Message Traceback Systems
Dancing with the Devil

Thesis by

Marios S. Andreou

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

Newcastle University

Newcastle upon Tyne, UK

2009

(Defended September 24, 2009)

Acknowledgements

There are too many things to thank my parents for. Most of all I would like to thank Kyriacos and Irene Andreou for their unconditional support of any pursuit that I ever expressed an interest in. Also I thank my wife and best friend Maria for making me smile; without you this thesis may never have come to fruition.

Thanks goes to my supervisor Aad van Moorsel for his patience and professionalism in helping me resolve any doubts regarding this work. A special thanks for Pete Lee who has been a true mentor since I first arrived at Newcastle Uni. in 2001; Pete thank you for always being available and for being so dedicated to what you do. I would also like to thank Santosh Shrivastava for his support and advice along the way and Robert Stroud for his encouragement in pursuing this degree in the first case.

Also thanks to all those people that make the School of Computing at Newcastle University 'work'. The positive impact that they have on the overall student experience cannot be understated. In particular I would like to thank Chris Ritson, Jim Wight, Tim Smith, Shirley Craig, Catherine McAndrew, Debbie Corbett, Louise Talbot, Alex Barfield, Emma Wilson, Keith Malcolm.

Thanks also to the original gang from 10.02: Christiaan Lamprecht, Michele Mazzucco, Joris Slegers, Chris Smith and Khaled Alekeish.

Abstract

The research community has produced a great deal of work in recent years in the areas of IP, layer 2 and connection-chain traceback. We collectively designate these as message traceback systems which, invariably aim to locate the origin of network data, in spite of any alterations effected to that data (whether legitimately or fraudulently). This thesis provides a unifying definition of spoofing and a classification based on this which aims to encompass all streams of message traceback research. The feasibility of this classification is established through its application to our literature review of the numerous known message traceback systems. We propose two layer 2 (L2) traceback systems, switch-SPIE and COTraSE, which adopt different approaches to logging based L2 traceback for switched ethernet.

Whilst message traceback in spite of spoofing is interesting and perhaps more challenging than at first seems, one might say that it is rather academic. Logging of network data is a controversial and unpopular notion and network administrators don't want the added installation and maintenance costs. However, European Parliament Directive 2006/24/EC requires that providers of publicly available electronic communications networks retain data in a form similar to mobile telephony call records, from April 2009 and for periods of up to 2 years. This thesis identifies the relevance of work in all areas of message traceback to the European data retention legislation. In the final part of this thesis we apply our experiences with L2 traceback, together with our definitions and classification of spoofing to discuss the issues that EU data retention implementations should consider. It is possible to 'do logging right' and even safeguard user privacy. However this can only occur if we fully understand the technical challenges, requiring much further work in all areas of logging based, message traceback systems. We have no choice but to dance with the devil.

Publications

The work in Chapters 4 and 5 was presented as:

- Marios S. Andreou and Aad van Moorsel: *Logging Based IP Traceback in Switched Ethernets.* In EUROSEC '08 Proceedings of the 1st European Workshop on System Security, pages 1-7. ACM, 2008.
- Marios S. Andreou and Aad van Moorsel: *COTraSE : Connection Oriented Traceback in Switched Ethernet.* In JIAS Journal of Information Assurance and Security, 4(2):91-105, 2009.
- A preliminary version of the above journal article was presented at conference as:
  Marios S. Andreou and Aad van Moorsel: *COTraSE : Connection Oriented Traceback in Switched Ethernet.* In IAS '08 The Fourth International Conference on Information Assurance and Security, 2008.

Glossary

- **Access port**
  In COTraSE and switch-SPIE this is a switchport to which a network host is attached *see Link port*
- **Connection Chain Traceback**
  Determining the instigating origin host when packets are sent via a chain of connections through compromised hosts *see Spoofing, Post send spoofing*
- **Effective origin**
  The last network host to make alterations to received network data *see Instigating origin*
- **External link port**
  In COTraSE this is a link port that does not connect a switch to a given logging node *see Link port, Internal link port*
- **Instigating origin**
  The network host that instigated the transmission of received network data *see Effective origin, Spoofing*
- **Internal link port**
  In COTraSE this is a link port that connects a switch to a given logging node, *see External link port, Access port*
- **IP Traceback**
  Determining the first hop router of the instigating origin host via mechanisms deployed at IP routers, in spite of sender spoofing, *see Spoofing, Sender spoofing*
- **L2 Traceback**
  Determining the instigating origin with the Layer 2 network of that host (e.g. switched ethernet, IEEE 802.11 wifi), in spite of any spoofing whether sender or post send *see Spoofing, Sender spoofing, Post send spoofing*

- **Link port**

  In COTraSE and switch-SPIE this is a switchport attached to another switch or router, *see Access port, Internal link port, External link port*

- **MAC-table**

  The MAC address table maintained by all ethernet switches; a given MAC address is reachable over the switchport from which frames carrying that address as source were most recently seen, *see Switchport*

- **Post send spoofing**

  Spoofing occurs after the data leaves the instigating origin, *see Spoofing, Sender spoofing*

- **Sender spoofing**

  Spoofing occurs at the instigating origin *see Spoofing, Post send spoofing*

- **Service data**

  Collectively refers to network data from the Transport, Network and Data-Link layers of the TCP/IP model, *see User data*

- **Spoofing**

  Alteration of any network data intended as an identifier, that effectively obscures the origin of the instigating origin; alterations can occur at the instigating origin or at any number of intermediate network hosts *see Sender spoofing, Post send spoofing*

- **Switchport**

  A physical Ethernet interface on an Ethernet switch (100Mbit/s *or* 1Gbit/s), *see Access port, Link port*

- **User data**

  Data from the Application layer of the TCP/IP model, *see Service data*

Contents

List of Figures

Chapter 1

Introduction

Hosts on the Internet are assigned a unique Internet Protocol (IP) address, which is reported as the source address in each IP packet header. This header source address is taken as an indication of the originating machine's identity. However, IP can not in itself prevent creation of packets with a forged source address. Explicit generation of these 'spoofed' packets requires a degree of planning and effort and so they are often associated with malevolent network activities [8]. The most widely known of these are Denial of Service attacks; it is obvious for instance that obscuring the origin of an attack may prolong its effects.

Another, far more ubiquitous form of 'spoofing', which is typically transparent to the user, results from commonly employed resource provisioning mechanisms. In a Local Area Network (LAN), a number of machines may share a smaller number of public IP addresses through the use of Network Address Translation (NAT), and repetitive requests for commonly accessed web pages are minimized with a proxying web cache. In such cases, the source IP address of packets leaving the LAN is overwritten with that of a gateway router. Effectively, the packet's originator is 'hidden' behind the collective identity of the local network. Even if the eventual wide-spread adoption of IPv6 renders NAT obsolete, mechanisms such as web-caching utilise similar techniques and will still be a necessary part of a typical LAN deployment (especially so for larger corporate or university networks).

Determining the originating host for a given IP packet, regardless of forged or overwritten source address is the domain of so called 'IP traceback systems'. A separate body of work addresses the issue of 'connection chain traceback' and is traditionally considered to be distinct from IP traceback (and its subdomain L2 traceback). In the first part of this thesis we develop a general classification of spoofing which aims to unify these related streams of research. We will then present our two L2 traceback systems designed for switched ethernet. The final part of this work discusses the relevance of work in all areas of message traceback to the European data retention directive, which requires widespread logging of network communications data.

1.1 Motivation

The fact that the source address of IP packets cannot always be taken to identify the origin host is compounded by the stateless operation of IP routers. At each router, any state generated in the process of forwarding a packet is discarded once it is successfully transmitted.

IP traceback systems in general try to compensate for this lack of state, and adopt either a signaling or logging approach (though hybrid schemes have been proposed [9]). In signaling, routing nodes signal their presence on a packet's network path; some approaches suggest marking the packet itself [10] whilst the use of separate messages has also been considered [11].

As their name suggests, logging based traceback systems require that routing nodes maintain a log of forwarded packets. The development of mechanisms to reduce memory requirements in such systems is thus central to their viability. Some logging based proposals achieve significant memory efficiency by using space efficient bloom filters for packet data [3] whilst our own work in this area adopts a 'connection-oriented' approach [12, 13].

However, underlying the obvious challenges is the much more subtle and technically challenging issue of how the 'origin' of a communication is defined. In the physical sense, the origin is an electronic device providing access to Internet connectivity services (a network host). In this sense the origin can be described in terms of geographical coordinates, the granularity of which will depend on the device's mobility. However, and as shown by work in the area of 'device fingerprinting', establishing a correlation between network data and the originating device (in the physical sense) remains an open issue [14].

IP traceback systems are typically intended for transit networks, and are deployed within or in close proximity to IP routers. Thus, at best they can determine the first hop router [5] of the origin host. Considering the size of even the smallest ISP networks, the 'origin' at this level of granularity is not useful in achieving accountability for network attacks. The actual origin can potentially be one of thousands of hosts. Investing in the infrastructure to log traffic only at routers will in practice be of little use to investigation or prosecution of online crime. Furthermore, in those cases where data is overwritten as in NAT, the source network is already identified by default as we can trust the address inserted by a routing node to be 'true'.

A sub-domain of IP Traceback has emerged in recent years as a direct result of this shortcoming, with proposals for "Layer 2 Traceback" (L2 traceback). These 'internal' traceback systems extend the tracking process beyond the leaf router, and into the originating network [5, 9, 15, 12]. Generally, L2 traceback systems combine an identifier of the network hardware and (where applicable) the port number from which data first entered the (local) network to specify 'origin'.

An end to end *message* traceback system that is able to trace the origin of any given IP packet (to any useable granularity) would need to be composed of both IP, L2 and in some cases connection-

chain traceback components. This thesis first sets about to unify these streams of research through a common and all encompassing definition of spoofing.

The issue of determining an IP packet's originating host when packets have been *spoofed* and in the face of a DoS attack is certainly interesting, though somewhat academic, mainly due to the associated costs. However, in April 2006 the European Parliament published a Directive mandating data retention in communications networks [16]. By April of 2009, all member states are expected to have put in place "laws, regulations and administrative provisions necessary" to store information regarding each "communication" made by hosts on "public communications networks". Work in the areas of IP and L2 traceback and especially logging based proposals, are useful in exposing the difficulties that implementations of this Directive will need to address.

Though April 2009 is the 'deadline' for EU member states to enforce data retention through national legislation, it will realistically be a number of years before they become a reality. The implementation of data retention will be difficult and come at a great cost to network and service providers, and by extension to all network operators, regardless of their size. Furthermore and considering the inherently sensitive nature of retained data, it is imperative that we ensure not only its *security* but also its *quality*; it *must not* for instance be possible to implicate an innocent tertiary party as the source of a suspect communication by 'spoofing' their network identity. The latter parts of this thesis identify and discuss the relevance of work in the areas of logging based message traceback systems (IP, L2 or connection chain) to the EU data retention directive.

.

1.2 Goal

The EU data retention directive will affect European citizens in a very personal way. Work in the areas of IP, Layer 2 and connection-chain traceback have exposed the many technical challenges posed by identifying the origin machine in spite of spoofing. Switched Ethernet provides a very interesting deployment environment for L2 traceback, due to the exclusivity with each each network host is connected to its switch, bringing both positive and negative consequences. For our L2 traceback system to be viable we must find ways to reduce memory requirements, but also to preserve user privacy whilst maintaining accountability for any given packet.

We hope that through practical work and better understanding we can develop a classification of spoofing and message traceback. This can then be applied to reasoning about implications that message traceback system design decisions will have on the volume, origin granularity, quality, security and data-mining characteristics of retained data.

Goal: **Our primary aim is to improve the technology by which logging based message traceback is implemented in switched ethernet networks, compared**

to known work in this area. We also aim to provide a comprehensive technical definition of spoofing that considers alterations to data at all layers of the TCP/IP model and which unifies known work in all areas of message traceback. Finally we apply our understanding and practical experiences to identify the important trade-offs that should be considered by implementations of EU Council Directive 2006/24/EC on the retention of data.

1.3 Approach

Our primary goal of developing a viable Layer 2 traceback system was achieved through an iterative development process. We studied available literature to derive the requirements that our intended system was to fulfill. From the outset, we sought a 'hands on' systems approach, as we felt that only through this would we be able to understand the *practical* issues that implementations of message traceback (or more generally data retention) need to address.

Layer 2 traceback systems, as we will see, are directly influenced by the specific (OSI) layer 2 network protocols and hardware in use (e.g. *shared* or *switched* ethernet). Switched Ethernet seemed to present the greatest challenges to logging based message traceback, mainly due to the lack of an obvious vantage point from which to observe network traffic. Interestingly however, it also allows for a trace result with the highest granularity, by identifying a specific network access point, which is why we chose this as a deployment environment.

Our first attempt at L2 traceback was an adaptation of an existing IP traceback system for switched ethernet [15]. The 'Source Path Isolation Engine' (SPIE) [3] satisfied our two key requirements of low memory utilitsation and preservation of user privacy. Furthermore, the source code of a SPIE implementation was publicly available from [17].

The available implementation is for the linux operating system and consists of two sub-components. A 'kernel space' driver receives IP packets and logs these as hashed digests, whilst the 'user space' program allows control of the driver and maintains a ring buffer of previously logged packets. This early work took far longer than anticipated, mainly due to the steep learning curve of familiarisation with linux device drivers and networking. However, this also served to demonstrate the difficulty of deploying such systems 'in the real world' (i.e. beyond academic research).

The SPIE system uses space efficient bloom filters to store packet digests. However, in a switched ethernet environment, each 'routing node' (i.e. ethernet switch) needs to allocate independent bloom filters for *each switch port*. Furthermore, to achieve an acceptably low level of false positives from the bloom filters the amount of allocated memory must be increased accordingly. The link speed is a dominating factor in determining memory allocation and false positive rate, as we will see, and so in a $1Gbit/s$ ethernet environment, memory requirements can quickly become excessive.

This early work provided us with the insights necessary to develop a logging based L2 traceback system for switched ethernet, which we term 'COTraSE' (Connection Oriented Traceback in Switched Ethernet). The aforementioned EU Directive on data retention provided the inspiration for our 'connection oriented' approach, which allows us to achieve acceptable memory requirements. We preserve user privacy by storing 'connection records' as hashed digests. Our earlier experiences with switched ethernet were central to the development of our 'switchport resolution' algorithm, with which we claim to detect malicious network behaviours such as MAC address spoofing.

Conceptually, our COTraSE system is one potential solution to the Layer 2 traceback system, which is preserving of user privacy and which has acceptable memory requirements. However, in practice it could also prove expensive to deploy as a logging node is required between each pair of network switches (though this can be relaxed under certain conditions). Furthermore, and as we will see, the privacy preserving features of COTraSE may need to be sacrificed if this is adapted to the requirements of the EU data retention.

This hands on experience allowed us to develop a general classification of spoofing and message traceback which unifies the relevant streams of research. This understanding and experience is then applied to a 'real world' network logging system. EU Council Directive 2006/24/EC legislates data retention but this has yet to be implemented in most member states and so there is still scope for influencing any potential designs.

1.4 Novelty of Approach

For our practical work in developing a Layer 2 traceback system, we have sought a feasible 'real world' solution, that is, one which does not assume functionality that is not currently available. At the same time, we must make provisions for the potential of 'spoofed' MAC and IP addresses, as this reflects the realistic abilities of any determined attacker.

This is evident in both of our proposals and in the ways with which we overcome the problems of logging in switched ethernet. Other proposals do not always adhere to this principle; in [9] an 'in switch' process is assumed for accessing ethernet frames whilst in [5] unreasonable processing demands are made on the leaf router which logs all traffic from the local network.

In our first proposal, switch-SPIE [15], we employ switchport mirroring for access to network traffic. This functionality is available in most network switches and though not standardised is identified as one of the monitoring mechanisms for switched ethernet [18, 19]. The effects of using mirroring are twofold; first, the logging processes are distributed to each switch, and so we avoid making unrealistic demands on any single routing node, as was the case in [5]. This provides for a more flexible deployment environment where we do not need to make any assumptions regarding network topology. Secondly, the process of determining the origin switchport, even in the face of

spoofed MAC addresses is much closer to the origin host which gives a higher degree of accuracy and makes forging the source MAC address harder.

All known logging based IP and L2 traceback systems that claim to traceback *any given packet* can be characterised as 'explicit packet logs', that is all packets are logged. This obviously has implications on the memory requirements for such systems. In our second proposal, COTraSE, we instead maintain 'connection records'. We note that though this is similar to the approach taken by CISCO NetFlow [20], we use different flow termination heuristics and allow for the possibility of spoofed MAC or IP addresses.

