implements the *Must* and *Should* classes of the methodology. The Request-URI is anonymized until the next to last "/" to retain the many 2-level path attacks. If the Request-URI contains a script tag, the URI is anonymized until this occurrence. The non-anonymized test file was fed to Anonymator and the Strong anonymization scheme was applied. The resulting anonymized file was replayed. In this experiment Snort reported 472 positives.

4.6.4 Exp. 4: Strongest scheme

This experiment was conducted to see how the Strongest anonymization scheme influenced the number of positives. The Strongest anonymization scheme implements the *Must*, *Should* and *Could* classes of the methodology. The whole Request-URI is anonymized except if the URI contained a script tag or an /etc/passwd-attack. In these cases the URI was anonymized until these occurrences. The non-anonymized test file was fed to Anonymator and the Strongest anonymization scheme was applied. The resulting anonymized file was replayed. In this experiment Snort reported 246 positives.

4.6.5 Exp. 5: Customized: Weak URI, no headers anonymized

This experiment was conducted to see how the Customizable anonymization scheme influenced the number of positives when Weak Request-URI anonymization was applied in addition to leaving all customizable header fields in clear. The non-anonymized test file was fed to Anonymator and the mentioned Customizable anonymization scheme was applied. All customizable headers were marked. The filter-in methodology leaves all marked header fields in clear. The resulting anonymized file was replayed. In this experiment Snort reported 590 positives.

4.6.6 Exp. 6: Customized: Strong URI, no headers anonymized

This experiment was conducted to see how the Customizable anonymization scheme influenced the number of positives when Strong Request-URI anonymization was applied in addition to leaving all customizable header fields in clear. The non-anonymized test file was fed to Anonymator and the mentioned Customizable anonymization scheme was applied. All customizable headers were marked, leaving all customizable header fields in clear. The resulting anonymized file was replayed. In this experiment Snort reported 472 positives.

4.6.7 Exp. 7: Customized: Strongest URI, no headers anonymized

This experiment was conducted to see how the Customizable anonymization scheme influenced the number of positives when Strongest Request-URI anonymization was applied in addition to leaving all customizable header fields in clear. The non-anonymized test file was fed to Anonymator and the mentioned Customizable anonymization scheme was applied. All customizable headers were marked, leaving all customizable header fields in clear. The resulting anonymized file was replayed. In this experiment Snort reported 246 positives.

4.6.8 Exp. 8: Customized: Weak URI, all headers anonymized

This experiment was conducted to see how the Customizable anonymization scheme influenced the number of positives when Weak Request-URI anonymization was applied in addition to anonymization of all customizable header fields. The non-anonymized test file was fed to Anonymator and the mentioned Customizable anonymization scheme was applied. No customizable headers were marked. The filter-in methodology will anonymize all non-marked header fields. The resulting anonymized file was replayed. In this experiment Snort reported 590 positives.

4.6.9 Exp. 9: Customized: Strong URI, all headers anonymized

This experiment was conducted to see how the Customizable anonymization scheme influenced the number of positives when Strong Request-URI anonymization was applied in addition to anonymization of all customizable header fields. The non-anonymized test file was fed to Anonymator and the mentioned Customizable anonymization scheme was applied. No customizable headers were marked, resulting in anonymization of all customizable header fields. The resulting anonymized file was replayed. In this experiment Snort reported 472 positives.

4.6.10 Exp. 10: Customized: Strongest URI, all headers anonymized

This experiment was conducted to see how the Customizable anonymization scheme influenced the number of positives when Strongest Request-URI anonymization was applied in addition to anonymization of all customizable header fields. The non-anonymized test file was fed to Anonymator and the mentioned Customizable anonymization scheme was applied. No customizable headers were marked, resulting in anonymization of all customizable header fields. The resulting anonymized file was replayed. In this experiment Snort reported 246 positives.

4.7 Results

Table 4 gives a summary of the results from the experiments in the form of total numbers of positives reported by Snort. Table 9 listed in Appendix B gives a more detailed view over the distribution of attacks detected by Snort.

The results are produced by replaying a data set with traffic recorded at the wireless LAN at Gjøvik University College. Attacks generated by Nessus are mixed into the data set. The results mentioned in this section relate only to this data set. Other data sets may produce different results. However, the present results will give an indication of the numbers to be expected and the relationship between the results. This section will therefore explain thoroughly the differences between the test results. This is done first of all to evaluate the methodology and the prototype implementation. Secondly, it will

Experiment	Positives
Exp.1: Raw dataset	590
Exp 2: Weak anonymization	590
Exp 3: Strong anonymization	472
Exp 4: Strongest anonymization	246
Exp 5: Customized/Weak URI, no headers ^a	590
Exp 6: Customized/Strong URI, no headers	472
Exp 7: Customized/Strongest URI, no headers	246
Exp 8: Customized/Weak URI, all headers ^b	590
Exp 9: Customized/Strong URI, all headers	472
Exp 10: Customized/Strongest URI, all headers	246

^a "No" means *Should* and *Could* headers are kept in clear. *Must* headers are always anonymized.

^b "All" means *Should* and *Could* headers are anonymized. *Must* headers are always anonymized.

Table 4: Number of positives reported by Snort

give other users of the methodology and users of Anonymator, or other anonymization software, directions for what to look for when interpreting future test results.

4.7.1 Weak related schemes

As can be seen from Table 4, Snort reports the same number of positives in experiments 1, 2, 5 and 8, where the last three being Weak related anonymization schemes. Table 9 listed in Appendix B shows almost the same numbers for each attack type for the four experiments. The only exception is the number reported for SID 1147r7 for experiment 8, compared to experiments 1, 2 and 5. In experiment 8, Snort logs 15 alerts for SID 1147 in its alert logfile. In experiments 1, 2 and 5 Snort logs 12 alerts in its alert logfile. Although Snort logs three more alerts for SID 1147r7 in experiment 8, Snort still reports a total of 590 alerts for experiments 1, 2, 5 and 8. When looking at the attacks containing the signature related to SID 1147r7, "cat%20", there are 15 requests containing this pattern. After comparing the alert logfiles for experiments 1, 2, 5 and 8 these requests trigger SID 1147r7:

These are found for experiment 1, 2, 5 and 8

```
GET /cgi-local/eshop.pl/seite=;cat%20eshop.pl|
GET /cgi-bin/eshop.pl/seite=;cat%20eshop.pl|
GET /scripts/eshop.pl/seite=;cat%20eshop.pl|
GET /eshop.pl/seite=;cat%20eshop.pl|
GET /cgi-bin/php-ping.php?count=1+%26+cat%20/etc/passwd+%26&submit=Ping%21
GET /scripts/php-ping.php?count=1+%26+cat%20/etc/passwd+%26&submit=Ping%21
GET /php-ping.php?count=1+%26+cat%20/etc/passwd+%26&submit=Ping%21
GET /cgi-local/shop.pl/page=;cat%20shop.pl|
GET /cgi_bin/shop.pl/page=;cat%20shop.pl|
GET /cgi-bin/shop.pl/page=;cat%20shop.pl|
GET /scripts/shop.pl/page=;cat%20shop.pl|
GET /shop.pl/page=;cat%20shop.pl|
```