A common characteristic of logging based L2 traceback systems for switched ethernet is the use of the switch MAC address table (MAC-table) to determine the originating switchport based on the source MAC address reported in ethernet frame headers. In COTraSE however, we uniquely correlate the MAC-table values from two directly adjacent switches. This allows us to more accurately determine the source switchport and as we will see also provides an element of error detection, such as MAC address spoofing. Furthermore, COTraSE also takes into account the effects of *overwritten* MAC addresses, as in NAT or web caching mentioned earlier, and maintains 'translation records' at the leaf router.

Finally, we are the first to identify the relevance and applicability of work in the areas of IP, L2 and connection-chain traceback to the EU data retention directive. Furthermore we are the first to consider IP, L2 and connection-chain traceback systems as part of a larger end-to-end message traceback system.

1.5 Contributions

The contributions of this thesis go beyond the provision of two Layer 2 traceback systems for switched ethernet. We will see how a unifying definition and simple classification of spoofing can bring together relevant research in the areas of IP, L2 and connection-chain traceback. Furthermore, we will show that we can directly apply the experience in developing these message traceback systems to the implementation of EU directive 2006/24EC. The main contributions of this thesis are:

- Unifying definition and classification of spoofing,
- Extensive literature review of all known work in the areas of IP, L2 and connection-chain traceback using our own classification from above,
- The switch-SPIE L2 traceback system, adapted from SPIE, presented as 'Logging Based IP traceback in Switched Ethernet' at Eurosec 2008 [15],
- The improved L2 traceback system, COTraSE, inspired by EU Directive 2006/24/EC, presented as 'COTraSE: Connection Oriented Traceback in Switched Ethernet' at IAS 2008 [13]

and later published in the Journal of Information Assurance and Security [12],

- Identification of the relevance of message traceback systems to the EU data retention directive,

- Identification and discussion of key challenges faced by data retention implementations.

Work in the areas of IP and connection chain traceback have traditionally been considered as distinct. In this thesis we argue that IP traceback (and its L2 sub-domain) and connection-chain traceback are both fundamentally addressing the same issue of message traceback: How can we traceback a given network message (e.g., an ethernet frame) to its origin host if we assume that the information contained in that message (especially source addresses) has been altered? Our definition of spoofing will primarily aims to define spoofing in a way applicable to all related domains. We show that IP and L2 traceback address the issue of sender spoofing, whilst connection-chain traceback addresses post-send spoofing. We present the literature review using our classification and definition in order to demonstrate their viability.

The switch-SPIE system is our first attempt at L2 traceback for a switched ethernet, which explicitly logs all packets using bloom filters. We adapted the open source SPIE IP traceback system and assigned an exclusive bloom filter for each switchport. We added the step of switchport resolution to the 'normal' SPIE logging routines, in order to determine the appropriate bloom filter for each received frame. We aimed to improve an earlier adaptation of SPIE for switched ethernet by deploying logging locally at each switch, rather than at the gateway router. The implications of this are that the traceback result is more reliable as local MAC-tables can be refreshed from the switch more frequently. Furthermore, this topology allows one to traceback frames that are entirely local and do not traverse the gateway router. The problem of low traffic visibility in switched Ethernet is well known and switch-SPIE took the approach of using switchport mirroring. Our experiments showed that this is not a reliable means of receiving network traffic. This is especially true for larger switches where the aggregate traffic from a number of switchports will quickly overwhelm the monitor port. Our discussion of memory requirements showed the limitation of our 'one bloom filter per switchport' approach; memory requirements are excessive in the worst case of a very large switch (1000+ switchports), especially when maintaining a low false positive rate (equating to a larger bloom filter).

The COTraSE system was primarily intended to address the challenges revealed by switch-SPIE. This is evident for instance in the placement of connection record logs at a tap between switches, rather than relying on switchport mirroring. However, at about this time the EU data retention directive came to our attention and we instantly recognised the applicability of logging based IP and L2 traceback systems to this legislation. The connection oriented logging approach taken by COTraSE was in part inspired by the EUDRD as we have seen. In COTraSE we further exploit the

switch MAC-tables to make the switchport resolution algorithm even more reliable. We will see that at the cost of an extra MAC-table lookup and some prior configuration at each switch to classify switchports (e.g., access or internal link ports) switchport resolution can even detect malicious activity such as sender spoofing of source MAC address. Furthermore, by maintaining translation records at the gateway router, COTraSE can traceback IP packets that are delivered to external hosts, regardless of any legitimate post-send spoofing such as NAT. The potential memory efficiency of the connection oriented approach was demonstrated using WAN traces from large ISPs that were taken as input by our connection record log implementation. The traces consist of over $136,503,068$ packets (over 12 hours) from an 'OC12' link in Miami, $114,181,288$packets (over 30 minutes) from an 'OC48' link in California and $89,357,967$ packets (over 90 minutes) from a 100Mbit/s Ethernet transit link. We will see that even in the worst case of a 10 second purge interval, the number of connection records as a percentage of total frames is consistently lower than 15% across all traces.

Finally we will introduce the EU data retention Directive with an executive summary, before listing the key challenges that any implementation will have to consider. Our discussion draws on our experiences with switch-SPIE and COTraSE and also makes use of the definitions that were developed in the first parts of this thesis.

1.6 Thesis Outline

This Chapter introduces our thesis, outlining the main goals and our approach in satisfying these. We have introduced the notions of spoofing and message traceback systems and their relevance to the European data retention directive 2006/24/EC. We also summarised the main contributions and are outlining the thesis contents here.

In Chapters 2 and 3 we will unify the relevant threads of research, namely 'IP and L2 traceback' but also 'connection-chain traceback' and 'host fingerprinting'. In Chapter 2 we begin this process by providing a unifying definition of 'spoofing' together with a classification of its various forms. We will see that spoofing is the alteration of any network data intended as an identifier (ephemeral or otherwise), that effectively obscures the identity of the instigating origin (sending) host.

Whilst Chapter 2 unifies the various forms of spoofing, Chapter 3 provides an extensive literature review. We consider proposals that in some way address the issue of tracing back to the origin of network data, in spite of any sender or post-send spoofing. We will see that IP traceback systems aim to identify the origin of network data in spite of sender spoofing. We will also summarise all known proposals for L2 traceback and the main challenges that these identify. Chapter 3 also considers proposals for connection chain traceback that address the issue of post-send spoofing.

In Chapter 4 we present the first of our L2 traceback systems, switch-SPIE. For any given IP

packet switch-SPIE aims to identify the instigating origin host, through the unique pairing of switch identifier and switch-port number (sID, pNO). Our proposal improves over an earlier adaptation of SPIE for switched ethernet [9] by deploying logging at each switch, rather than at a single network location. As we see switch-SPIE can traceback frames that are entirely local (i.e., do not traverse the gateway router) and our configuration allows the MAC-tables maintained by each logging node to be updated with a higher frequency than in [9]. Our experiments show that whilst the switch can 'keep up' with switch-port mirroring without adverse effect to the primary switching function, the network host and switch-port to which traffic is mirrored are quickly overwhelmed. Table 4.2 shows the memory requirements for 100Mbit and 1000Mbit ethernet with different bloom filter sizes and the corresponding false positive rates.

Chapter 5 presents our second attempt at L2 traceback for a switched ethernet, where we build on our experience with switch-SPIE. Our COTraSE system adopts a connection oriented approach to logging as we will see, that was in part inspired by EU Directive 2006/24/EC. COTraSE decreases memory requirements by only logging a subset of all packets. Crucially and as is explained this does not sacrifice our ability to traceback any given ethernet frame or packet and provide the unique {sID, pNO} pairing to identify the instigating origin. We also provide details of our implementation of the connection record logging algorithm which uses 'real' WAN trace data as input.

In Chapter 6 we present EU Directive 2006/24/EC and identify the relevance of work in the areas of IP and L2 traceback to the implementation of any 'data retention' system. We will argue the importance of the accuracy and reliability of retained data and consider the subtle tradeoff between storing data as plaintext or as encrypted ciphertext. We consider the problem posed by legitimate post send spoofing such as NAT and the difficulties associated with logging of email when application level data is designated as off-limits.

Finally Chapter 7 concludes this work. We begin with a thesis summary and discussion of its main contributions. We consider future research directions and finally provide our concluding remarks.

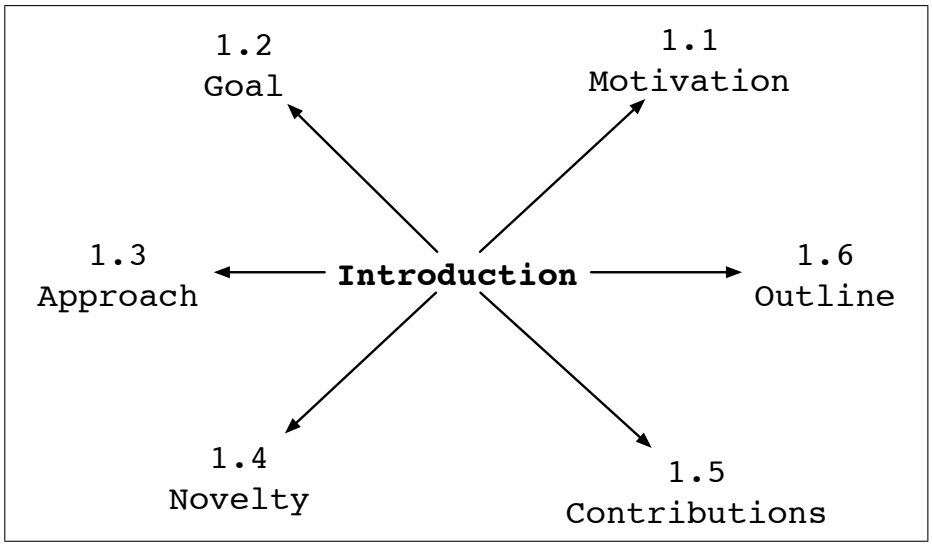


Figure 1.1: Idea map for Chapter 1 showing placement of sections

Chapter 2

On Spoofing

2.1 Introduction

This chapter provides an extended definition and discussion of spoofing as this is directly relevant to our work in the area of L2 traceback. Furthermore, whilst all message traceback systems address some form of spoofing, use of the term is not always consistent. In IP traceback proposals for instance it refers exclusively to alteration of source IP address in packets before these are transmitted from a network host, whilst in connection-chain proposals it refers to alterations effected at some compromised host.

The effect of any alteration to network data is invariably to obscure the origin of that data from the point of view of a recipient host. However, there are other well known methods beyond IP address forgery with which attackers conceal their identity (but which are perhaps not traditionally considered 'spoofing'). We thus develop a broader definition of spoofing and identify the two main types of sender and post-send spoofing. For each of these we consider alterations effected to user data (Application layer) or service data (Transport/Network/Data-link layer). Our definition considers legitimate alterations to packet data as well as the more obvious fraudulent alterations. In so far as our definition of "spoof" is more inclusive, it is also more faithful to the original, broader connotation of the word, attributed to a 19th century card game involving "trickery and nonsense" [21].

2.2 Spoofing

In this thesis we broaden the definition of spoofing from that typically implied and adopted by the relevant network security literature; there are more elements to spoofing beyond the use of a forged source address in IP packets before these are transmitted. Our definition considers which other fields within network data might be altered, as well as when this might occur; within the instigating (sending) host, *or*, en route to the intended destination.

Furthermore, our definition explicitly removes the implication of fraudulence from the term spoof-

ing. The aim is to include all mechanisms by which network data might be altered before reaching a recipient network host; this includes legitimate network processes (such as Network Address Translation - NAT [22, 23]) as well as illegitimate mechanisms on compromised hosts (as in connection chaining [24, 6]). We first provide our definitions and introduce some associated terms before considering each type of spoofing with examples in subsequent sections.

Definitions:

- **Instigating Origin**: the network host that instigated the transmission of received network data; the sending host.
- **Effective Origin**: the last network host to make any alterations to received network data; effectively the origin of network data. In some instances this is the same as the instigating origin.
- **Spoofing**: Alteration(s) of any network data intended as an identifier (ephemeral or otherwise), that effectively obscures the identity of the instigating origin; alteration(s) can occur at the instigating origin or in any number of intermediate network hosts en route to the intended recipient host. Spoofing does not imply fraud as it will often follow from a legitimate network process.

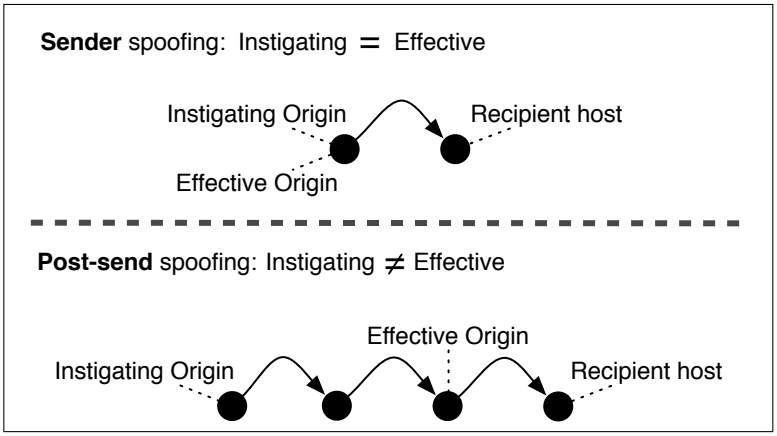


Figure 2.1: Instigating and Effective origins

Figure 2.1 illustrates the relation between the instigating and effective origins in terms of where data is altered; as shown in the diagram, we use the term sender spoofing to describe alterations to network data at the instigating origin. This differs from post-send spoofing, where alterations are made to network data by intermediate network hosts, the last of which becomes the effective origin.

In terms of which data is altered, we differentiate between user data and service data. As can be seen in Figure 2.2, the former refers to data at the Application layer of the Internet Model [25]

(also known as the 'TCP/IP model'), whilst the latter includes data from the Transport, Network and Data-link layers.

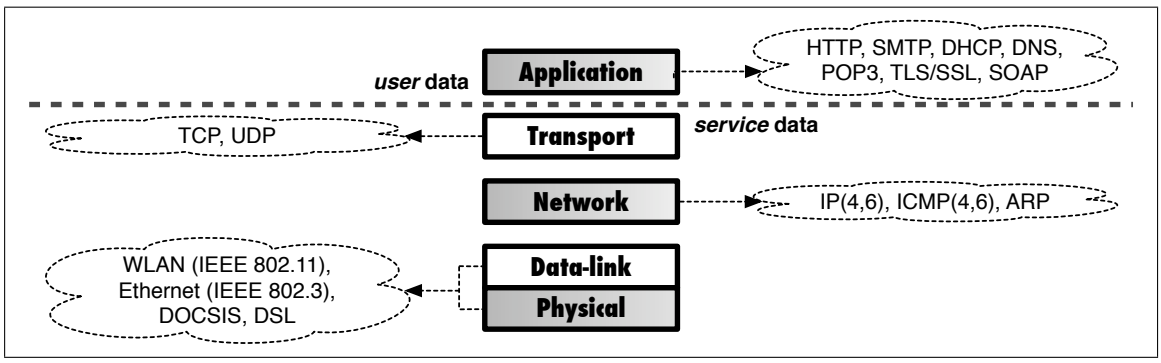


Figure 2.2: The Internet model showing the division between user and service data

The user data represents the information for which an end user has requested transmission across the network. The service data on the other hand comes from the networking services which facilitate that transmission. The user does not normally modify the service data, at least not directly (e.g. this is delegated to the operating system). To directly modify service data is anomalous and two well known examples are IP and MAC address spoofing to make packets appear to have originated from a tertiary host. We use the term sender spoofing to describe such cases, where alterations to user or service data occur only at the instigating origin host.

Together, the user and service data are forwarded through the network towards the intended recipient host. If this is a local destination, that is, within the same Local Area Network (LAN), then all data should arrive unchanged in its entirety. For external destinations, the service data is very likely to have been altered. A common and legitimate occurrence are alterations at the Data-link layer, signifying a change between the local network's physical transmission medium (e.g., switched Ethernet or WiFi) and that of the link to adjacent networks (e.g., DSL, ATM). As we will see, changes at the Network layer are also common due to resource provisioning mechanisms such as web caching or Network Address Translation. We use the term post-send spoofing to describe such cases, where alterations to user or service data occur after they are transmitted from the instigating origin host. The last host to effect alterations to network data becomes the effective origin, as illustrated in Figure 2.1.