Additional for experiment 8:

```
GET /cgi-bin/directory.php?dir=%3Bcat%20/etc/passwd
GET /scripts/directory.php?dir=%3Bcat%20/etc/passwd
GET /directory.php?dir=%3Bcat%20/etc/passwd
```

Snort rule SID 1147r7:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC cat%20 access"; flow:to_server,established;
content:"cat%20"; nocase; reference:bugtraq,374; reference:
cve,1999-0039; classtype:attempted-recon; sid:1147; rev:7;)
```

The three requests not being detected in experiments 1, 2 and 5 use the hexadecimal representation "%3B" of the ";" character. In some way Snort fails to recognize the pattern "cat%20" when the URI contains "%3B". The only difference between the experiments is that all *Must*, *Should* and *Could* header fields are anonymized in experiment 8. The Request-URI is however unchanged. This strange occurrence has not been investigated further. A possible explanation could be that there is a bug in Snort. This explanation may be supported by the fact that the counts of positives are as shown in Table 9 in Appendix B. Snort counts three more positives for SID 1147r7 in experiment 8, but still keeps the total number of 590 positives.

Experiment 1 provides the basis for all comparisons, applying no anonymization at all. The following are the similarities and differences between these experiments:

- Experiments 2, 5 and 8 use the Weak anonymization scheme for the Request-URI.
- Experiments 2, 5 and 8 anonymize *Must* headers.
- Experiment 8 anonymizes all *Should* and *Could* headers.
- Experiments 2 and 5 do not anonymize *Should* and *Could* headers.

These results indicate that using Weak anonymization of the Request-URI has no influence on the number of positives. This is in accordance with the conclusions for the revised expectations listed in Section 4.5. There are also equal results for applying anonymization to all *Must*, *Should* and *Could* headers (experiment 8) as to only applying anonymization to *Must* headers (experiments 2 and 5). Also, since the results for the Weak related anonymization schemes are equal to the result using the non-anonymized data set, it can be concluded that anonymization of the header fields have no influence on the number of positives. This is in accordance with the predictions listed in Sections 4.3 and 4.5. The prediction for the Weak related schemes was that about 100% of the attacks would be retained. This prediction seems to hold.

Conclusions

The conclusions based on the Weak related anonymization schemes are:

Conclusion 1 The Weak anonymization scheme and the Customizable anonymization scheme applying Weak Request-URI anonymization retain all attacks in the data set.

Conclusion 2 Anonymization of *Must*, *Should* and *Could* header fields have no influence on the number of positives.

4.7.2 Strong related schemes

As can be seen from Table 4, Snort reports the same number of positives in experiments 3, 6 and 9. Table 9 listed in Appendix B also shows the same numbers for each attack

Type	Occurrences	Explanation	Rules failing (# failing)
Script-tag	30	Script tag makes Anonymator anonymize everything until the tag	1875r5(1), 1998r4(3), 2208r5(3), 1406311(1), 1997r4(1), 1592r6(1), 2410r2(3), 1637r8(3), 1534r8(1), 1602r9(3), 2326r4(3), 882r5(3), 1654r7(4)
3-level signature	3	Snort rule use 3-level URI	1827r7(1), 1829r5(1), 908r9(1)
/	78	Several "/" occurring after the content Snort is looking for	2565r1(3), 3465r2(3), 1969r4(3), 901r10(3), 1396r9(3), 1395r9(3), 1815r5(3), 1590r7(1), 3463r2(6), 1591r6(1), 900r11(3), 1816r3(3), 1106r11(3), 1522r10(8), 1523r10(8), 2060r1(1), 2484r1(1), 2002r5(4), 2328r3(3), 1862r7(6), 1300r7(3), 2366r4(3), 1301r11(3)
%3-flaw	5	Anonymator searches for "%3", when it should search for "%3c" or "%3C", both being hex for "<"	1376r5(5)
http_inspect	2	2 attacks slipped by Snort as a result of anonymization	119:18:01
Total	118	Total amount of attacks rendered ineffective	

Table 5: Information causing drop in number of positives for Strong related schemes

type for the three experiments. Numbers are about as predicted since 472 is 80% of the 590 attacks from experiment 1. The prediction was about 81.4%.

Table 5 shows the types of information causing the lower number of positives for the Strong related schemes. When Anonymator finds a script tag, indicated by "<", "%3c" or "%3C" it anonymizes the entire Request-URI until the tag. The following shows such a situation:

Attack:

```
GET /sgynamo.exe?HTNAME=<script>foo
```

Snort rule:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
```

```
(msg:"WEB-IIS sgdynamo.exe access"; flow:to_server,established; uricontent:
"/sgdynamo.exe"; nocase; reference:bugtraq,4720; reference:cve,2002-0375;
reference:nessus,11955; classtype:web-application-activity; sid:2326; rev:4;)
```

Here Anonymator anonymizes `"/sgdynamo.exe?HTNAME="`, retaining `"<script>foo"`. The Snort rule looks for `"/sgdynamo.exe"`. Since this part is anonymized, the rule is not triggered. Such cases cause 30 fewer positives reported by Snort.

Some Snort rules use 3-level URIs or more to look for attack patterns. Since the Strong related anonymization schemes anonymizes everything until and including the third level, these attacks are rendered ineffective. Here is an example of such a rule:

Snort rule::

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC Tomcat TroubleShooter servlet access"; flow: established,
to_server; uricontent:"/examples/servlet/TroubleShooter"; reference:
bugtraq,4575; reference:nessus,11046; classtype:web-application-activity;
sid:1829; rev:5;)
```

In this example Anonymator will anonymize "examples", but leave the leading "/" and "/servlet/TroubleShooter" intact, resulting in this request line: `"GET /nnnnnnnn/↔servlet/TroubleShooter HTTP/1.1"`. Since this is a pattern Snort has no rule to recognize, Snort will not report a true positive. Also, the attack will be rendered ineffective. Such cases cause a drop of 3 in the number of positives.

The largest influence on the number of positives is caused by several "/" occurring after what represents the attack signature. One such example is:

Attack:

```
GET /cgi-bin/mrtg.cgi?cfg=../../../../../../../../winnt/win.ini
```

Snort rule:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-CGI mrtg.cgi directory traversal attempt"; flow:to_server,established;
uricontent:"/mrtg.cgi"; content:"cfg=../"; reference:bugtraq,4017; reference:
cve,2002-0232; reference:nessus,11001; classtype:web-application-attack;
sid:1862; rev:7;)
```

In this example the attack signature is `"/mrtg.cgi"` and `"cfg=../"`. Since Anonymator in the Strong Request-URI anonymization anonymizes until the second to last "/", just `"/winnt/win.ini"` remains in the clear. This renders the attack ineffective. Such cases cause 78 fewer positives reported by Snort.

Snort rule 1376r5 revealed a flaw in Anonymator:

Snort rule:

4.8 Conclusions regarding experimental work

The conclusions based on the experiments are listed in Table 7. The first thing to notice is that the anonymization schemes work almost as expected. The Weak related schemes retain all attacks in the data sets. The Strongest related anonymization schemes work almost opposite to this, rendering all attacks ineffective, with the exception of those being specifically treated. The Strong related schemes work almost as expected, anonymizing until the second to last "/" of the Request-URI. This leads in some cases to anonymization of the 2-level URI reflected in a Snort rule when e.g. a path is provided as parameter to the Request-URI.

Secondly, anonymization of header fields has no influence on the number of positives. Note that this pertains only to the present test data set. Other data sets having header field values matched by a Snort rule will trigger an alert. Those headers present in the current data set and which Snort triggers on have values not subject to anonymization.

Third, some values are treated specifically by Anonymator. For the current version of Anonymator the specific strings are: "<", "%3", "/etc/passwd" and "login=0". Especially the first three are responsible for retaining a large amount of attacks when the Strong and Strongest related schemes are applied. The "%3" string represents a flaw in Anonymator, which will be corrected in a newer version of Anonymator.

Fourth, two additional flaws were found during testing. The only directory path separator character implemented in Anonymator is "/". Some requests, mostly related to Windows path traversal, use the "\" character. This is also put on the to-do list for further development of Anonymator. The other flaw is that Request-URIs starting with another character than "/" or "h" are not anonymized at all. This is also put on the to-do list.

The last thing to draw attention to is when applying anonymization to a request being an attack, the attack may be rendered ineffective while still keeping the information Snort is looking for. This might for example be an /etc/passwd-attack where the path before "/etc/passwd" is anonymized. This will still trigger a positive by Snort. However, instead of being a true positive, this must be considered a false positive. The reason for anonymizing this way is to prioritize anonymity in the trade-off between anonymity and reality.

	Conclusion
Conclusion 1	The Weak anonymization scheme and the Customizable anonymization scheme applying Weak Request-URI anonymization retain all attacks in the data set.
Conclusion 2	Anonymization of <i>Must</i> , <i>Should</i> and <i>Could</i> header fields have no influence on the number of positives.
Conclusion 3	Cross site scripting attacks are retained, but will cause signature patterns preceding the script tag to be altered.
Conclusion 4	2-level signatures are retained and work as expected. More levels will be anonymized.
Conclusion 5	Attacks having more than two slashes ("/") after the pattern searched for by Snort are rendered ineffective.
Conclusion 6	A "%3"-flaw is discovered and put on the to-do list.
Conclusion 7	A backslash-flaw, where "\" is used for directory-separation, is discovered. This is put on the to-do list.
Conclusion 8	Anonymization of header fields has no influence on the number of positives when Strong related anonymization schemes are used.

Continued on next page

	Conclusion
Conclusion 9	Anonymization of the entire Request-URI has significant influence on the number of positives.
Conclusion 10	When anonymizing until "<", "%3c", "%3C" and "/etc/passwd" positives are still reported. However, the positives must be considered false since the path is anonymized, rendering the attack ineffective.
Conclusion 11	A leading character flaw is found. When a Request-URI starts with other characters than "/" or "h" the Request-URI is kept as is. This flaw is put on the to-do list for Anonymator's further development.
Conclusion 12	Anonymization of header fields has no influence on the number of positives when Strongest related anonymization schemes are used.

Table 7: Conclusions for experimental work

5 Conclusions

This thesis has presented a methodology for anonymization of payload data present in recorded traffic. Such anonymized data sets can freely be distributed for use in evaluations of intrusion detection systems. A prototype called Anonymator based on the methodology is also implemented.

5.1 Methodology

There have been more previous work on network and transport layer anonymization than on application layer anonymization. Anonymity considerations for some network and transport protocols are listed in Sections 3.3.1, 3.3.2 and 3.3.3. These are only superficially considered for the current version of the methodology, and have not been classified according to the classes devised. The methodology presented focuses on HTTP traffic. The author encourages further research to include more application layer protocols to the methodology and also to classify the listed network and transport protocols. There are also more network and transport protocols to be added to the methodology.

A classification system has been devised to classify protocol header fields and other information. The devised classes are:

Must ...be anonymized. Information classified as *Must* has the greatest potential to identify a subject and must be anonymized.

Should ...be anonymized. Information classified as *Should* has medium potential to identify a subject, but are in most cases not a threat to privacy.

Could ...be anonymized. Information classified as *Could* has low potential to identify a subject, and are very seldom a threat to privacy.

No ...anonymization necessary. Information classified as *No* cannot in any way be used to identify a subject.

The information related to a HTTP packet can be divided in request line, response line, header fields and other data. For the request line only the Request-URI is anonymized. The schemes are as follows:

Must The domain part of the URI is anonymized to provide a basic anonymization scheme.

Should The URI is anonymized until the second to last "/" to preserve the many 2-level attacks.

Could The entire URI is anonymized to provide the highest level of anonymity.

The response lines are not anonymized at all since they do not contain information revealing subject's identity.

Each header field is considered carefully and classified as *Must*, *Should*, *Could* or *No*. Table 8 in Appendix A lists the headers defined in [38, 46, 49, 52] with the appropriate

classifications. The header field names are not anonymized, but the options and values are, where appropriate. In some cases only some options are anonymized, leaving other options in clear. This is however not implemented in Anonymator, which in the current version anonymizes all options.

Other data not recognized by the methodology is anonymized entirely. This is data we do not know anything about. Since such data may contain identifying information it must be anonymized.

Due to differing legislation between countries regarding privacy, the methodology leaves much of the decisionmaking to the operator of the anonymizing software. The *Must* headers and domain part of the Request-URI are always anonymized. Anonymization of other information can be chosen by the operator using a filter-in approach, suggested in [13], where the operator chooses the headers to be kept in clear. Other headers will be anonymized. This approach is chosen to minimize accidental revelation of identifying information.

5.2 Prototype

The prototype, called Anonymator, is an implementation of parts of the methodology. Anonymator anonymized only HTTP data. Other application layer protocols and link, network and transport protocols are left as they are. The exception are fields dependent on application layer data, such as the TCP checksum, requiring a recalculation in order to be correct. Other data are copied without further ado to the new packet. It is advised to extract other data than HTTP data with adherent link, network and transport data from the data set before feeding the data to Anonymator. Anonymator may not work correctly for other protocols. This is said with the knowledge that Anonymator is just a prototype, subject to throwing away or to further development.