Just as service data should not be altered by the user, the user data should not be altered by network services en route. User data is generally regarded as private, and rarely processed by routing nodes except in certain exceptional cases (we will consider an example subsequently).

We incorporate these concepts into a classification of spoofing based upon when alterations to data occur, as shown in Figure 2.3. This simple classification allows a more systematic examination of spoofing to follow, in which we elaborate on the examples of which data might be altered in each

case as summarised in Figure 2.3. Sender spoofing is always fraudulent, whilst post-send spoofing can also be legitimate; for each of these we first consider user data before considering alterations to service data, at each protocol layer.

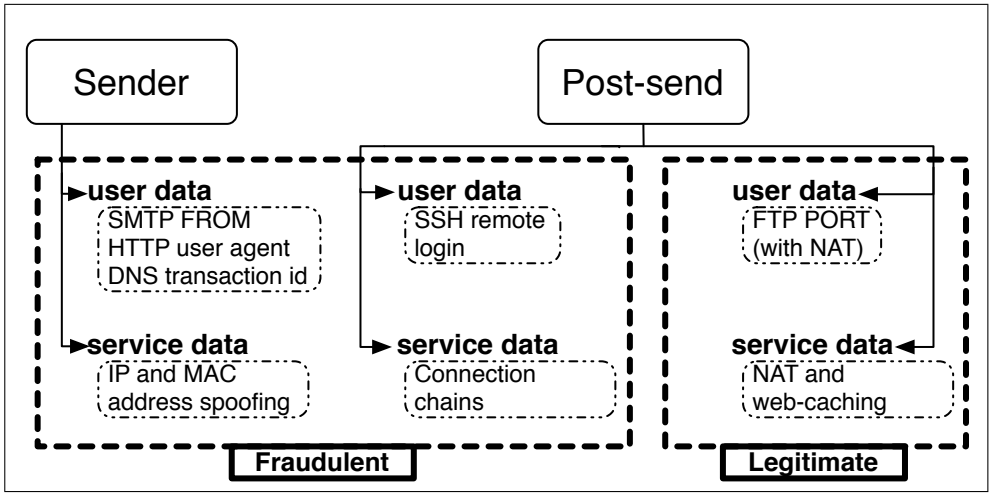


Figure 2.3: Classification of spoofing distinguishing when alterations occur and which data is altered with examples.

2.3 Sender spoofing

As can be seen in Figure 2.3 we consider that sender spoofing is always fraudulent. We make this claim as we cannot envisage any conditions under which sender spoofing might be legitimately employed.

In the context of sender spoofing, we refer to alterations of network data values from those that are pre-designated or otherwise true. For instance alteration of the MAC address provided by a networking hardware manufacturer, alteration of an IP address provided by a network administrator or alteration of an initial sequence number from that which is chosen at some instance in time by a particular TCP implementation.

2.3.1 Sender spoofing of user data

As can be seen in Figure 2.2 the user data refers to the Application layer of the Internet model. The user data cannot be trusted as a reliable indication of the instigating origin's identity without additional authentication mechanisms such as SSL. Thus, the use of untrue or invalid values in user data should not cause any problems to recipient hosts (this would mean an easily exploitable vulnerability).

However, there are well known cases in which the user data contains values that serve as identifiers. It is not possible to enumerate all application layer protocols and identify all data fields within

those that might be altered to the effect of obscuring the instigating machine's identity. In what follows we provide two interesting and well known cases of sender spoofing of user data to illustrate the potential effects.

The ubiquitous Electronic Mail (e-mail) is one well known example where user data contains values that identify the sender and specifically the source e-mail address. A motivated network user can trivially change the `FROM` field in an `SMTP` exchange; spammers using open `SMTP` relays will often use a false source e-mail address in their message. A common use of this type of spoofing is in 'phishing' where an e-mail may claim to have been sent by a known or trusted third party (such as a bank). Thus one should not typically rely on the source address of an e-mail to accurately identify its sender.

Another example is found in the `HTTP` protocol 'user-agent' header, which should be used as part of client requests [26] such as a `HTTP GET`. Web browsers include a user-agent string in their requests for resources from HTTP servers, to identify the browser and operating system of the instigating origin host. For instance, the string:

> User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.12) Gecko/20080201 Firefox/2.0.0.12

identifies the Firefox browser running on Windows XP. The user agent is used to address incompatibility issues across different web browsers, by tailoring content sent from the server to the given client browser. A common use is in the downloading of multi-platform applications; web servers can use the user agent to determine which installation file the client should receive (e.g., the Microsoft Windows '.exe' file or the Apple OS X '.dmg' file).

It is trivial for a user to change their user agent, thus spoofing their `HTTP` identity. Some popular web browsers even make such functionality directly available [27]. For a web server to use the 'user-agent' as a criterion in granting access to resources is then naive, to say the least. None the less, it has been suggested [28] that one may gain access to resources on websites that otherwise require completion of a registration procedure by masquerading as a search engine 'bot'. A 'bot' (robot) is a network host which automatically retrieves and indexes web sites on behalf of a given search engine. The string "`Googlebot/2.1 (+http://www.googlebot.com/bot.html)`" for example is one of the user-agent strings used by Google bots. These indexing hosts are usually given liberal access to a web site's pages, unless the site administrator takes measures against this. The vulnerability of spoofing your HTTP identity to gain access to restricted web sites is becoming less prevalent. Web servers are taking precautions such as verifying if the IP address of a host claiming to be a bot falls within the range of addresses known to be used by the given search engine. The introduction of such measures may explain why we have been unable to verify this exploit, which is described in [28].

2.3.2 Sender spoofing of service data

In the discussion that follows we consider the utility of spoofing a given value from an attacker's point of view as we consider that sender spoofing is always fraudulent. That is we cannot envisage a scenario in which sender spoofing occurs without the intent to defraud. Of course, it is not possible to foresee all possible ways in which this may be employed or manifested in an attack or exploit.

The service data, being charged with facilitating transmission of the user data through the network, will typically contain a number of values used as identifiers, and these have differing significance through the Internet Model layers.

At the Transport layer, a destination `TCP` or `UDP` port number is used to identify which application on the recipient host is to receive and process user data. Similarly, the source port number identifies the sending application, to which any replies need to be addressed. Two other important Transport layer identifiers, exclusive to reliable data delivery services (i.e., TCP as opposed to UDP), are the sequence and acknowledgement number fields. These serve to uniquely identify the user data bytes sent and successfully received by each party in a given communication. Though both ports and sequence numbers are intended as identifiers, their semantic scope is limited to a specific property of some network process within the host. Thus, they cannot be taken in isolation to identify the host itself.

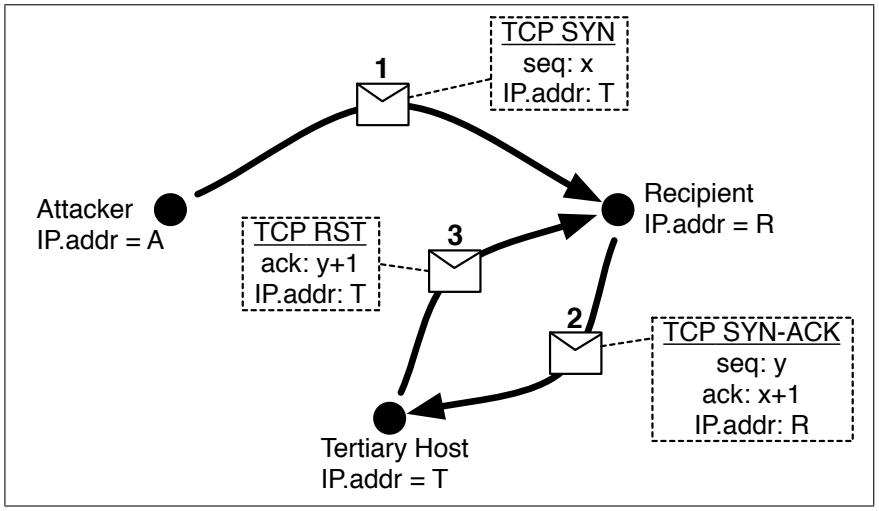


Figure 2.4: An attacker attempting to establish a spoofed TCP connection is foiled by the Tertiary host's TCP RST, which resets the connection

Nonetheless, an attacker needs to very carefully and specifically alter this Transport layer data, when the objective is to establish a (spoofed) TCP connection. Consider that an an attacker forging his source IP address cannot expect to receive any replies, as illustrated in Figure 2.4. Here the attacker with IP address A sends a `TCP SYN` to establish a connection with host R. However the attacker uses the IP address of the tertiary host as a source address. Thus, any replies are addressed

to T and so the attacker isn't privy to the initial sequence number chosen by the victim. This is of course a prerequisite to completing the 'TCP handshake' [29]. For this reason, spoofing at the Transport layer (TCP sequence numbers) often goes hand-in-hand with (and is in fact made necessary by) spoofing at the Network layer (source IP address), which we consider forthwith.

Sender spoofing at the Network and Data-link layer are well known and termed 'IP spoofing' and 'MAC spoofing' in the literature. The source IP and source MAC addresses are the only values in service data intended, in isolation, to indicate the identity of the host from which a given frame or packet has originated. In some respects, the latter is of higher significance, as each 48 bit MAC address is both globally unique and permanent; the first 3 bytes provide the Organisationally Unique Identifier assigned to each manufacturer, whilst the last 3 identify a specific network interface and designated during production. An IP address on the other hand is more transient, as it may be re-assigned to another host (of course this is not true for IPv6). None the less, sender spoofing at the Network rather than the Data-link layer may be more important for an attacker, depending on the locality of the attack.

The source MAC address will not reach destinations beyond the local network, as it is replaced at a leaf router. So we can say that generally, if targeting non-local destinations, sender spoofing at the Network (and Transport) layer is the most important for an attacker, with the values at the Data-link layer being irrelevant. However, when launching local attacks, then sender spoofing at both Network and Data-link layers is necessary. It would not make much sense for a malicious party to reveal their actual IP address whilst obscuring their MAC address, if both of these are visible to the destination. Thus, attack locality is a determinant factor in whether MAC or IP spoofing are employed (or indeed both).

Another important issue requiring an attacker's consideration is the validity of the altered data values, as this bears upon the potential impact of an attack. In their classification of Distributed Denial of Service (DDoS) attack mechanisms, Mirkovic *et al* [30] consider "source address validity" referring specifically to the IP address and distinguish between routable and non-routable addresses. Routable IP addresses belong to active tertiary hosts, whilst non-routable addresses are either assigned and not currently used (host is offline), or come from within the `IPv4` private address range [31]. The authors go on to distinguish how an attacker might select the spoofed IP address and consider use of a random address, selecting one from a fixed set or using one from the local network (random, fixed and subnet spoofing, respectively). We can extend the notion of validity and relate it to our discussion of TCP and the initial sequence number; an attacker must ensure that when TCP is in play, consecutively transmitted packets carry valid values, especially sequence numbers and ports.

We incorporate the concepts developed by Mirkovic *et al* here to show the interplay between validity of altered service data values and attack locality. When the attacker alters a source address,

whether Network or Data-link layer depending on attack locality, the recipient victim(s) will send any replies to the address that was indicated as source, as illustrated in Figure 2.4. We have already seen that this is significant in determining the attacker's ability to establish spoofed TCP connections [29]. The prediction of sequence numbers has become a less viable option for the attacker, as the TCP implementations of most popular operating systems will guard against such attacks [32].

In [30] the authors explain that the use of non-routable addresses falling within the `IPv4` private address range is easily detectable. We make a slight qualification to say that the use of these addresses is not so easily detectable if instigating origin and recipient host are on the same LAN (in general these addresses should not be seen on the Internet [25], but are employed in many LANs). Furthermore, the use of non-routable addresses that are assigned but not active may cause generation of Internet Control Message Protocol (ICMP) `Destination Unreachable` or `Time Exceeded` messages by any handling routers, which would reveal the spoofing attempt.

Thus, an attacker may best choose a routable address from within the local network (subnet). Referring to attack locality, in this case attacker and tertiary host are in the same local area network and so a shared physical transmission medium (e.g., shared Ethernet or wireless LAN) is advantageous to the attacker. It becomes possible for the attacker to 'sniff' network traffic in order to discover the initial sequence numbers chosen by the victim. Another advantage from using a routable and local address is that an attacker can more feasibly ensure that a currently active tertiary host does not respond to the unsolicited packets sent from the victim. This is illustrated in Figure 2.4 where a tertiary host responds to an unsolicited `SYN-ACK` with a `RST` to reset the connection, preventing the attacker from establishing a spoofed TCP connection.

2.3.3 Attacks facilitated by sender spoofing

As seen above there are a number of issues influencing how sender spoofing might be used by an attacker. Here we provide a brief overview of some network attacks that employ sender spoofing. In most cases spoofing serves only to obscure the instigating origin's identity whilst in others the act of altering network data also delivers the attack. Sender spoofing is most often associated with Denial of Service (DoS) which we will examine after first considering some non DoS attacks.

We have discussed the establishment of a spoofed TCP connection and the challenges this poses for an attacker (as summarised in Figure 2.4). A classic example is that of Kevin Mitnick, who employed sender spoofing to successfully steal files from Tsutomu Shimomura's computer [33]. Sequence number prediction was used to spoof a TCP connection to the victim with a separate Denial of Service attack on the tertiary host to stop it from transmitting `RST` frames. Furthermore, the attacker was non-local, which serves to demonstrate that spoofing of TCP connections is possible even under the most adverse conditions for the attacker.

Another example is the "dumb host scan" [34] which allows for stealthy (TCP) port scanning.

An attacker is able to determine whether a given TCP port on a potential victim host is accepting connections; they may wish to know if a given service is available for which an exploit is known. However, the attacker does not want direct communication with the victim in case logging is deployed, implicating the attacker in subsequent investigations. The port scan begins with the recruitment of a suitable and responding tertiary host, that increments its IPv4 identification field (IPID) in a predictable way. The attacker first notes this IPID value then sends a TCP SYN to the victim, spoofed as having come from the tertiary host. If the victim's port is responding, it will send a SYN ACK to the tertiary host, that will respond with a RST as seen in Figure 2.4. By obtaining the value of the tertiary host's IPID field after the scan, the attacker can determine whether the victim had responded to the initial (spoofed) SYN, prompting the tertiary host to send RST packets to the victim. In this case the port was open and can then be targeted by the attacker.

In general, the goal of a Denial of Service (DoS) attack is to cause the recipient host to enter a stalled state; this may occur as a result of resource exhaustion caused by an incoming packet flood, or, through the exploit of a specific vulnerability in the recipient application. The 'denial of service' refers either directly to the loss of service at the targeted host, or to denial of service for any tertiary hosts whom the victim served (e.g., the victim could be a local DHCP server). Figure 2.5 summarises the examples that we examine below.

In DoS attacks that exploit specific vulnerabilities only a small number of packets are required to deliver the attack. An example is the well known 'Land' attack whereby a single packet with the same source and destination TCP ports and IP addresses "freezes win95 boxes" [35, 36, 37]. This was later found to affect a great range of operating systems, including Ciscos IOS [38].

Another example is the Teardrop attack [36, 37] which exploits a vulnerability in the reassembly of fragmented IP packets with only two packets. In this case sender spoofing involves altering the IPv4 source address to hide the instigating origin's identity but also the IPv4 Fragment offset field which actually delivers the attack. Once memory is allocated for the first fragment received, the fragment offset of the second packet is altered to point inside the memory area allocated for the first fragment. This subsequently causes an irrecoverable failure in the memory allocation routine [37].

Packet fragment reassembly is also targetted by the dramatically named 'ping of death' [39]. In this case, an inflated ICMP echo-request (a 'ping') is created that is larger than the permissible maximum transmission unit at the Network layer, to ensure that the resulting IPv4 packet is fragmented for delivery. A recipient host will only reassemble fragments once all of these have been delivered, using the fragment offset field to dictate where in the reconstructed packet each fragment occurs [25]. An IPv4 packet cannot be larger than $65,536$ bytes, as constrained by the 16-bit total length field. In the ping of death the last fragment is of sufficient size so that when combined with the previous fragment offset creates a packet larger than the maximum permissible (causing a buffer overflow).

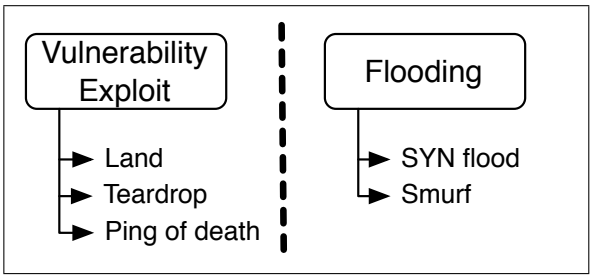


Figure 2.5: Examples of known DoS attacks that employ sender spoofing

Rather than targeting a specific vulnerability, flooding DoS attacks involve inundating a victim with an overwhelming packet load. The most well known example, perhaps due to its simplicity is the SYN Flood [40]. We have already mentioned the TCP handshake, that is a pre-requisite to connection establishment. In a SYN Flood, the attacker initiates a TCP connection but then does nothing else. At the recipient host, some state is maintained about the (potential) new connection, and the attacker's SYN is acknowledged with an ACK, sent from the victim to the tertiary masqueraded host. By repeatedly creating such 'half open connections' the attacker can consume resources at the recipient host until this becomes unable to accept any new connections.

In the examples seen so far the victim of a sender spoofing attack has also been the recipient of spoofed packets. However, in so called reflection attacks [8], it is the tertiary masqueraded host that most suffers the consequences of an attacker's efforts. An attacker can generate a large number of service requests (e.g., DNS requests) which carry the address of the tertiary host as source IP address; a sufficient number of replies will overwhelm the tertiary host. An example is 'smurf' [41], where a large number of ICMP echo-request packets are sent, appearing to have come from the tertiary host, which must deal with the ensuing wave of echo-reply packets.

2.4 Post-send spoofing

In post-send spoofing, alterations are effected to user and/or service data after it has left the instigating origin host. The last host to make changes to data before it reaches the intended recipient becomes the effective origin of that data as depicted in Figure 2.1. Thus, unlike sender spoofing and as per our definitions earlier, in post-send spoofing the instigating and effective origins are not the same network host.

Furthermore, unlike sender spoofing which is always fraudulent, post-send spoofing can also be used legitimately. There are a number of widely employed network mechanisms which result in user or service data being altered in such a way as to hide the identity of the instigating origin host. However, there are also fraudulent uses of post-send spoofing as well as some "grey" situations where legitimate mechanisms have the potential to be employed fraudulently. We consider examples

of each of these below.

A general point is that regardless of whether it is legitimate or fraudulent the outcome of post-send spoofing is the same: the alteration of certain values in network data impedes the ability of a recipient host to identify the instigating origin of that data. It makes no difference if the spoofing was intended or not; in cases of legitimate post-send spoofing the user is typically agnostic to the fact. Thus even legitimate post-send spoofing can be problematic when the need to identify the instigating origin of network data arises.

2.4.1 Post-send spoofing of user data

The user data refers to data from the Application layer of the Internet model, as depicted in Figure 2.2. As such it is not possible to foresee all circumstances in which this might be altered, legitimately or otherwise. Thus and as in our examination of sender spoofing of user data, the following should be taken as representative examples and not an exhaustive list.

One legitimate use of post-send, user data spoofing arises from the special treatment required by the File Transfer Protocol (FTP) when Network Address Translation (NAT) is employed [42, 25]. We will examine NAT and similar mechanisms in our discussion of post-send service data spoofing below but mention the case of FTP over NAT here as FTP is user data.

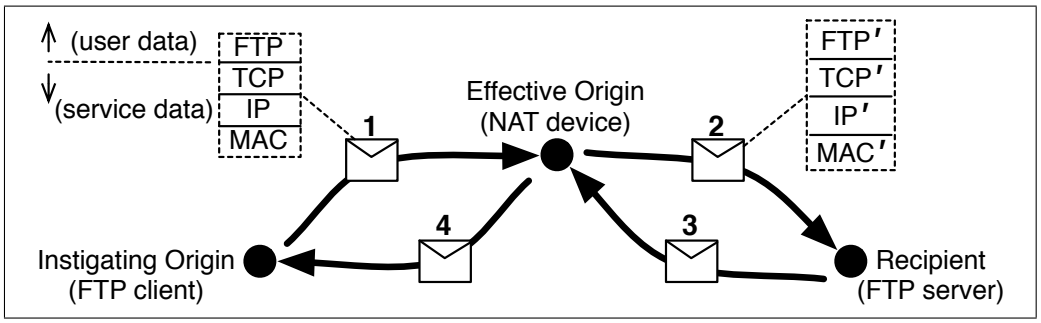


Figure 2.6: Post-send spoofing with changes to user and service data for active mode FTP behind a NAT device

In brief, in active mode FTP the client instructs the server to initiate the TCP connection over which the file transfer will operate. This instruction is communicated with a special control message (the `PORT` command) which informs the server of the IP address and TCP port to which to connect (on the client side). However, the client IP address specified in the instigating origin's (FTP) user data is non routable when NAT is in use and so the server will not be able to initiate the TCP connection. Thus, the NAT device needs to alter the control message sent to the server by replacing the specified IP address and TCP port with those being used by the NAT device for the file transmission.

Figure 2.6 shows the order of operations and how changes are made to both user and service

data. As far as the FTP server is concerned, it is the NAT device that has requested an FTP connection; in fact the server is completely oblivious to the existence of the instigating origin host. This phenomenon is not exclusive to FTP and can occur in any protocol that provides service data identifiers (e.g., IP address or TCP port) within the user data, including the H.323 Internet Telephony protocol ([25] - Chapter 5, Section 5.6.2 - "NAT - Network Address Translation").

Another example of post-send, user data spoofing is exhibited by chained Secure Shell (SSH) connections. The SSH protocol allows secure communication between network hosts (achieving confidentiality, integrity and authentication) [43]. As such it is often used to allow remote users access to machines in their home network (e.g. a corporate or campus network). A gateway host will typically run the SSH daemon process that accepts incoming connections from remote hosts. Once a user has completed a successful login using their account details, a second SSH connection can then be established to another host within the local network (e.g. the user's own office desktop).

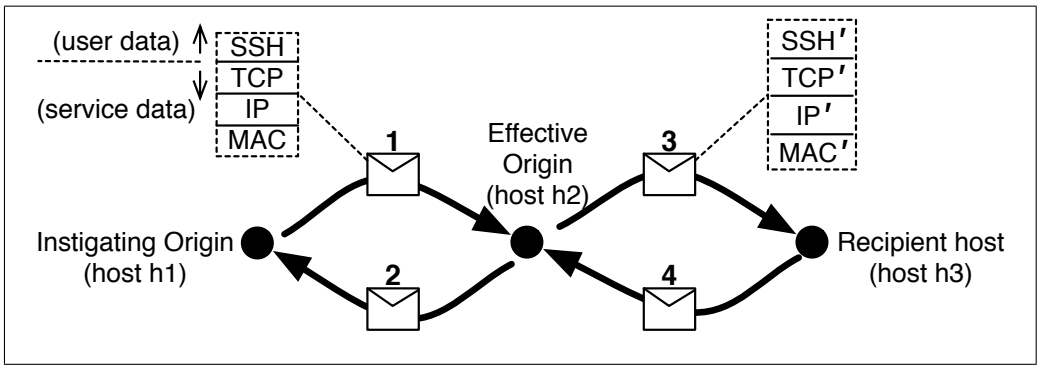


Figure 2.7: Post-send spoofing; host h1 establishes a chained SSH connection to host h3 through host h2

This scenario is depicted in Figure 2.7; a remote user at host h1 first performs an SSH login to host h2. Once logged in the user can perform commands on host h2 (depending on their account privileges) using the provided interactive (pseudo) shell. Thus the user issues the command to perform a second SSH login from h2 to the user's own machine, host h3. The SSH user data that flows between h1 and h2 ($h1 \leftrightarrow h2$) is not the same as the SSH user data between h2 and h3 ($h2 \leftrightarrow h3$). When the user issues a command at host h1, this is encrypted using the session keys agreed with host h2 to produce the data $h1 \leftrightarrow h2$. Once this arrives at h2 it is decrypted and then re-encrypted using the keys agreed between h2 and h3 to produce $h2 \leftrightarrow h3$ [43]. Thus the user data received by host h3 was actually produced at host h2 (the effective origin), even though the instigating origin of that data was host h1.

A similar situation arises from an extreme form of post-send spoofing, known as mix networks [44]. These are particularly successful in hiding the identity of communicating parties, as this is in fact their raison d'être. In mix networks the data transmitted by an instigating origin host will be passed through any number of proxy relays as depicted in Figure 2.8. Each of these makes a new

network connection to the next over which to forward received data until this is eventually delivered to the intended recipient. If confidentiality is provided with encryption of data between each pair of relays, as is the case with the "Tor Onion Router" [45] then the user data will be altered at every relay. In the example depicted in Figure 2.8, the instigating origin host h1 issues an encrypted `HTTP GET` request intended for the web server s1. At each relay host, the user and service data values are altered. The last relay host, h4 becomes the effective origin and issues the cleartext `HTTP GET` request to s1. Thus as far as the web server is concerned, the given user data originated at host h4.

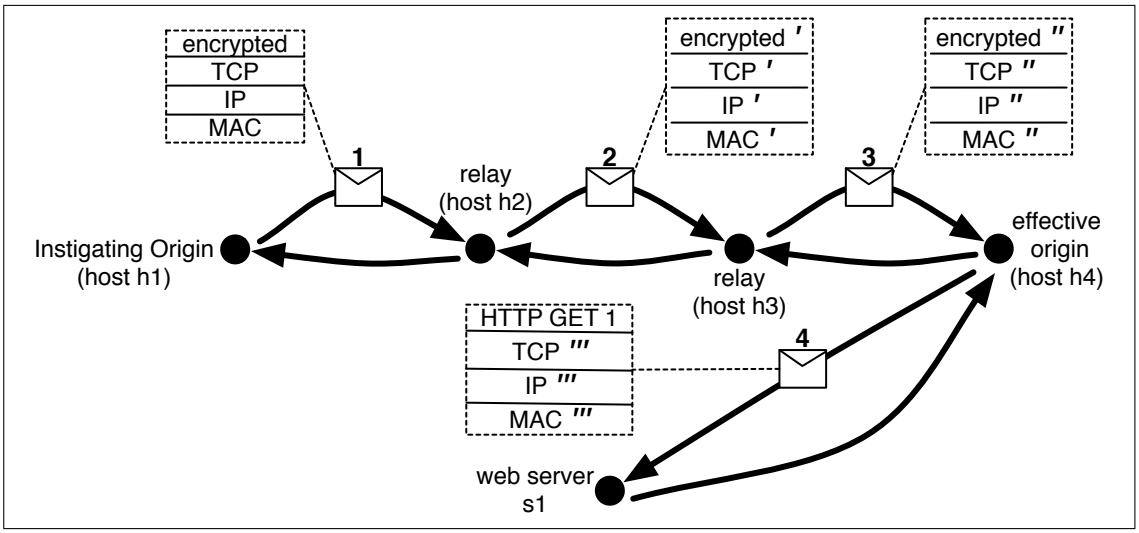


Figure 2.8: Mix network. The host h1 issues an encrypted HTTP request which is altered and relayed through a number of intermediate hosts before being delivered in cleartext to web server s1.

Recall that our definition of spoofing requires that alterations to network data result in the obscuring of the instigating origin's identity. In the case of both SSH chains and mix networks the user data transmitted by the instigating origin is encrypted. Thus, in any case we cannot rely on this user data to identify the originating machine. However the fact that the user data is altered in SSH chains or mix networks, albeit from one encrypted form to another still impedes our ability to track that data back to the instigating origin. It becomes difficult for instance to correlate the user data seen at different points of the network and thus reason about its origin.

The case of SSH chains is an example of a "grey" scenario where it is not clear if spoofing is fraudulent or legitimate. It is possible for this legitimate mechanism to be used fraudulently such as if an attacker uses the compromised account details of another user. In such cases the attacker may assemble a number of compromised accounts to construct a connection chain, from which to launch a network attack. This would provide the attacker with a number of 'layers' of anonymity (i.e. through each compromised host). We will consider connection chains and mix networks again in our discussion of post-send service data spoofing.

2.4.2 Post-send spoofing of service data

Two well known and widely used examples of post-send, service data spoofing are related to resource provisioning in Local Area Networks (LAN). We have referred to one of these, Network Address Translation (NAT) above and will be considering it further here.

In a LAN, a number of hosts may share a smaller number of publicly routable IP addresses through the use of Network Address Translation [22], and repetitive requests for commonly accessed web pages are minimized with a proxy web-cache [25]. Here we examine specific modifications made to service data under NAT, noting that the techniques employed by a web-cache are similar.

Briefly, when a client host sends an IP packet to the NAT process (which typically resides in the local network's Internet gateway router), the source TCP port and IP address are replaced. The NAT device selects an available TCP port (one that is not currently in use) and creates a connection to the intended destination from this port, using its own IP address. Local state regarding the origin host's IP address and source TCP port is stored for the duration of a connection, so that replies received at the NAT device's chosen TCP port are correctly forwarded.

Figure 2.9 provides a conceptual view of how NAT operates. The two local hosts, h1 and h2 are both assigned private (non-routable) IP addresses by the local DHCP server (not shown). Host h1 issues the request `HTTP GET 1` that is addressed to web server s1. The NAT device needs to alter the source IP address used by h1 with its own, routable address `IP.addr:9`; furthermore, the NAT device selects an available TCP port (port 1025 in Figure 2.9) and uses this to communicate with the web server s1. It should be noted that in the example shown and unlike the FTP over NAT scenario described above, alterations are limited to the service data. The instigating origin's user data (HTTP in this case) is forwarded unaltered.

As far as s1 is concerned the request `HTTP GET 1` was sent by the NAT device, which is the effective origin. Figure 2.9 also shows the state table maintained by NAT which allows it to correctly deliver any response packets. The TCP port to which the replies from each web server are addressed is used as a primary key in the state table, to reveal the address and TCP port used by the instigating origin; the reply packets are then altered accordingly and finally delivered to the local host.

Due to its legitimacy, this is easily the most common type of spoofing. However, this type of post-send spoofing poses as big a hindrance to identifying the instigating origin origin of network data as sender spoofing, even more so due to its pervasive use. Almost every home wireless router will provide NAT services, allowing a single broadband Internet connection (and IP address) to be shared amongst a number of devices. Indeed, the inability to correctly identify the instigating origin of IP packets that come from a network using NAT was tested in a recent Californian court case (in 2006). Tammie Marson was alleged to have downloaded copyrighted material in proceedings brought by Virgin Records America Inc. [46]. However, the case was dismissed as it could not be proven that it was Tammie Marson's computer which had downloaded said infringing material. The

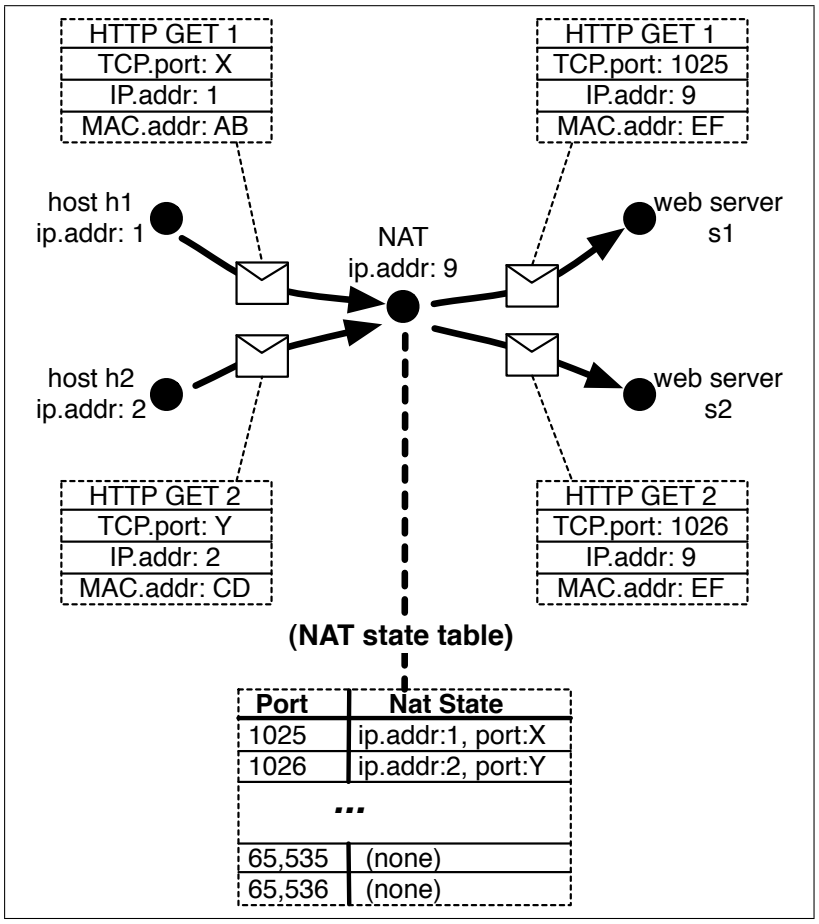


Figure 2.9: Network Address Translation

defense centered on the fact that the defendant's home network was unsecured, with NAT in place. Furthermore, her capacity as a cheerleading coach meant that any number of girls frequenting her house could have been the guilty party, which has lead to this important precedent being termed "the cheerleader defense" [47].