The classifications are implemented as different anonymization schemes. In Anonymator the operator can choose between these anonymization schemes:

Strongest This scheme implements anonymization to the *Must*, *Should* and *Could* headers and the Request-URI. For the Request-URI this means anonymizing the entire URI. To preserve some positives, Anonymator treats cross site scripting and /etc/passwd attacks specifically. When Anonymator detects patterns adhering to these attacks, it only anonymizes the Request-URI until these occurrences.

Strong This scheme implements anonymization to the *Must* and *Should* headers and the Request-URI. For the Request-URI this means anonymizing the URI until the second to last "/". The reason for this is to preserve the many positives relating to 2-level attack signatures. Also here Anonymator will preserve the signature of cross site scripting attempts.

Weak This scheme implements anonymization to the *Must* headers and the Request-URI. For the Request-URI this means anonymizing only the domain part of the URI.

Customized This scheme leaves the decision to the operator. All *Must* headers are anonymized by default. The operator chooses which fields belonging to the classes *Should* and *Could* should be anonymized. The operator also chooses which Request-URI-anonymization scheme to use.

Some values are treated specifically in Anonymator, due to their frequent use in attacks. Current the specific strings are "<", "%3" (which should have been "%3c and "%3C"), "/etc/passwd" and "login=0" (pertaining to the Cookie header). These strings are fetched from Snort rules after analyzing the attacks. More specific values can be added. Optimally all the Snort content-values should have been included. A way to do this is presented in Chapter 6.

Since Anonymator is a prototype, not all aspects of the methodology are implemented. One aspect not implemented is the anonymization of a subset of options for a header field. An example of this is anonymization of the Via general header field, where only the received-by option needs to be anonymized. The current version anonymizes all options. Further fine-graining of option anonymization is on the to-to list.

5.3 Experiments

The experiments conducted demonstrate correctness of the methodology and the implementation in major part. A thorough analysis of the expected behaviour was done before the experiments. After the preparation of the experiments the expectations were adjusted due to Snort reporting true positives for only about a third of the attacks generated by Nessus. After conducting the experiments, the results were within an acceptable fault tolerance range of the expected results. The main conclusions for the experiments are that the Strongest, Strong, Weak and Customized anonymization schemes work as expected, with the exception of some minor unpredicted behaviours and flaws in Anonymator. One example is the case when applying Strong Request-URI anonymization to an URI having another URI as parameter. In such cases the attack signature present in the start of the URI will be overwritten because Anonymator counts the "/" in the URI being the parameter.

The flaws detected are:

"%3" Anonymator shall search for the strings ">", "%3c" and "%3C" to reveal cross site scripting attacks. A typo makes Anonymator only look for "<" and "%3", leaving out the "c" and "C". The solution will be to include the "c" or "C" after "%3".

"\" Anonymator treats only "/" as the directory path separation character. This might for Windows systems be the character "\". The solution will be to treat "\" the same way as "/".

Leading Request-URI character Leading Request-URI characters other than "/" and "h" leads to keeping the URI in clear, no matter what anonymization scheme is applied. The solution will be to accept all characters as leading characters for the URI.

These flaws are put on the to-do list for the next version of Anonymator.

It is important to note that even if Snort reports a number of positives when replaying an anonymized data set, the positives must not necessarily be true positives. The point is that when Anonymator anonymizes a Request-URI it may anonymize a part of the path, but retain the information triggering an alert. This means that Snort will count a positive. However, the attack itself is rendered ineffective since a part of the path is altered in the anonymization process. The positive must in such a case be considered a false positive. Such situations are occurring most often when applying the Strongest anonymization related schemes. This may counter the goal of keeping the data set as

realistic as possible. However in the trade-off between reality and anonymization, the Strongest anonymization scheme must have focus on anonymization.

5.4 Recommendations

Since there are different legislation rules regarding privacy among countries, it is difficult to give an exact advise of which level of anonymization to choose. It should be noted that this methodology is not finalized, being just in its early stages. More protocols must be added, and link, network and transport layer anonymization must be implemented. To adhere to the legislation in Norway, the Strongest scheme will be the safest choice. However, the Strong choice would in most cases be adequate. If using the Strong scheme, the data set could be manually inspected to find out if any identifying information remains. As stated in [12], one can never be completely sure that any such information remains. The need for methodologies for automatic inspection of data sets in order to reveal unwanted information should be a topic for further research. So far simple scripting could be done to extract information from the anonymized data sets in order to present the information in a more readable format for manual inspection.

The Weak anonymization scheme is not recommended to use in a production environment. This scheme should be regarded as a research anonymization scheme with the property of retaining as many attacks as possible. The anonymization is however not considered good enough to provide unquestionable anonymity.

6 Further Work

This thesis describes a methodology for anonymizing network traffic in order to provide distributable data sets for IDS testing. The limited time available for the thesis made it possible only to focus on a small part of the methodology. To devise a complete and comprehensive methodology further research is necessary. The following sections provide some guidelines for further work, first of all for developing the methodology, but also for the development of Anonymator. Some recommendations regarding experimentations are also included.

6.1 Methodology

This thesis focuses on payload anonymization of network traffic, in particular HTTP anonymization. These are the recommendations for further development of the methodology:

- [40] contains additional HTTP headers not included in this methodology. These headers should be carefully considered and added to the methodology.
- Add more application layer protocols to the methodology. The protocols must be carefully considered regarding information revealing subject's identity. Header information and other information must be classified according to the classes *Must*, *Should*, *Could* and *No* for easier implementation. Ways to anonymize different data must be considered while at the same time keeping in mind the importance of retaining reality for the data set. In the Strongest anonymization schemes the trade-off must favor anonymization.
- Classify and add link, network and transport protocols to the methodology. The same considerations as mentioned in the former item apply also here.
- As noted, many attacks are rendered ineffective even though the anonymization process retains the attack signatures. Research should be conducted for ensuring the attacks to retain their full functionality without endangering privacy.
- This thesis has analyzed Snort rules to find out what information Snort searches for when trying to detect attacks. It would have been interesting to see how other IDSes uses different information to detect attacks.
- The current version of the new methodology focuses on altering information it explicitly can recognize. It would have been interesting to see if heuristics could be applied to the payload of a packet in order to discover more known patterns.