Nonetheless, the degree to which NAT will obscure the instigating origin is bounded by the number of hosts in the local network. If the mechanisms by which data is altered are legitimate, then we can take the altered data values as being true. Thus, we can at least identify the effective origin's network; in the case of NAT the instigating origin is contained within this network. In Figure 2.9 for instance, each of the recipient web servers would be able to identify that the instigating origin came from the network of the NAT device with IP address '9'. Furthermore, the instigating origin in this hypothetical scenario could only be one of two hosts (h1 or h2). Of course there are many real world deployments such as corporate or campus networks in which the instigating origin may be one of thousands.

In the case of sender spoofing to service data discussed in Section 2.3.2, an attacker can ensure that the source address in received IP headers is untrue. Thus, unlike in legitimate post send spoofing

the attacker would reveal nothing about their identity, not even their home network. This can be related to a notion of effort and associated reward; one can say that sender spoofing requires greater effort, with manual modifications to transmitted service data, but provides the greatest degree of anonymity. However, where post-send spoofing results from legitimate network processing, no effort is required by an attacker, though resulting in a lesser degree of anonymity. Furthermore, attacks that employ sender spoofing of service data are generally not possible when network-legitimate spoofing is in place, as the attacker's data is simply overwritten. As can be seen in Figure 2.9 if the host h1 had used sender spoofing to alter its IP address to that of host h2, the NAT device would overwrite this value with its own IP address.

We considered how mix networks can cause post send spoofing of user data if encryption is used between each pair of relays 'in the mix'. However in mix networks it is the alteration of service data values that poses the greatest hindrance to identifying the instigating origin. Mix networks are inherently fraudulent; their purpose is to intentionally hide the identity of communicating parties. This can of course be used for benevolent purposes such as in circumventing restrictions to Internet data mandated by government (the "great firewall of China" is one example). As depicted in Figure 2.8, each relay host in the mix will make a new network connection to the next, meaning that post send, service data spoofing occurs at every stage. The recipient host cannot rely on the service data (e.g. a source IP address) to identify the instigating origin, as this will belong to the effective origin.

Finally we consider a fraudulent form of post send, service data spoofing that is often associated with Denial of Service (DoS) attacks. In the relevant network security literature which we present in the next chapter, a series of successive compromised hosts, through which an attack is launched is called a 'connection-chain' [24]. Instances of this form of spoofing have been likened to to the "laundering" of network connections [48]. That is, an attacker assembles a number of compromised hosts and iteratively connects from one to the other (for instance using the SSH protocol) to eventually launch an attack from the last host 'in the chain', the effective origin.

Interestingly, though the problem of sender spoofing was recognised earlier [49, 50], it is this form of post send spoofing for which a solution was first proposed by Chen *et al.* Their seminal work [24] introduces the term 'connection chain', without however explicitly distinguishing which data is altered (i.e. service or user or both). Rather, it is assumed that across the connection chain data will be largely invariant "once protocol details are abstracted out". That is, they consider that once transmitted from the instigating origin, changes are restricted to the attacker's service data, as is reflected in their "thumbprinting" approach. We will further consider their proposal and other approaches to connection chain traceback in the next Chapter.

Other proposals dinstinguish whether an "attacker's communications" are fundamentally transformed or not when passing through compromised hosts [48]. We interpret this as referring to user data, given that service data is always transformed with changes at least to the Network layer IP

address. The authors of [48] distinguish between a "stepping stone", through which data flows "unchanged in essence" and a "zombie" which "fundamentally transforms" data. A similar distinction is adopted by Mirkovic *et al* [8], who use "agent" and "zombie" interchangeably, and introduce the term "handler" in reference to a stepping stone as in [48].

There are two distinct scenarios which we must consider using our notions of user and service data. The intermediate compromised hosts may relay an attacker's user data, making changes only to the service data in a manner similar to NAT or a web proxy as depicted in Figure 2.9. However, it is also possible for the attacker to direct the compromised host to create its own user data for transmission, as can occur in an SSH chain depicted in Figure 2.7.

It is not possible to generalise the circumstances under which changes will be restricted only to service data, as it will depend on the specific attack. Furthermore, as is pointed out by Mirkovic *et al* [8] an attack will likely involve elements of both scenarios, with the attacker employing handlers to relay commands to the agents, from which attack packets are generated. This is characteristic of attacks involving large numbers of compromised hosts, which act in coordinated unison to launch a large scale Distributed DoS. Instances of this type of spoofing are particularly successful at hiding an attacker's identity, though their feasibility and success will depend on the attacker's knowledge and determination in recruiting the required (compromised) hosts.

2.5 Chapter Summary

This concludes our definition and discussion of spoofing. We have distinguished where alterations to network data occur, that is, in the instigating origin host as in sender spoofing or, in an intermediate host (the effective origin) as in post send spoofing. In each case we have examined alterations at each layer of the Internet model using our generalisations of user data and service data. We saw that sender spoofing is always fraudulent and considered the factors that affect the way this type of spoofing might be maliciously employed with examples of known network attacks. However we also considered how post send spoofing may occur due to legitimate and widely employed network mechanisms.

Invariably and regardless of the intent to defraud, the effect of spoofing is to obscure the origin of received data, as this is observed at the recipient host or (assuming an appropriate vantage point) within the network itself. We now turn to a discussion of various proposals that aim to trace the origin of network data when sender or post send spoofing has been employed. Finally Figure 2.10 summarises the placement of each type of spoofing within this Chapter to aid the reader.

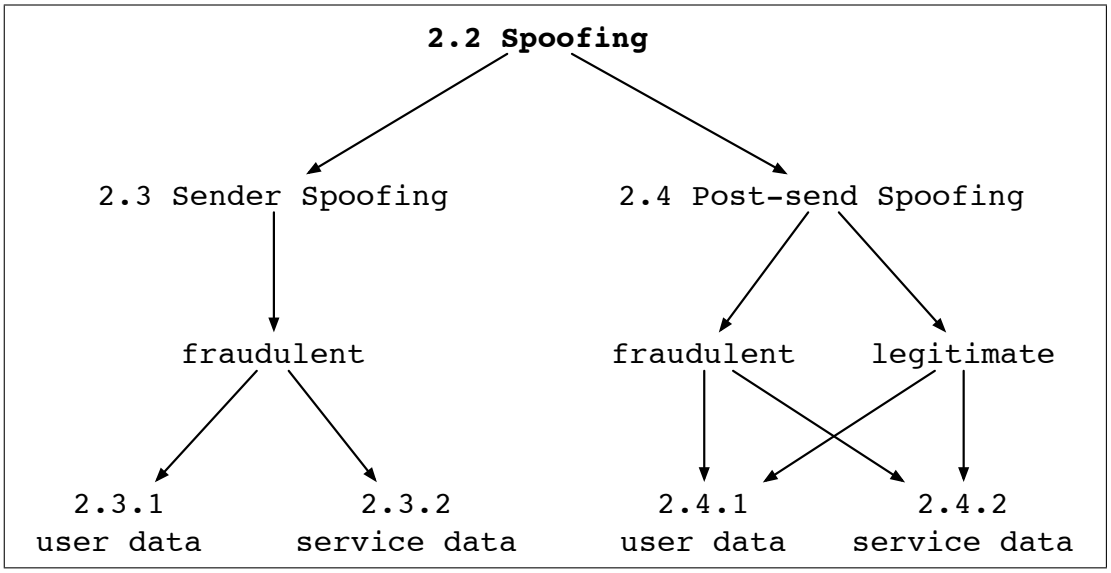


Figure 2.10: Idea map for chapter 2 showing the main types of spoofing and where these are discussed

Chapter 3

Elements of message traceback

3.1 Introduction

This chapter presents proposals that address the issue of identifying the instigating origin under conditions of spoofing as defined in chapter 2. Whilst some proposals deal with detecting the presence of or trying to prevent spoofing, most of the work considered here aims to "traceback" to the source of network traffic. Traceback means that one tries to determine the instigating origin host of some network data in spite of any alterations of that data.

Our definition of spoofing identified the two main types, as summarised in Figure 3.1. There are also two main areas of traceback research addressing each of those two types of spoofing. Generally, sender spoofing is addressed by 'IP traceback' and 'layer 2 traceback', whilst post-send spoofing is addressed by 'connection chain (or stepping stone) traceback', as in Figure 3.1. We introduce the term 'message traceback' to describe any system that aims to find the instigating origin of network data regardless of alterations to that data - whether sender or post-send. Our own systems contributions are in the area of layer 2 traceback and are presented in the next chapter.

No single approach to message traceback can claim to address all instances of spoofing. Snow *et al* explain that determining the identity of an attacker may require (up to) three stages [9], including IP and Layer 2 traceback, and where necessary connection-chain traceback.

As we will see, IP traceback can reveal the instigating origin's "first hop" router, at which point layer 2 traceback aims to reveal the instigating origin itself. However, IP traceback systems may fail if the traced data is altered at intermediate hosts. In such situations connection chain traceback aims to correlate connections across the network that are part of the same connection chain.

We begin our discussion by presenting IP traceback and layer 2 traceback as these are most relevant to our own systems work presented in the next chapter. We then conclude with a discussion of connection chain traceback both for completeness but also as proposals in this area both influence and are influenced-by IP traceback proposals.

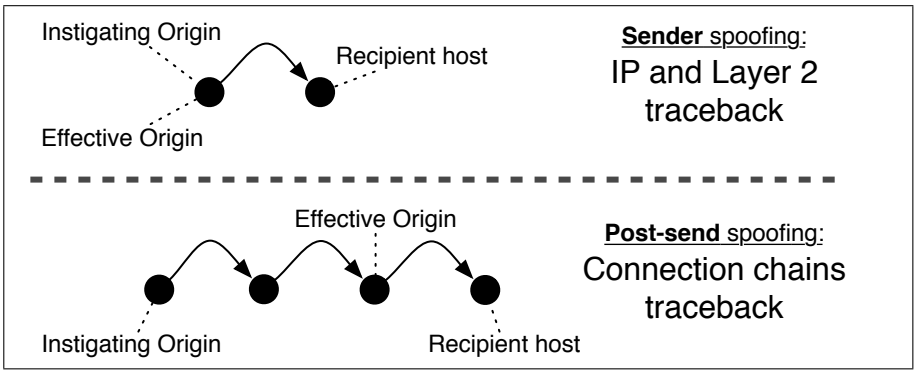


Figure 3.1: Summary of the main areas of message traceback research and the type of spoofing that these address

3.2 IP traceback

The majority of IP traceback proposals attempt to either log or insert marks into IP packets as these are forwarded by IP routers. We group such proposals and state that these follow the 'packet accounting' paradigm. That is, they try to 'account' for IP packets by recording a packet's identity, based on its content, as well as its route, based on which routers forwarded that packet. This record of identity and route is then used to successively traceback a packet through the routers that forwarded it.

We note that the two approaches of marking and logging have different emphasis, with regards to recording packet identity and route. When marking, the route is explicitly recorded by inserting each router's unique identifier into packets. There is no need to record the identity of each packet as this is provided by the marked packet itself. However, when logging packets, one must explicitly record the identity of each packet and the route is deduced by querying visited routers. This difference in approach is visualised in Figure 3.2 which shows the two main approaches to IP traceback and whether identity or route is explicitly recorded in each case (with the other captured implicitly).

By accounting for packets one can traceback a given packet after this has been delivered and is no longer being forwarded through the network. The duration during which a packet can be traced back after it has been processed by a given router is termed the 'traceback window'. This will inevitably depend on available storage resources but also on the efficiency of a particular traceback method.

With regards to sender spoofing, it is not possible or practical to expect a 'secure' TCP/IP implementation in all hosts to ensure the use of valid values in network data (e.g., ensure a valid source IP address is used). End user equipment is heterogeneous and IP traceback approaches typically treat this as 'off-limits'. Furthermore such an approach would not solve the cases of post-send spoofing as the valid values inserted at the instigating origin would simply be overwritten by the values inserted at the effective origin.

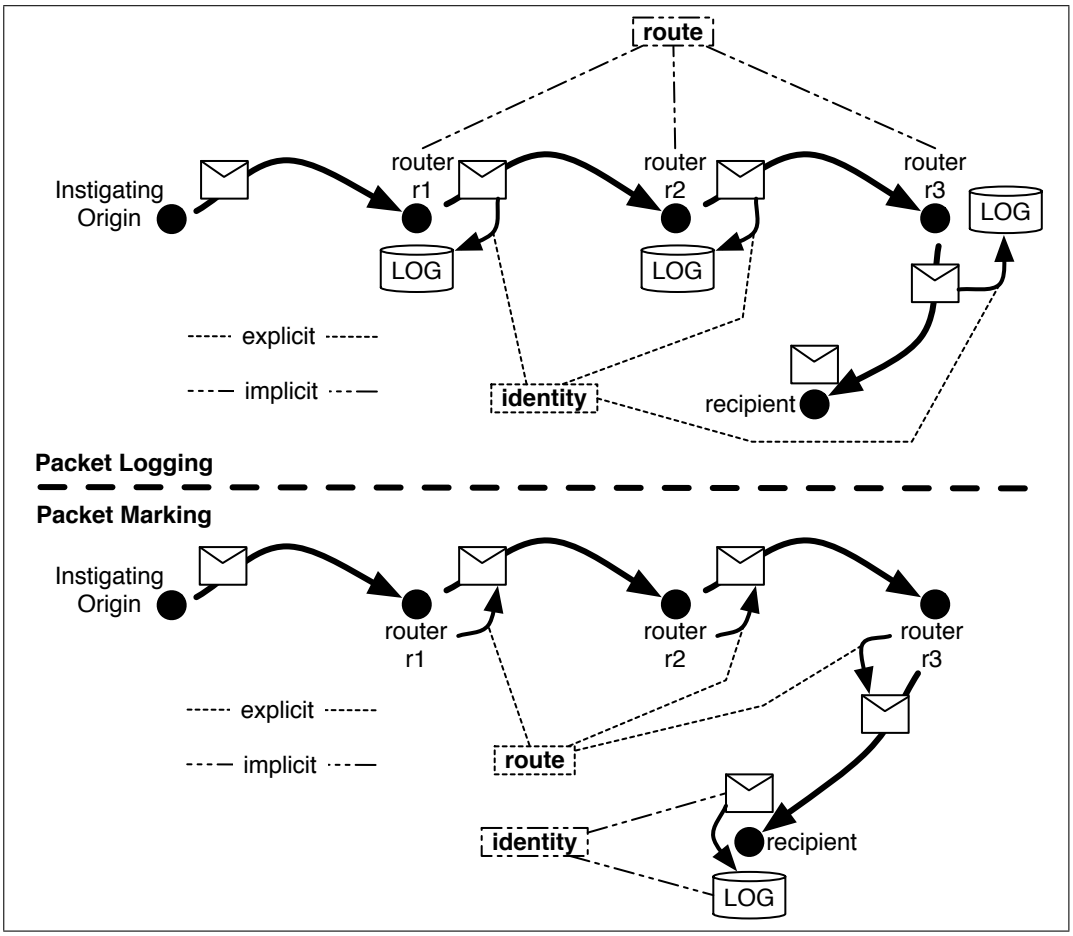


Figure 3.2: Accounting for packets by capturing identity and route; packet logging captures packet identity and packet marking captures packet route

We will now consider proposals that account for packets (packet marking, logging and hybrids therein) before presenting some interesting work related to spoofing detection, prevention and spoofing attack mitigation.

3.2.1 Packet Marking

The core idea of packet marking is for participating routers to insert a unique identifier into packets, before these are routed to their destination. Thus, a recipient host can use the marks to recreate the packet's route through the network. This will ultimately allow discovery of the first (marking) router to have handled a given packet, termed the instigating origin's 'first hop' router. Two central challenges are how marks are to be carried within a packet and how to prevent degradation of router throughput as a result of the additional per packet processing.

Savage *et al* were the first to propose Probabilistic Packet Marking [1]. In order to maintain router performance, only a subset of all packets traversing a router will be marked, governed by some

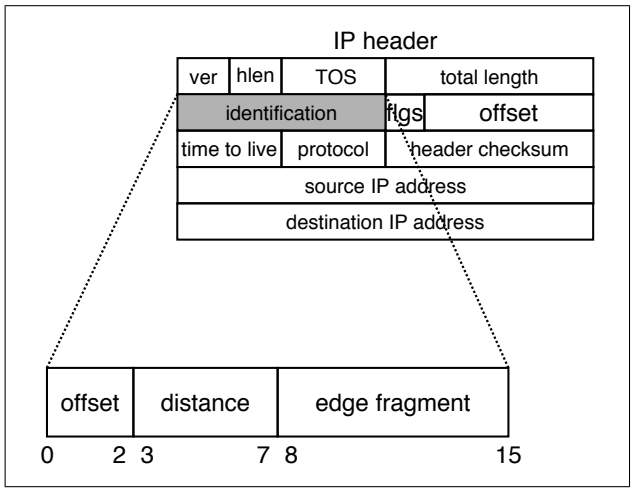


Figure 3.3: IPv4 header showing how the IPID field is utilised by the Compressed Edge Fragment Sampling scheme (this diagram is from the CEFS paper [1])

probability. The authors suggest use of the IP Identification (IPID) field to hold the marks, providing 16 bits of storage utilised as shown in Figure 3.3. The proposed encoding scheme, Compressed Edge Fragment Sampling (CEFS) uses the link between two routers, called an edge, as the packet 'mark'. As the IPID field does not provide enough space to hold an edge, each packet is probabilistically marked with partial edge information; a total of 8 marked packets are required to reconstruct each edge. This important early work pioneered the probabilistic packet marking (PPM) approach and addressed the two most obviously pressing issues central to packet marking: marking space and router performance. For example, the most frequent operation required of marking routers is to increment the 'distance' field of the CEFS scheme, which occurs in bits 3 through 7 of the IPID field. The alignment of this field with the IPv4 TTL field decremented at each router results in no changes being required to the header checksum.

Very shortly after CEFS, Doeppner *et al* proposed 'router stamping' [51] where each router inserts the IP address of the previous router (i.e., from which a given packet was sent). The authors suggest that the IPv4 header size be increased to accommodate packet marks. Schemes such as this that require changes to existing and widely used protocols are difficult to implement.

Song and Perrig proposed the Advanced and Authenticated Marking schemes [52]. Whilst they acknowledge the "promising solution" of CEFS and its acceptable router overhead, the authors introduce and address 3 additional and important challenges for PPM. The first is the computational burden on the recipient host in recreating a packet's route from edge fragments, in the face of a Distributed DoS. The second is that in such DDoS conditions the reconstruction process produces a large number of false positives. The Advanced marking scheme [52] addresses these issues, requiring however that the recipient has a view (map) of the entire network a priori. The third issue is protection from compromised routers, and this is addressed by the Authenticated marking scheme

[52]. Along these lines and around the same time, Park and Lee formalise the notion of an attacker spoofing the marking field [53].

Another proposal that considers false positives and the falsification of marks by a cognizant attacker is the 'randomize-and-link' approach by Goodrich [54]. Here, as with CEFS, each packet carries a 'message fragment' due to marking space constraints. However, rather than filling all available space with a fragment, some is instead devoted to conveying (part of) a checksum over the entire 'edge' being communicated. This serves to aid the reconstruction process whilst fortifying the scheme against malicious marking by an attacker. Goodrich uses 25 bits in the IPv4 header, in the fields identified by Dean *et al*, who develop a series of schemes for the "algebraic coding of paths". Besides the 16-bit Identification field, they also suggest using the 8-bit Type of Service field as well as 1 bit from the More Fragments flag. Again, the authors consider their scheme in the face of multiple attackers (i.e., DDoS) who can attempt to confuse the marking scheme.

Belenky and Ansari are the first to propose a deterministic packet marking (DPM) proposal [11], albeit with a somewhat relaxed traceback result. That is, for a given packet the authors aim to determine the point of entry to the packet marking enabled network. Thus, edge routers mark all ingress packets with the IP address of the recipient interface. A total of 17 bits are required in the packet header and as in other proposals the authors suggest to use the Identification field as well as the Reserved flag (the "evil bit" [55]). Jones *et al* [56] propose another deterministic marking approach that aims to discover a given packet's network entry point. Their approach is novel in that the 8 bit IPv4 Time-To-Live field is used. Edge routers set this to an arbitrary and configurable value that is unique to each router interface in order to convey the given packet's ingress or egress point.

In his study Adler [57] considers the tradeoff between the number of bits b used to mark each packet and the number of packets n required for traceback. He shows that most previous proposals require $b \geq \log n$. In the case of a single attacker Adler demonstrates the possibility of using $b = 1$ bit though requiring a great number of packets for reconstruction. In the case of multiple attackers such that packets follow k different paths it is concluded that b must be at least $\log 2k - 1$.

Yet another deterministic proposal is made by Velloso *et al* who suggest that each packet can carry its entire route [58]. Their scheme utilises an adaptation of the space efficient "bloom filter" [59] in the packet header. Though they find that 16 bits suggested in other proposals are insufficient for this purpose, they do not identify where their 'generalised bloom filter' (GBF) might be accommodated. In their short paper, Castelucio *et al* build on the GBF scheme and suggest a new "IP Traceback" community for the Border Gateway Protocol to allow peer networks to propagate their traceback capabilities amongst themselves [60].

Lai *et al* propose the novel 'Ant based' traceback where the marking procedure is initiated by the recipient of a malicious flow [61]. Starting at the recipient's first hop router, the traceback

packets (termed 'ants') use flow information maintained by routers to probabilistically determine which router they should visit next. The premise is that in DoS situations the large size of a given flow at a router will be indicative of its participation in an attack. Thus links with larger flows are assigned a higher probability so that more ants follow these. The ants record all routers they have visited until they reach the edge of the monitored network.

Shi *et al* propose a deterministic marking scheme for IPv6 [62]. Their approach is based on the encoding schemes by Song and Perrig [52] considered earlier. Each network link is assigned a unique 16 bit signature and the authors suggest the 20 bit Flow Label field of IPv6 to communicate these to the packet recipient, or alternatively an IPv6 extension header. A router marks all packets by performing the XOR of the egress link's signature with the current packet marking area so that a single packet will carry the whole route.

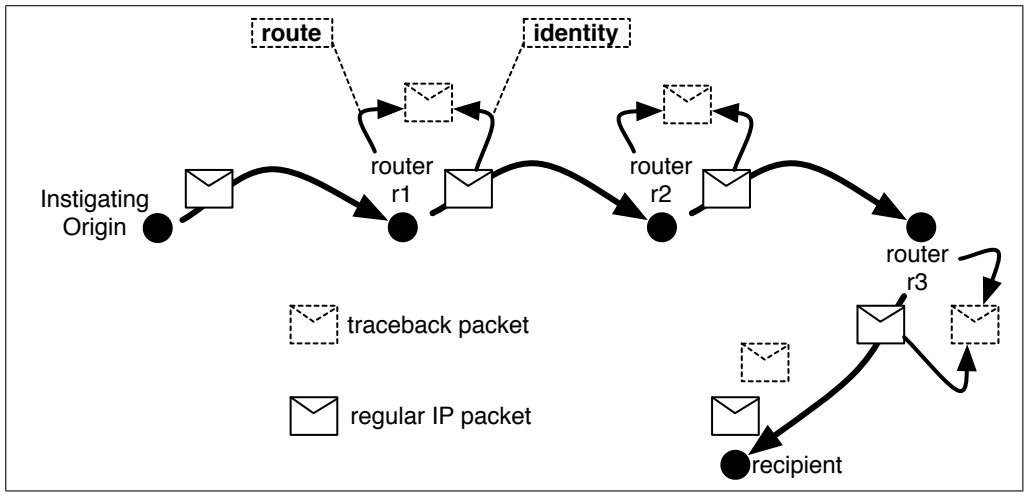


Figure 3.4: The messaging approach to IP traceback - both packet and router identity are captured and inserted into traceback messages

The use of additional messages rather than packet marks was first proposed by Bellovin and Leech [63] who defined a new type of ICMP message for this purpose. This is generally known as the 'messaging' approach to IP traceback and we briefly consider it here as marking and messaging are in-band and out-of-band signalling, respectively. In their proposal Bellovin and Leech suggest that routers probabilistically send traceback messages to a given packet's recipient, complete with authentication data, a timestamp and part of the original packet. Mankin *et al* [64] provided an improvement to this scheme which utilised an 'intention bit' in routing tables to better guide the (probabilistic) generation of traceback messages. Whilst this served to lower the number of traceback messages required for path reconstruction, Kuznetsov *et al* [65] note that the messaging approach would still require deployment at all routers in order to be useful. They suggest that aside from generating traceback messages, routers should also probabilistically insert their own IP address into the traceback messages generated by upstream (i.e., earlier) routers to 'fill in the gaps' when some

routers do not generate traceback messages.

In general, the messaging approach is at an inherent disadvantage. As was discussed, to account for IP packets one must identify each packet and its route. Whilst logging and marking need only explicitly record one of these factors as visualised in Figure 3.2, messaging must convey both. This is illustrated in Figure 3.4 where at each router the identity of a packet and its route (by way of the router's identity) is captured and inserted into traceback messages.

3.2.2 Packet Logging

In the packet logging approach to IP traceback, rather than signalling the route to recipient hosts as in packet marking, state is instead retained within the network for subsequent querying. Thus, rather than reconstructing the routes themselves, end hosts will dispatch traceback requests either directly to routers or by way of a traceback 'management' host. As with packet marking, the challenges that logging schemes face are processing overheads for logging routers and the amount of logged data. As we will see there is an inherent tradeoff in these two factors; improvement in one adversely affects the other.

The "Trajectory sampling approach" by Duffield *et al* [2] is one of the earliest proposals relevant to our discussion, though the authors do not use the term 'IP traceback'. Rather, the aim is to determine the paths followed by packets between the ingress and egress points of a monitored domain for "resource allocation and capacity planning". In trajectory sampling, a cryptographic hash value is computed over the invariant fields of the packet header and a part of the payload (in total 40 bytes) for each received packet. If the hash value falls within a predefined and configurable range, then the packet is logged as a 20 bit label - this is sufficient as the authors are not interested in reproducing packet content, but merely the 'presence' of a given packet. Thus by using the same hash function and range at all routers for a given time interval, packets are logged at all nodes or not at all, which improves correlation of logging compared to each router independently and probabilistically logging packets. The identification of invariant fields in the packet header is an important contribution of this proposal with similar fields being used by Snoeren *et al* in their SPIE proposal which we consider subsequently. Figure 3.5 shows as unshaded the fields with "high entropy" that Duffield *et al* use as input to the cryptographic hash function. As can be seen fields such as the IP version number and header length (HLen) are not used as they are considered to have low entropy - that is there will not be a great variation in these values in different IP packets.

The seminal Source Path Isolation Engine [3] (SPIE) is the most widely cited of logging based traceback proposals. The authors demonstrate the feasibility of a deterministic logging process where all packets are logged. This is accomplished through the use of Bloom filters [59]. A Data Generation Agent (DGA) is deployed at each router and is responsible for processing and logging packets. The DGA maintains a bloom filter for each time interval. A bloom filter is a one dimensional bit array,

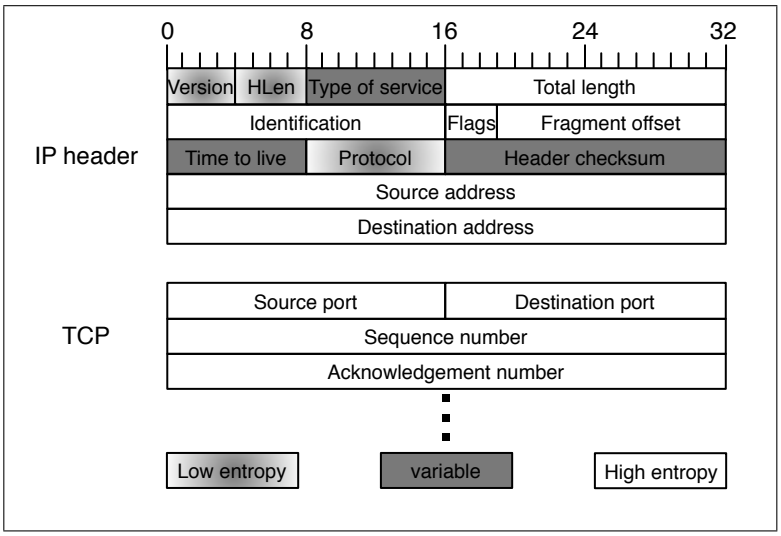


Figure 3.5: Invariant fields in the IP packet header identified by Duffield *et al* [2] - this diagram is recreated from a colour version in their paper

with all elements initialised to 0. The n bit output of a cryptographic hash function over each packet's content is used as an index into the bloom filter which is of length 2^n, as shown in Figure 3.6 (this diagram is taken from the SPIE paper [3]). The bloom filter elements identified by each of k hash functions over the packet content are set to 1. Thus, to check whether a given packet was inserted (a packet 'membership test') requires the computation of k hashes and confirmation that all specified elements in the given bloom filter are 1. The invariant fields used by Snoeren *et al* differ slightly to those identified in trajectory sampling above, with a total of 24 bytes used as input to the hash function (8 of which are from the packet payload). Our first proposal for a Layer 2 traceback system [15] is an adaptation of SPIE, which we will consider in more detail in the next chapter.

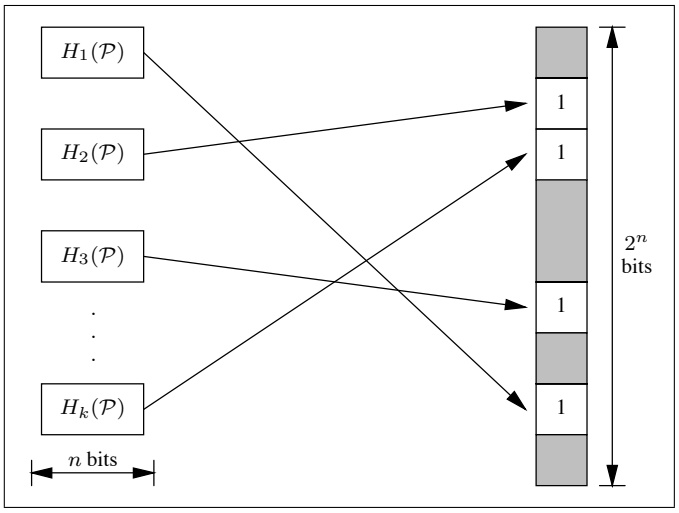


Figure 3.6: For each packet the n bit output of k hash digests is used to set elements into a bloom filter of size 2^n (this diagram is from the SPIE paper [3])

In their proposal Baba and Matsuda term the logging approach to IP traceback as 'hop by hop tracing' [66]. Routing nodes log packets together with their data-link level identifier (e.g., the MAC address of the recipient or sending router interface), reasoning that an attacker will have no control over the information inserted by forwarding routers. The authors use a "packet feature structure" to uniquely identify packets consisting of similar fields to those used in trajectory sampling and SPIE (the authors do not refer to those proposals themselves) as well as up to 20 bytes of packet payload. Storing packet data together with data-link level identifiers simplifies the traceback process. When a router is found to have processed a given packet, the 'next' router to be queried is explicitly identified, though the authors do not consider how to reduce logged data for efficient storage.

In their work, Shanmugasundaram *et al* introduce the hierarchical bloom filter (hbf) for "payload attribution" [67]. They argue that IP traceback based on logging of invariant packet content is of little use unless one has the entire IP packet to be traced. The hbf can accomodate tracing of "a reasonably long excerpt" such as 128 bytes of the original packet payload. The authors first introduce a 'block bloom filter' (bbf): each packet is split into 'blocks', each of which is appended with its offset before being inserted into a bloom filter (which is now a bbf). A collection of these forms the hierarchical bloom filter structure. At level 0 of the hierarchy, packets are inserted in blocks of size n whilst at level 1 the block size is increased to $2n$ until eventually the entire packet is used as input to a bloom filter. Thus, given a packet excerpt, the traceback procedure tries to establish a match at the lowest level before moving up to match as much of the packet as possible depending on excerpt length.

Lee *et al* consider consider 'scalable packet digesting schemes' [68]. Whilst they acknowledge the clever bloom filter technique of Snoeren *et al*, the authors predict that storage requirements for deterministic, per packet logging will continue to be a problem with increasing link speeds. They discuss the use of 'flows' as well as 'source-destination' grouping (using only the source and destination IP address) to lower storage requirements.