6.2 Prototype

The current version of the prototype implements most of the methodology. However, some functionality present in the devised methodology is not implemented. In addition some flaws were found as a result of the experiments. The prototype should also be developed in parallel with the methodology. These are the recommendations for the

further work regarding the prototype:

- Correct the flaws found during the experiments:
 - The "%3"-flaw should be corrected by searching for "%3c" and "%3C" instead of just "%3".
 - The "\"-flaw should be corrected by implementing "\" as directory path separator in addition to the present "/".
 - The "leading Request-URI character"-flaw should be corrected by allowing any character to be the first in the Request-URI.
- Implement partial header anonymization for those header fields having many options, of which only a subset is subject to anonymization.
- Implement headers defined in [40] after adding them to the methodology.
- Implement additional application layer protocols added to the methodology.
- Implement anonymization of link, network and transport layer information.
- Optimal anonymization combined with the retention of a high number of positives could be accomplished by letting Anonymator build a data structure over the `content-` and `uricontent` values present in Snort rules. Such a technique has been used in the traffic generator Mucus[58, 59]. The data structure could be used for searching data sets for these values.
- Research with the goal of providing automatic checking for unwanted information in anonymized data sets would provide the operator with an additional level of assurance that the data set produced is adequately anonymized.

6.3 Experimental

The experiments conducted for this thesis use Snort to count attacks when differently anonymized data sets are replayed. It would have been interesting to see how other IDSes react on the anonymized data. This might also be used to improve the methodology.

Bibliography

- [1] Tanenbaum, A. S. 2003. *Computer Networks*. Pearson Education International, Vrije Universiteit, Amsterdam, The Netherlands.
- [2] Creswell, J. W. July 2002. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications.
- [3] Leedy, P. & Ormrod, J. E. 2003. *Practical Research : Planning and Design*. Prentice Hall, 8 edition.
- [4] Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., & Zissman, M. A. 2000. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. *disceX*, 02, 1012.
- [5] Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. 2000. The 1999 darpa off-line intrusion detection evaluation. *Comput. Networks*, 34(4), 579–595.
- [6] Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. 2000. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, 162–182, London, UK. Springer-Verlag.
- [7] Durst, R., Champion, T., Witten, B., Miller, E., & Spagnuolo, L. 1999. Testing and evaluating computer intrusion detection systems. *Commun. ACM*, 42(7), 53–61.
- [8] Shipley, G. Dragon claws its way to the top. <http://www.networkcomputing.com/1217/1217f2.html>, last visited 20th June 2006.
- [9] Yocom, B. & Brown, K. October 2001. Intrusion battleground evolves. <http://www.nwfusion.com/reviews/2001/1008bg.html>, last visited 20th June 2006.
- [10] The NSS Group Ltd. August 2003. Intrusion detection systems group test (edition 4). <http://www.nss.co.uk/ids/edition4/index.htm>. New edition of this report is published every year.
- [11] McHugh, J. November 2000. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. 3(4), 262–294.
- [12] Mell, P., Hu, V., Lippmann, R., Haines, J., & Zissman, M. An overview of issues in testing intrusion detection systems. Technical Report NIST IR 7007, National Institute of Standards and Technology, August 2003.
- [13] Pang, R. & Paxson, V. 2003. A high-level programming environment for packet trace anonymization and transformation. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 339–351, New York, NY, USA. ACM Press.

- [14] Pang, R., Allman, M., Paxson, V., & Lee, J. October 2005. The devil and packet trace anonymization. Under submission.
- [15] Verykios, V., Bertino, E., Fovino, I., Provenza, L., Saygin, Y., & Theodoridis, Y. 2004. State-of-the-art in privacy preserving data mining.
- [16] Justis- og politidepartementet. January 2001. Lov om behandling av personopplysninger (lov-2000-04-14-31). <http://lovdata.no/cgi-wift/wiftldles?doc=/usr/www/lovdata/all/nl-20000414-031.html&ddep=alle&kort+,+titt=personopplysningsloven&>, last visited 20th June 2006.
- [17] Clarke, R. Introduction to dataveillance and information privacy, and definitions of terms. Oak Ridge National Laboratory, <http://www.anu.edu.au/people/Roger.Clarke/DV/Intro.html#Priv>, last visited 20th June 2006.
- [18] Tan, C. 2003. Unpublished document. <http://www.cra.org/Activities/craw/dmp/awards/2003/Tan/research/draft1.html>, last visited 20th June 2006.
- [19] Xu, J., Fan, J., Ammar, M., & Moon, S. B. 2001. On the design and performance of prefix-preserving ip traffic trace anonymization. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 263–266, New York, NY, USA. ACM Press.
- [20] Peuhkuri, M. 2001. A method to compress and anonymize packet traces. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 257–261, New York, NY, USA. ACM Press.
- [21] Bro intrusion detection system. Lawrence Berkeley National Laboratory, <http://www.bro-ids.org/>, last visited 20th June 2006.
- [22] Kreibich, C. 2005. Netdude, the hacker's choice. <http://netdude.sourceforge.net/>, last visited 20th June 2006.
- [23] Kreibich, C. 2004. Design and implementation of netdude, a framework for packet trace manipulation (awarded best student paper!). In *USENIX Annual Technical Conference, FREENIX Track*, 63–72.
- [24] Clifton, C., Kantarcioglu, M., & Vaidya, J. 2002. Defining privacy for data mining.
- [25] Snort. <http://www.snort.org/>, last visited 20th June 2006.
- [26] Dubrawsky. May 2002. Portsentry for attack detection. <http://www.securityfocus.com/infocus/1580>, visited 20th June 2006.
- [27] Arkin, O. August 2000. Nmap hackers: Tosing oss out of the window / fingerprinting windows 2000 with icmp. <http://lists.insecure.org/lists/nmap-hackers/2000/Jul-Sep/0037.html>, visited 20th June 2006.
- [28] Arkin, O. October 2000. Nmap hackers: Precedence field value in icmp error messages with linux. <http://lists.insecure.org/lists/nmap-hackers/2000/Oct-Dec/0009.html>, visited 20th June 2006.