Zhang and Guan proposed the 'Topology aware' single packet traceback scheme [69] which tries to decrease the false positives inherent to bloom filters (and specifically as employed by SPIE). Given a number of bloom filter insertions the number of hash functions used for each packet's insertion and the bloom filter size can be tuned to produce an optimally low false positive rate. However as the number of insertions is dictated by the rate at which packets traverse a router (e.g., link speed) it is difficult to optimally tune these a-priori. The authors propose a "k-adaptive" algorithm whereby a router maintains a number of bloom filters, each using an increasing number of hash functions for packet insertions. At the end of each time interval the bloom filter producing the lowest false positive rate is retained whilst the rest are discarded. The authors adopt a similar approach to that of Baba and Matsuda, in including the layer 2 "predecessor information" (e.g., sending or recipient router interface MAC address) in logged data.

The issue of partial deployment is the focus of a study by Korkmaz *et al* who consider the scenario where only some Autonomous Systems are 'traceback enabled' [70]. They propose the Border Gateway Protocol BGP as a medium for communication between SPIE deployments and use simulation to show that logging based traceback can still be useful in a partial deployment. The authors consider the trade off between deployment ratio and traceback success rate in a number of network topologies.

Finally, Thing *et al* propose 'non intrusive' traceback for DDoS attacks [71]. The premise here is that packets between a given source-destination pair will follow a relatively static route as routing table updates are infrequent. Furthermore, the most popular web servers will have stable routes as they use reliable and well managed connections to the Internet. When the source address is spoofed then packets will pass through a non expected path (for that source-destination pair). Thus the authors advocate that routers build a cache of "valid source addresses" in learning mode. This consists of the source-destination addresses as well as a timestamp. The "white list" cache is then used for comparison to sampled traffic in operational mode. The authors use simulation to show that this scheme need only be deployed at border routers, though the generation of attack graphs from sampled data is not well explained.

3.2.3 Hybrids of the marking and logging approach

Advocates of a packet marking approach may argue that logging incurs an unacceptable storage and performance cost within the network. Similarly, advocates of packet logging will argue that marking incurs an even higher performance penalty at routers but also a performance and storage penalty at (resource poor) end hosts. Some proposals have thus taken a hybrid approach in an effort to achieve the 'best of both worlds'.

Li and Sung argue that though efficient, the SPIE system cannot scale to higher link speeds (such as OC768 at 39.8Gbit/s). Their 'One Bit Random Marking and Sampling' proposal (ORMS) [72] thus adopts a sampling approach to achieve an "order of magnitude smaller processing and storage cost" than SPIE, which deterministically logs packets. Their scheme utilises a single bit in marked packets which serves to improve the correlation factor between neighboring routers, in terms of which packets are logged. This bears some resemblance to the 'intention driven' messaging approach by Mankin *et al* [64] that we considered earlier, in which a single bit in routing table entries serves to better guide the probabilistic generation of ICMP traceback messages. In ORMS, for each packet received a router must check a single signalling bit; if this has been set to 1 by a previous router then the packet is stored into the bloom filter and the mark reset to 0. Otherwise, if the bit is 0 then it is probabilistically either set to 1 and the packet logged, or the packet is simply forwarded normally. The authors suggest that by sampling 3.3% of packets into a bloom filter with $k = 12$ hash functions, a memory utilisation as low as 0.58 bits per packet is achievable.

Another marking and logging hybrid is TRACK (rou*T*er po*R*t m*A*rking and pa*CK*et filtering) by Chen *et al* [73], who also make use of a marking flag. Interestingly the authors use a router interface as the 'atomic unit' of traceback (rather than the router itself). Each interface is given a (locally) unique 6 bit identifier, that is probabilistically inserted into packets before these are forwarded. In total 18 bits are required: 5 bits record the distance at which the packet was marked and are taken from the Time to Live field, 6 bits for the interface identifier, 6 bits hold the exclusive-OR of all interface identifiers encountered and a 1 bit marking flag. Routers also maintain a 'trace table' (the logging component) which records the hop counts, port identifiers and value of the XOR field from received packets, to be used during path reconstruction. In 'active' mode, for each packet received and if the marking flag is set to 0, then the packet is probabilistically marked with all aforementioned fields being updated, and the flag set to 1. In 'passive mode' however packets with no marking flag set are forwarded without modification. When the marking flag is set in passive mode, then only the XOR field is updated. A victim initiates the traceback procedure by creating its own trace table from received packets, before sending this to the first hop router, which eliminates those entries that did not participate in the packet's route. This is then forwarded to all second hop routers and the process repeated until the network edge. Border routers then return the trace table to the victim.

In their 'Distributed-log-based scheme' (DLS) Jing *et al* [74] build on their earlier 'Hierarchical Traceback System' (HITS) and suggest a HMAC enhanced HITS. The HITS scheme describes an architecture in which marking agents (MA) insert identifiers into forwarded packets, and Evidence and Collection Agents (ECA) retrieve marks from packets for logging. In their DLS, 29 bits are required in packet headers, 21 of which maintain a hash of the marking router's IP address and the rest hold the current value of the Time to Live field (as in TRACK this conveys marking distance). When a packet is to be marked (depending on some probability) and it has previously been marked, then a router will log the packet's mark fields before inserting its own values. Each log entry maintains the given router's IP address (and the HMAC of this), the HMAC from the marked packet, the packet's destination and a hop count. Furthermore, each entry also has a 'count number' recording the number of packets in a given flow (records needn't be maintained for each packet).

A similar proposal is made by Al-Dwairi and Govindarasu [75] with their "store, mark and forward" approach. Distributed-link-list traceback (DLLT) is the the first of two schemes developed, in which routers probabilistically store the current values from a previously marked packet into a local Marking Table before inserting their own IP address into a 32 bit marking field. Bloom filters are employed as in SPIE, however, the value of the first of the k hash functions in which a 0 bit is encountered is used as an index in memory which holds the marking information (the Marking Table entry) for the given packet. Thus, the packet requires an additional space beyond the 32 bits of IP address to convey this hash number. When a packet is not marked it is also not logged, and

the use of an extra 'marking flag' to improve logging/marking correlation is considered. The second scheme proposed is the Pipelined probabilistic packet marking (PPPM) which requires more than 40 bits for packet marks: a 32 bit IP address, an 8 bit Time to Live and a 'c-bit' value uniquely identifying the given packet. For each unmarked packet received, and when there is no current marking information to be conveyed to the destination, the router will insert is own values and forward. If a packet is already marked, then the current values are buffered locally, before this is re-marked. Marking processes are probabilistic and even when it is decided that a packet is not to be marked, locally buffered data (i.e., earlier marks) that are addressed to the given packet's destination are 'piggy-backed' using the usual marking fields.

Finally we consider the interesting Proactive Signalling Architecture (PSAT) by Fadlallah and Serhrouchni [76]. Interestingly their proposal is a logging and messaging approach in which IP flows are monitored and logged. The information maintained includes the flow information (source and destination IP addresses and TCP/UDP ports), a unique session identifier generated by the first participating router, and a list of routers encountered (termed 'Traceback Signalling Entities'). The authors define three types of messages: signal, query and response. The signal message is generated by routers from monitored IP flows, to convey to other routers that a given flow is of interest and should be logged. This contains the flow information, session identifier and a list of routers along the flow's path (IP address and HMAC of this) as well as a timestamp.

3.2.4 Attack Detection or Mitigation proposals

Finally we will consider some proposals do not follow the paradigm of accounting for packets (marking, logging and hybrids) but rather attempt to detect, prevent or mitigate the effects of sender spoofing. We briefly consider some of this work here to complete our overview of IP traceback research.

Templeton and Levitt explore a number of interesting detection techniques based on the use of probes [77]. These involve a probe packet being sent to the purported source of incoming packets, with subsequent replies (if any) checked for consistency with the original. They describe how the IPID (Identification) and TTL (Time to Live) fields of the IPv4 header might be used to detect a forged source address, e.g., the IPID of the replies should be greater than and close in value to those of the original packet. The notion of analysing the values of TCP/IP fields in network traffic to reason about that traffic's origin was first employed in the field of 'operating system fingerprinting' [78, 79, 80, 81]. A very interesting proposal for 'Physical Device Fingerprinting' [14] takes this approach further, by using the TCP timestamps option [82] to detect the unique clock skews of end hosts. That is, it "fingerprints" the machine itself rather than deducing what software it is running.

A simple though somewhat limited preventive measure was suggested by Ferguson and Senie with 'ingress and egress filtering' [83] at the entry point to a network. Traffic entering a network

(ingress traffic) should not carry a source address from within that network's address range. As shown in Figure 3.7 the packets sent by remote host $rh2$ are succesfully blocked as they carry the source address 192.168.1.2 which is used within the local network. Similarly, traffic exiting the network should only carry a valid source address, that is, one that exists within that network (egress filtering). In Figure 3.7 host $h1$ tries to send packets that carry a source address not currently in use within the local network and these are successfully blocked. However and as demonstrated in the diagram, one shortcoming of this approach is lack of protection from local spoofing attacks; local host h2 can still successfully masquerade as local host h1. Furthermore, remote host rh1 can still employ sender spoofing but not with an address used in the local network.

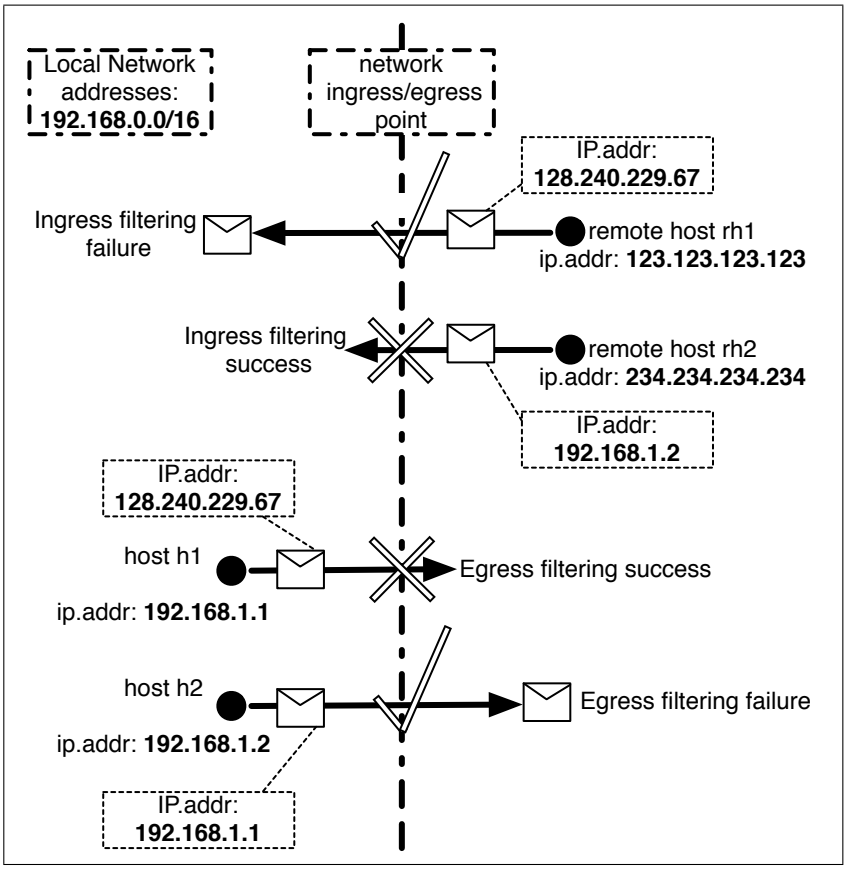


Figure 3.7: Ingress and Egress filtering

A relatively recent proposal by Wang *et al* advocates 'hop count filtering' as a defence against spoofed IP source addresses [84]. Their scheme is based on the premise that an attacker cannot forge the number of hops a packet takes in reaching its destination, *and* that this can be inferred from the IP Time to Live field (TTL). Hop count filtering (HCF) begins in a learning mode in which 'IP to Hop Count' (IP2HC) tables are constructed from established TCP connection data. In filtering mode, packets are dropped if there is inconsistency between the current values for a given source address and those stored in the IP2HC table. The authors acknowledge the problems posed

by NAT, as multiple hop counts may appear to originate from one host, as initial TTL values are not consistent across different Operating Systems (though the TTL used by a given OS is usually "well known"). They also propose address prefix clustering to reduce the memory requirements of the IP2HC table (so hop counts aren't maintained for individual hosts).

One of the earliest proposals that explicitly aimed to *determine the origin* of a DoS attack but which still did not follow the 'packet accounting' paradigm is the novel and potentially controversial 'Controlled Flooding' by Burch and Cheswick [4]. In the face of a packet flood, the authors suggest selectively flooding incoming network links with controlled 'bursts' of traffic. If a difference is seen in the attacking packet stream before and after flooding a given link, the attack is said to originate 'behind' that link. The process is then repeated upstream towards the attacker's network. Flooding of a link is achieved with the TCP character generator (chargen) service [85] which generates continuous data to a requesting party. It is interesting to note that the authors in fact employ sender spoofing of packets to accomplish the flooding. When testing the link between router 1 and router 2 ($R1$ and $R2$) and where $R1$ is closest to the victim network (from which flooding is instigated), service requests are sent to chargen servers beyond $R2$, but carrying the source address of $R1$, which will then receive chargen replies (and the link between the routers is flooded).

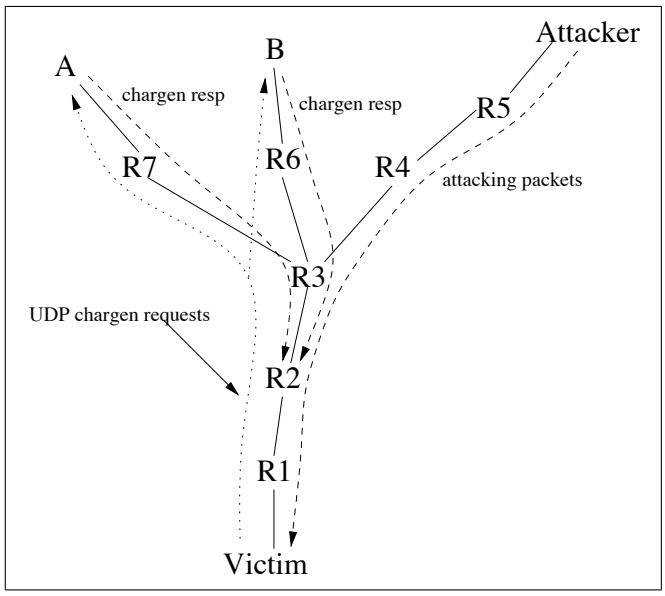


Figure 3.8: Controlled flooding - diagram is taken from [4]

Another early and important proposal is 'CenterTrack' by Stone [86] which as with Controlled Flooding above, requires a catholic view of the network a priori (this is not the case in many 'packet accounting' methods as we will see). In CenterTrack, an 'overlay network' is formed between all border routers and a dedicated tracking host in the network center (e.g., with the use of IP tunnels). When an attack is detected at the border routers the attacking packet stream is forwarded to the

tracking host, which can quickly determine the ingress adjacency; that is, depending on the border router forwarding the packet stream, we can determine which adjacent network the traffic originated from.

The last two proposals are important as they predate most other IP traceback work. However, they are inherently limited in requiring an ongoing attack stream in order to proceed the traceback (and not just a small number of packets or even a single packet). This also means that they cannot perform traceback *post mortem*. This is one of the important advantages of proposals that either mark or log packets (i.e., that follow the traffic accounting paradigm), but which of course comes at the cost of storage and performace overheads as we will see. Consider that the controlled flooding technique above is essentially stateless; no information needs to be stored either within the network or at end hosts.

This concludes our examination of IP traceback proposals which address the issue of sender spoofing by trying to traceback to the instigating origin. We have considered proposals that account for IP packets by recording each packet's identity and route, either through packet marking or packet logging (as well as hybrids therein). We have collectively termed these proposals as following the packet accounting paradigm. We have also given an overview of related work that attempts to detect, prevent or mitigate the effects of sender spoofing rather than trying to identify the instigating origin. We now turn to layer 2 traceback which also addresses sender spoofing and extends the traceback process to the instigating origin's home network.

3.3 Layer 2 traceback

Layer 2 (L2) traceback systems as with IP traceback above address the issue of sender spoofing. Generally L2 traceback aims to locate the physical point of connection within the instigating origin's home network. That is, given some received network data L2 traceback aims to locate the point at which this data was first processed by a networking device. In the case of switched ethernet for instance, L2 traceback proposals use a switch identifier (sID) together with the switch-port number (pNO).