- [29] Internet Assigned Numbers Authority. October 2005. Ip tos parameters. <http://www.iana.org/assignments/ip-parameters>, visited 20th June 2006.
- [30] Arkin, O. & Yarochkin, F. August 2001. Phrack volume 11, issue 57. <http://www.phrack.org/show.php?p=57&a=7>, visited 20th June 2006.
- [31] Andreasson, O. 2005. Iptables tutorial 1.2.0, ip headers. <http://iptables-tutorial.frozentux.net/chunkyhtml/x167.html>, visited 20th June 2006.
- [32] Symantec attack signatures. http://securityresponse.symantec.com/avcenter/nis_ids/ last visited 20th June 2006.
- [33] Ptacek, T. H. & Newsham, T. N. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.
- [34] Sun, Q., Simon, D. R., Wang, Y.-M., Russell, W., Padmanabhan, V. N., & Qiu, L. 2002. Statistical identification of encrypted web browsing traffic. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 19, Washington, DC, USA. IEEE Computer Society.
- [35] U.S. Department of Energy. October 1998. Freebsd tcp rst denial of service vulnerability. <http://ciac.llnl.gov/ciac/bulletins/j-008.shtml> last visited 20th June 2006.
- [36] Abdulla, A. What is covert channel and what are some examples? http://www.sans.org/resources/idfaq/covert_chan.php?printer=Y visited 20th June 2006.
- [37] Kohno, T., Broido, A., & Claffy, K. May 2005. Remote physical device fingerprinting. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*.
- [38] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. June 1999. Hypertext transfer protocol – HTTP/1.1. RFC 2616. <http://www.ietf.org/rfc/rfc2616.txt>.
- [39] Klyne, G., Nottingham, M., & Mogul, J. 2004. Registration procedures for message header fields.
- [40] Nottingham, M. & Mogul, J. December 2005. Http header field registrations.
- [41] Manoj, G. January 2006. An extension to cache-control, http/1.1 for group caching.
- [42] Snort. 2006. Sourcefire VRT Certified Rules - The Official Snort Ruleset (registered user release), ver. 2.3. <http://www.snort.org/pub-bin/downloads.cgi>, last visited 8th of March 2006.
- [43] Braden, R. October 1989. Requirements for internet hosts - application and support.
- [44] T.Berners-Lee, R.Fielding, L. 1998. Uniform resource identifiers (uri): Generic syntax. RFC 2396.

- [45] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., & Stewart, L. June 1999. HTTP authentication: Basic and digest access authentication. RFC 2617. <http://www.ietf.org/rfc/rfc2617.txt>.
- [46] Kristol, D. & Montulli, L. October 2000. Http state management mechanism. RFC 2965. <http://www.ietf.org/rfc/rfc2965.txt>.
- [47] trac, integrated scm & project management. <http://projects.edgewall.com/trac/>, last visited 20th June 2006.
- [48] Lenz, C. December 2004. Http caching in trac 0.9. <http://lists.edgewall.com/archive/trac/2004-December/001389.html>, last visited 20th June 2006.
- [49] Internet protocol details. Microsoft Exchange Server TechCenter, <http://www.microsoft.com/technet/prodtechnol/exchange/guides/E2k3TechRef/fbc63ab6-f17c-4526-a96a-2013b5baf08d.msp?mfr=true>, last visited 20th June 2006.
- [50] Hill, B. August 2003. Using web dav with iis 5.0. <http://securitypronews.com/it/security/spn-23-20030814UsingWebDAVwithIIS50.html>, last visited 20th June 2006.
- [51] Kristol, D. & Montulli, L. 1997. Http state management mechanism.
- [52] Troost, R., Dorner, S., & Moore, K. August 1997. Communicating presentation information in internet messages: The content-disposition header field.
- [53] libpcap. <http://www.tcpdump.org/>, last visited 20th June 2006.
- [54] Winpcap: The windows packet capture library. <http://www.winpcap.org/>, last visited 20th June 2006.
- [55] Ethereal. <http://www.ethereal.com/>, last visited 20th June 2006.
- [56] Høgskolen i Gjøvik, Postboks 191, Teknologivn. 22, 2802 Gjøvik.
- [57] Rekhter, Y., Moskowitz, B., Karrenberg, D., de, G. J., & Lear, E. Address allocation for private internets. RFC 1918, Internet Engineering Task Force, February 1996.
- [58] Mutz, D., Vigna, G., & Kemmerer, R. December 2003. An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems. In *Proceedings of the 2003 Annual Computer Security Applications Conference*, Las Vegas, Nevada.
- [59] Mucus. <http://www.cs.ucsb.edu/~rsg/Mucus/index.html>, last visited 20th June 2006.

A Classification table

This table is a summary of the methodology and the classification

Header	Classification	Snort rules	Substitution
HTTP 1.1 - General headers			
Cache-Control	<i>Could</i>	N	cache
Connection	<i>No</i>	N	
Date	<i>No</i>	N	
Pragma	<i>No</i>	N	
Trailer	<i>No</i>	N	
Transfer-Encoding	<i>No</i>	Y	
Upgrade	<i>No</i>	N	
Via	<i>Must</i>	N	www.foo...foo.bar
Warning	<i>Must</i>	N	www.foo...foo.bar
HTTP 1.1 - Request headers			
Method	<i>No</i>	Y	
Request-URI	<i>Must</i>	Y	n
HTTP-Version	<i>No</i>	Y	
Accept	<i>No</i>	Y	
Accept-Charset	<i>Should</i>	N	charset
Accept-Encoding	<i>No</i>	N	
Accept-Language	<i>Should</i>	N	l
Authorization	<i>Must</i>	Y	credentials
Cookie	<i>Must</i>	Y	cookie
Cookie2	<i>No</i>	N	
Expect	<i>No</i>	N	
From	<i>Must</i>	N	email
Host	<i>Must</i>	Y	www.foo...foo.bar
If-Match	<i>Should</i>	N	ifmatch
If-Modified-Since	<i>No</i>	N	
If-None-Match	<i>Should</i>	N	ifnonematch
If-Range	<i>Should</i>	N	ifrange
If-Unmodified-Since	<i>No</i>	N	
Max-Forwards	<i>No</i>	N	
Proxy-Authorization	<i>Must</i>	N	credentials
Range	<i>No</i>	N	
Referer	<i>Must</i>	N	www.foo...foo.bar
TE	<i>No</i>	N	
Translate	<i>No</i>	Y	
User-Agent	<i>Could</i>	Y	browser
HTTP 1.1 - Response headers			
HTTP-Version	<i>No</i>	N	
Status-Code	<i>No</i>	Y	

Continued on next page

Header	Classification	Snort rules	Substitution
Reason-Phrase	<i>No</i>	Y	
Accept-Ranges	<i>No</i>	N	
Age	<i>Could</i>	N	age
ETag	<i>Should</i>	N	etag
Location	<i>Must</i>	Y	www.foo...foo.bar
Proxy-Authenticate	<i>Must</i>	N	www.foo...foo.bar
Retry-After	<i>No</i>	N	
Server	<i>Should</i>	N	server
Set-Cookie	<i>Must</i>	N	setcookie
Set-Cookie2	<i>Must</i>	N	setcookie
Vary	<i>No</i>	N	
WWW-Authenticate	<i>Must</i>	N	www.foo...foo.bar
Authentication-Info	<i>No</i>	N	
Proxy-Authentication-Info	<i>No</i>	N	
HTTP 1.1 - Entity headers			
Allow	<i>No</i>	N	
Content-Encoding	<i>No</i>	N	
Content-Language	<i>Should</i>	N	l
Content-Length	<i>No</i>	Y	
Content-Location	<i>Must</i>	N	www.foo...foo.bar
Content-MD5	<i>No</i>	N	
Content-Range	<i>No</i>	N	
Content-Type	<i>Should</i>	Y	type
Expires	<i>No</i>	N	
Last-Modified	<i>No</i>	N	
Content-Disposition	<i>Must</i>	Y	file
HTTP 1.1 - Message body			
Message Body	<i>Must</i>	Y	x

Table 8: Classification

B Number of positives

This table is a summary of the number of positives reported by Snort during the experiments

Snort SID	Attack	# positives reported in experiment...									
		1	2	3	4	5	6	7	8	9	10
1497	WEB-MISC cross site scripting attempt	119	119	119	119	119	119	119	119	119	119
1122	WEB-MISC /etc/passwd	105	105	105	105	105	105	105	105	105	105
1165	WEB-MISC Novell Groupwise gwweb.exe access	16	16	16		16	16		16	16	
1147	WEB-MISC cat%20 access	12	12	12	3	12	12	3	15	12	3
1301	WEB-PHP admin.php access	12	12	9		12	9		12	9	
1201	ATTACK-RESPONSES 403 Forbidden	11	11	11	11	11	11	11	11	11	11
2281	WEB-PHP Setup.php access	11	11	11		11	11		11	11	
1540	WEB-COLDFUSION ?Mode=debug attempt	10	10	10		10	10		10	10	
1555	WEB-CGI DCShop access	10	10	10		10	10		10	10	
119:18:01	(http_inspect) WEBROOT DIRECTORY TRAVERSAL	10	10	8	1	10	8	1	10	8	1
3463	WEB-CGI awstats access	10	10	4		10	4		10	4	
1774	WEB-PHP bb_smilies.php access	8	8	8		8	8		8	8	
1614	WEB-MISC Novell Groupwise gwweb.exe attempt	8	8	8		8	8		8	8	
1523	WEB-MISC ans.pl access	8	8			8			8		
1522	WEB-MISC ans.pl attempt	8	8			8			8		
2565	WEB-PHP modules.php access	7	7	4		7	4		7	4	
1470	WEB-CGI listrec.pl access	6	6	6		6	6		6	6	
1668	WEB-CGI /cgi-bin/ access	6	6	6		6	6		6	6	
884	WEB-CGI formmail access	6	6	6		6	6		6	6	
1602	WEB-CGI htsearch access	6	6	3		6	3		6	3	
1969	WEB-MISC ion-p access	6	6	3		6	3		6	3	
882	WEB-CGI calendar access	6	6	3		6	3		6	3	
2326	WEB-IIS sgdynamo.exe access	6	6	3		6	3		6	3	
1376	WEB-MISC jrun directory browse attempt	6	6	1	1	6	1	1	6	1	1
1862	WEB-CGI mrtg.cgi directory traversal attempt	6	6			6			6		
1566	WEB-CGI eshop.pl access	4	4	4		4	4		4	4	
3813	WEB-CGI awstats.pl configdir command execution attempt	4	4	4		4	4		4	4	
2152	WEB-PHP test.php access	4	4	4		4	4		4	4	
2002	WEB-PHP remote include path	4	4			4			4		
1654	WEB-CGI cart32.exe access	4	4			4			4		
122:03:00	(portscan) TCP PortswEEP	3	3	3	3	3	3	3	3	3	3
1096	WEB-MISC Talentsoft Web+ internal IP Address access	3	3	3		3	3		3	3	
823	WEB-CGI cvsweb.cgi access	3	3	3		3	3		3	3	
1016	WEB-IIS global.asa access	3	3	3		3	3		3	3	
1149	WEB-CGI count.cgi access	3	3	3		3	3		3	3	
2241	WEB-MISC cwmail.exe access	3	3	3		3	3		3	3	
812	WEB-CGI webplus version access	3	3	3		3	3		3	3	
1078	WEB-MISC counter.exe access	3	3	3		3	3		3	3	
1824	WEB-CGI alienform.cgi access	3	3	3		3	3		3	3	
1825	WEB-CGI AlienForm af.cgi access	3	3	3		3	3		3	3	
1560	WEB-MISC /doc/ access	3	3	3		3	3		3	3	

Continued on next page

SID	Attack	1	2	3	4	5	6	7	8	9	10
1054	WEB-MISC weblog/tomcat .jsp view source attempt	3	3	3		3	3		3	3	
1875	WEB-CGI cgicso access	3	3	2		3	2		3	2	
1106	WEB-CGI Poll-it access	3	3			3			3		
2328	WEB-PHP authentication_index.php access	3	3			3			3		
901	WEB-CGI webspircgi access	3	3			3			3		
1396	WEB-CGI zml.cgi access	3	3			3			3		
1816	WEB-PHP directory.php access	3	3			3			3		
1300	WEB-PHP admin.php file upload attempt	3	3			3			3		
1815	WEB-PHP directory.php arbitrary command attempt	3	3			3			3		
2366	WEB-PHP PhpGedView PGV authentication_index.php base directory manipulation attempt	3	3			3			3		
3465	WEB-CGI RiSearch show.pl proxy attempt	3	3			3			3		
1637	WEB-CGI yabb access	3	3			3			3		
900	WEB-CGI webspircgi directory traversal attempt	3	3			3			3		
1998	WEB-PHP calendar.php access	3	3			3			3		
2410	WEB-PHP IGeneric Free Shopping Cart page.php access	3	3			3			3		
1395	WEB-CGI zml.cgi attempt	3	3			3			3		
2208	WEB-CGI fom.cgi access	3	3			3			3		
1521	WEB-MISC server-status access	2	2	2		2	2		2	2	
2585	WEB-MISC nessus 2.x 404 probe	2	2	2		2	2		2	2	
1770	WEB-MISC .FBCIndex access	2	2	2		2	2		2	2	
1520	WEB-MISC server-info access	2	2	2		2	2		2	2	
1852	WEB-MISC robots.txt access	2	2	2		2	2		2	2	
1213	WEB-MISC backup access	2	2	2		2	2		2	2	
2066	WEB-MISC Lotus Notes .pl script source download attempt	2	2	2		2	2		2	2	
835	WEB-CGI test-cgi access	2	2	2		2	2		2	2	
1847	WEB-MISC webalizer access	2	2	2		2	2		2	2	
1145	WEB-MISC / root access	2	2	2		2	2		2	2	
2484	WEB-MISC source.jsp access	2	2	1		2	1		2	1	
2060	WEB-MISC DB4Web access	2	2	1		2	1		2	1	
2441	WEB-MISC NetObserve authentication bypass attempt	1	1	1	1	1	1	1	1	1	1
2056	WEB-MISC TRACE attempt	1	1	1	1	1	1	1	1	1	1
1042	WEB-IIS view source via translate header	1	1	1	1	1	1	1	1	1	1
1848	WEB-MISC webcart-lite access	1	1	1		1	1		1	1	
1564	WEB-MISC login.htm access	1	1	1		1	1		1	1	
1551	WEB-MISC /CVS/Entries access	1	1	1		1	1		1	1	
976	WEB-IIS .bat? access	1	1	1		1	1		1	1	
1660	WEB-IIS trace.axd access	1	1	1		1	1		1	1	
993	WEB-IIS iisadmin access	1	1	1		1	1		1	1	
1554	WEB-CGI dbman db.cgi access	1	1	1		1	1		1	1	
1773	WEB-PHP php.exe access	1	1	1		1	1		1	1	
1769	WEB-MISC .DS_Store access	1	1	1		1	1		1	1	
1402	WEB-IIS iissamples access	1	1	1		1	1		1	1	
1493	WEB-MISC RBS ISP /newuser access	1	1	1		1	1		1	1	
1872	WEB-MISC Oracle Dynamic Monitoring Services dms access	1	1	1		1	1		1	1	
1857	WEB-MISC robot.txt access	1	1	1		1	1		1	1	
1826	WEB-MISC WEB-INF access	1	1	1		1	1		1	1	
971	WEB-IIS ISAPI .printer access	1	1	1		1	1		1	1	
1040	WEB-IIS srchadm access	1	1	1		1	1		1	1	
887	WEB-CGI www-sql access	1	1	1		1	1		1	1	
1230	WEB-MISC VirusWall FtpSave access	1	1	1		1	1		1	1	

Continued on next page

SID	Attack	1	2	3	4	5	6	7	8	9	10
1880	WEB-MISC oracle web application server access	1	1	1		1	1		1	1	
1154	WEB-MISC Domino names.nsf access	1	1	1		1	1		1	1	
2238	WEB-MISC WebLogic Console-Help view source attempt	1	1	1		1	1		1	1	
1286	WEB-IIS _mem_bin access	1	1	1		1	1		1	1	
1212	WEB-MISC Admin_files access	1	1	1		1	1		1	1	
1242	WEB-IIS ISAPI .ida access	1	1	1		1	1		1	1	
1214	WEB-MISC intranet access	1	1	1		1	1		1	1	
2242	WEB-MISC ddicgi.exe access	1	1	1		1	1		1	1	
1125	WEB-MISC webcart access	1	1	1		1	1		1	1	
1288	WEB-FRONTPAGE/_vti_bin/access	1	1	1		1	1		1	1	
1385	WEB-MISC mod-plsql administration access	1	1	1		1	1		1	1	
1874	WEB-MISC Oracle Java Process Manager access	1	1	1		1	1		1	1	
896	WEB-CGI way-board access	1	1	1		1	1		1	1	
1997	WEB-PHP read_body.php access attempt	1	1			1			1		
1406	WEB-CGI agora.cgi access	1	1			1			1		
908	WEB-COLDFUSION administrator access	1	1			1			1		
1534	WEB-CGI agora.cgi attempt	1	1			1			1		
1591	WEB-CGI faqmanager.cgi access	1	1			1			1		
1592	WEB-CGI /fcgi-bin/echo.exe access	1	1			1			1		
1827	WEB-MISC Tomcat servlet mapping cross site scripting attempt	1	1			1			1		
1829	WEB-MISC Tomcat TroubleShooter servlet access	1	1			1			1		
1590	WEB-CGI faqmanager.cgi arbitrary file access attempt	1	1			1			1		
	Total number of positives	590	590	472	246	590	472	246	593	472	246

Table 9: Number of positives

C Non-disclosure agreement

Høgskolen i Gjøvik
IT-Tjenesten
Postboks 191
Teknologiveien 22
2802 GJØVIK

Avtale om fravikelse fra offentliggjøring.

Antall sider i dette dokument: 2

Denne avtale dekker bruk av konfidensiell informasjon utlevert av Høgskolen i Gjøvik, IT-tjenesten (heretter omtalt som avgiver), mottatt av Vidar Evenrud Seeberg (heretter omtalt som mottaker).

Avgiver medgir med dette at informasjon som kan ansees som sensitiv og/eller konfidensiell, samt inneholde informasjon som kan falle inn under norsk lov om behandling av personopplysninger av 14. april 2000 og/eller Almindelig borgerlig Straffelov § 122 og/eller § 145, er innhentet i form av kommunikasjonslogger fra nettverkskommunikasjon, samt utdrag av trafikkdata fra datakommunikasjon. Avgiver har innhentet denne informasjon med hjemmel i gjeldene reglement om bruk av datasystemer ved Høgskolen i Gjøvik, titulert Reglement for bruk av høgskolens datautstyr, Fastsatt av høgskolestyret 14. juni 1999 (STY 40/99)". Avgiver har kun utført innhenting av slik informasjon i elektronisk form, og har ikke utført innsyn i informasjonen.

Mottaker medgir med dette at slik informasjon er overlevert mottaker fra avgiver i elektronisk form, til det formål å benytte dette som grunnlagsdata i forskningsøyemed. Mottaker aksepterer å frastå fra å skaffe seg innsyn i kommunikasjonsinnhold i form av nyttedata. Mottaker aksepterer å kun benytte informasjonens innhold om endepunkter, kommunikasjonssynkronisering og dataflyt-kontroll.

Mottaker medgir med dette å ikke ha tillatelse eller rett til å offentliggjøre informasjonen som er mottatt fra avgiver i sin opprinnelige form, og at en

slik tillatelse eller rett heller ikke innehas for deriverte data fra den opprinnelige informasjon, med unntak slik nevnt nedenunder.

Avgiver gir mottaker anledning til å gjennomføre anonymisering av informasjonen, hvor det settes krav til at alle nytteedata ekskluderes, og informasjonen modifiseres på en slik måte at opprinnelig avsender og mottaker i datakommunikasjon ikke kan identifiseres eller spores tilbake. Avgiver tillater mottaker å benytte utdrag av slik anonymisert informasjon som basisdokumentasjon i argumentasjon for forskningsresultater. Avgiver gir også mottaker rett til å benytte deriverte summariseringsresultater fra anonymisert informasjon til presentasjon av forskningsresultater.

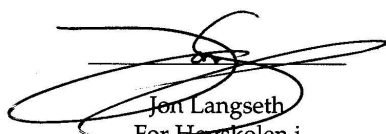
Med undertegnelse av denne avtale bekrefter begge parter de begrensninger og tillatelser som gir i dette dokument, samt å etter beste evne å ha bekreftet dokumentet å være fritt for opplysningsmessige feil.

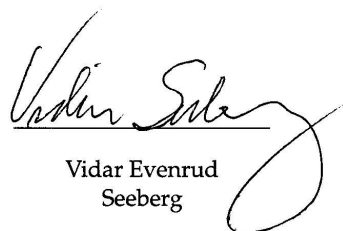
Gjøvik

Sted

29/5-06

Dato


Jon Langseth
For Høgskolen i
Gjøvik


Vidar Evenrud
Seeberg