IP traceback systems cannot themselves identify the instigating origin as invariably, packet accounting mechanisms (whether marking or logging) are deployed in IP routers. However, the layer 2 data (such as source MAC address) that may help to identify the instigating origin are not available as these are removed at the gateway of the origin's home network. This shortcoming of IP traceback is well known; Hazeyama *et al* state that [5] IP traceback reveals the origin network but not the origin host. Figure 3.9 illustrates that IP traceback can identify the instigating origin's "first hop" (or gateway) router, at which point layer 2 traceback (L2 traceback) aims to reveal the instigating origin itself.

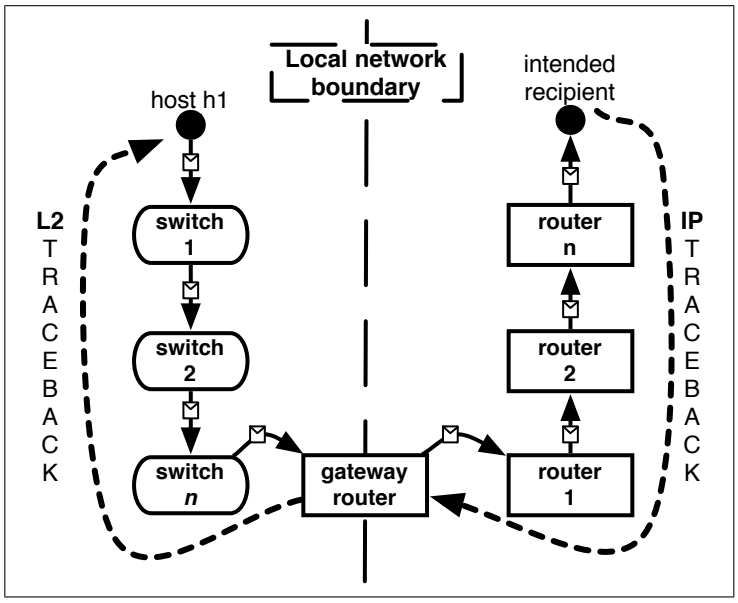


Figure 3.9: IP traceback identifies the instigating origin's first hop router. Layer 2 traceback extends the traceback process to identify the instigating origin within the local network

Our own proposals [15, 12, 13] are in the area of L2 traceback for switched ethernet. As such we defer a discussion of the challenges of this environment for traceback when we present our work in the next chapter. We now give an overview of some L2 traceback proposals noting that there has generally been less work in this area compared to IP traceback.

3.3.1 Layer 2 traceback proposals

Hazeyama *et al* [5] are the first to directly address the issue of traceback within the instigating origin network. They adapt the Source Path Isolation Engine (SPIE) [3] for deployment in a switched ethernet network. As introduced earlier, SPIE [3, 17] is a logging based IP traceback system where a Data Generation Agent (DGA) logs a hashed digest of each packet forwarded by a router, using bloom filters to achieve significant memory efficiency.

In [5] Hazeyama *et al* propose a Layer 2 extension to hash based IP traceback. They deploy their extended DGA (xDGA) in gateway routers. For each received frame the switch identifier sID is inferred from the MAC address of the recipient router interface (i.e., the frame's destination MAC address). By maintaining local copies of each switch MAC address table and given the sID the MAC-table of the given switch is used to determine the origin port number pNO based on the frame source MAC address. The sID and pNO are then concatenated with packet data before a message digest is produced and used to set bloom filter elements (Figure 3.6). This important work is the first to directly address the issue of traceback within the instigating origin network. However the deployment suggested by the authors requires a very restrictive network topology where all hosts are

separated from the network edge by only a single switch. This is illustrated by Figure 3.10 where in the case of host $h1$, the frame destination MAC address X allows one to infer that the instigating origin is on switch 1 as this is the only switch reachable on router interface 1. However, in the case of the frame sent from host $h3$ it is not possible to infer the instigating origin's switch. Thus it is not clear which switch MAC-table should be used for the port number lookup based on the frame's source MAC address. This constraint on network topology somewhat limits the applicability of this proposal in real world deployments.

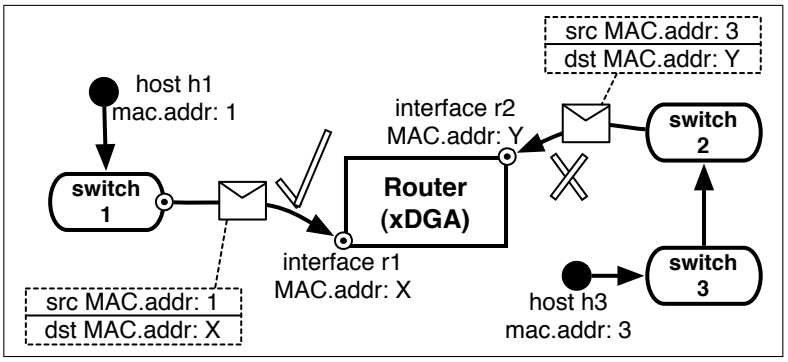


Figure 3.10: Deployment of xDGA in a Layer 2 extension to IP traceback [5]

In their more recent proposal, Hazeyama *et al* [87] deploy a single packet digesting tapbox that has a direct link to every switch and router in the network. Due to the problems of traffic visibility in the switched ethernet environment (which we address in the next chapter) they use N to 1 switch port mirroring to replicate all frames forwarded by each switch to the single digesting tapbox. The replicated frames are sent to a special monitor port on each switch, to which the packet digesting tapbox is connected. The digesting engine maintains a 'retrieve hash table', which maps a message digest to a source MAC address (taken from the given frame). When a traceback request is received, the message digest is computed and the retrieve hash table searched for a match. If this is found the appropriate source MAC address is used to query the switch forwarding databases of all switches (FDB) to reveal the origin switchport. In this paper they term Layer 2 traceback as "Intradomain traceback". This newer work [87] was published at the same time as switch-SPIE [15] and the two systems share some similarities in the use of port mirroring. However, this newer work still faces the same scalability issues as their earlier work [5] by using a single digesting tapbox.

We encountered TRACK (rou$T$er po$R$t mArking and paCKet filtering) [73] in our discussion of hybrid IP traceback systems (i.e. that use logging and marking components). Snow *et al* [9] build on that earlier work to propose Link-Layer traceback in switched ethernet networks. This scheme is also called TRACK (Tagged fRAme tracebaCK) and similarly adopts a hybrid approach with both logging and marking elements. Tags are applied to ethernet frames by an 'in switch' process. Each tag includes a keyed-Hash Message Authentication Code over the first 32 bytes of IP data

carried by its frame. A separate process in the 'Analysis and Collection Host' removes and logs the tags with links to a sorted 'host table' (each host is sID and pNO). This very interesting proposal trivialises a core process of other known solutions: establishing causality between a given frame and the originating switchport (pNO). TRACK assumes an 'in switch' process and we agree that this is the best vantage point for establishing pNO. However, implementing traceback 'in switch' assumes functionality that is (typically) not available.

Finally we consider two proposals that address the issue of Layer 2 traceback in wireless networks rather than switched ethernet. In [88] Huang *et al* and in [89] Kim *et al* explain the unique set of requirements created by a mobile ad hoc network (MANET) for layer 2 traceback of frames. For instance, there is no fixed network infrastructure, all nodes act as both hosts and routers, and no nodes are trustworthy. Furthermore, in such an environment the traceback result must be relaxed; rather than try to uniquely identify the instigating origin the traceback process terminates at the first encountered 'malicious node' (which we term the effective origin).

Huang *et al* [88] propose Hotspot based traceback for mobile ad hoc networks (MANET). Here the authors use the 'tagged bloom filter'. Whilst a 'regular' bloom filter as used in SPIE [3] is a 2^n bit array of single dimension (Figure 3.6), a tagged bloom filter is a 2-dimensional array of size 2^n by c. The c bits of the second dimension are used to store a relative time to live (RTTL) when setting elements of the bloom filter. The RTTL is calculated from the `Time to Live` (TTL) value in the `IP` packet header and serves as the tag for each frame. The premise is that well behaving routers will decrement the TTL by 1 and so this can aid the path reconstruction process. Kim *et al* [89] propose attacker traceback with cross-layer monitoring in wireless multi-hop networks. The authors focus on persistent attacks, that is, sender spoofing employed in flooding Denial of Service attacks as examined in Chapter 2. In this logging based approach, a normal profile is constructed for each node based on the cumulative frequency of observed data (e.g., number of frames). This is stored together with MAC and IP layer data in the abnormality table which is indexed by destination MAC address but also contains the previous hop MAC address.

3.4 Connection chain traceback

We now consider those proposals that aim to traceback to the origin of network data, when post send spoofing has been employed. As we saw in the previous chapter, post-send spoofing can be fraudulent as well as legitimate. The proposals presented here invariably address fraudulent post-send spoofing.

In the literature, a series of successive compromised hosts, through which an attack is launched is called a 'connection-chain' [24]. In most proposals the connection-chain is assumed to be formed via multiple remote logins through compromised hosts [6, 90, 91, 24, 92, 7, 93, 94, 95]. That is, the attacker performs an unauthorised remote login to a compromised service and account (e.g., Secure

Shell login if TCP port 22 is responding). From there the attacker performs the next remote login to the next compromised host and service. This is illustrated in Figure 3.11 where the attacker host $h1$ assembles a chain of compromised hosts and then launches a flooding Denial of Service (DoS) attack from the last host in the chain.

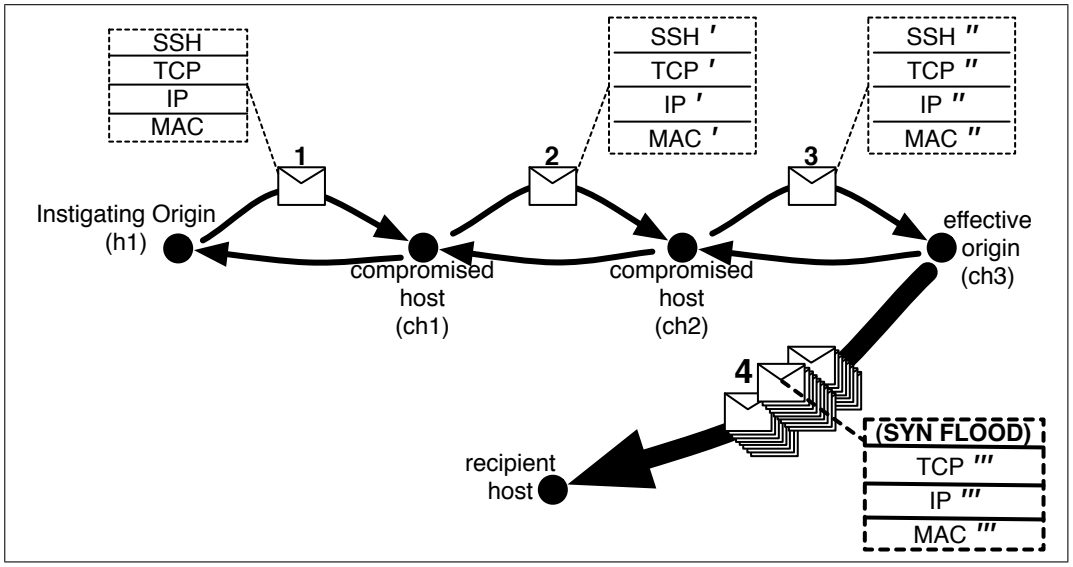


Figure 3.11: Post-send spoofing. A connection chain is formed and a DoS attack launched from the last host in the chain

The scenario depicted in Figure 3.11 implies interactive and bi-directional traffic from the attacker towards the connection chain (i.e., between hosts $h1$ and $ch1$). As we have seen, it is difficult for an attacker to employ sender spoofing when reply packets are required such as in establishing a `TCP` connection (`SSH` operates over `TCP`). Thus, *within* the chain the attacker's service data is assumed to be legitimate, in that the reported source IP address is the one currently being used by the sending host. However, service data flowing from the last host in the chain (the effective origin, host $ch3$ in Figure 3.11) and towards the recipient does not need to be legitimate. That is, the attacker can employ sender spoofing at host $ch3$ so that the data IP ‴ does not reveal this host's identity.

This subtle distinction perhaps explains why some connection chain proposals seem to question the utility of IP traceback systems that address sender spoofing [96, 97, 98]. It is often (unrealistically) assumed that attacks originating from a compromised host will carry the true source IP address of that host. If this is the case then it becomes possible to identify the effective origin, to serve as a starting point in locating the instigating origin. Other proposals however do recognise that IP and connection chain traceback are "discrete but complimentary" [94]. We would go further to say that they are both important components of an end-to-end message traceback system; in some cases, both are required in order to traceback to the originating machine or network as they address different elements of the same problem (sender and post-send spoofing).

The term 'connection chain' is introduced in a seminal paper by Staniford-Chen *et al* [24]. However, one finds the concept of a 'chain of intermediaries' in their earlier work which proposed a network security monitor for local networks [90]. This earlier work is characterised as similar to 'host based intrusion detection' [90] rather than connection-chain traceback. Furthermore, in [24] the authors recognise the applicability of systems such as the 'Distributed Intrusion Detection System' [93] to connection chain traceback. Thus, the foundations of connection-chain traceback can be directly traced to work in the area of Intrusion Detection.

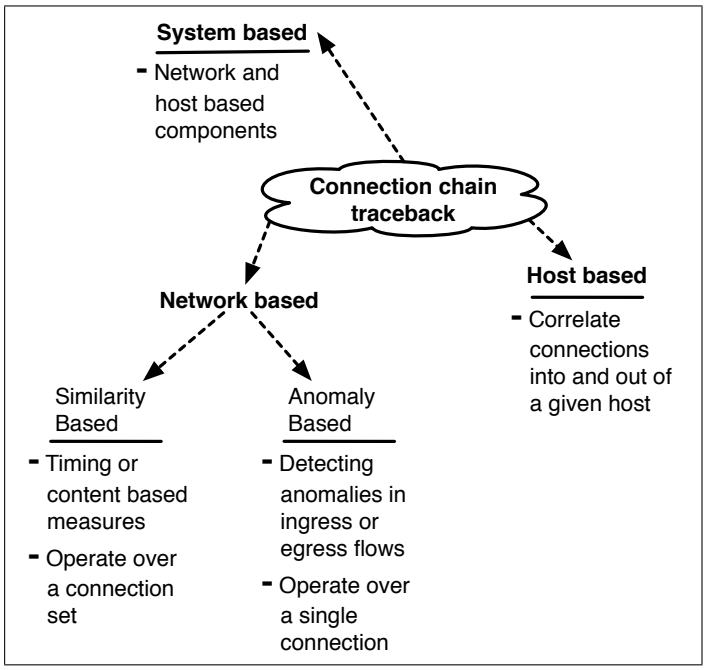


Figure 3.12: Thr classification of connection-chain traceback methods adopted by [6]

It is well established that connection-chain proposals fall mainly into the two categories of network or host based methods [91, 99], in reference to where these are deployed. Almulhem *et al* suggest a third 'system based' category for those proposals that require both host and network based components [6]. Generally, connection chain traceback systems aim to determine whether two connections are part of the same connection chain and do so whilst data is still flowing in the network. This differs from the traffic accounting paradigm adopted by many IP or L2 traceback proposals that allow traceback a posteriori. In what follows we give a brief overview of the approaches suggested in the literature, using the classification developed by [6] and which we summarise in Figure 3.12.

3.4.1 Network based connection chain traceback

We first examine the network based 'thumbprinting' method proposed by Staniford-Chen *et al* [24]. The general idea is to compute a summary based on the contents of each connection and then search

for the existence of this summary in other parts of the network. The thumprints are derived using the frequencies of characters appearing in the user data per unit time (their experiments use a 1 minute interval). The implicit assumption here is that user data will be invariant across the connection chain with only service data being changed at each compromised host (i.e., hosts in the chain simply relay an attacker's traffic). This most important work is not only the first proposal specific to connection chain traceback, but can also be seen to influence future IP traceback proposals which we examine in the next section. The notion of logging a summary version of network traffic, to allow subsequent reconstruction of attack paths is central to all logging based IP traceback systems.

A related network based proposal is made by Zhang *et al* who instead adopt a timing metric to distinguish connections [92]. They explain that content based measures such as 'thumbprinting' will fail in the presence of encryption. Rather, they concentrate on identifying "invariants" in network traffic, based on its intrinsic properties and not its content. They note a distinctive 'on-off' pattern in interactive network traffic, representing the transition from a user typing at a keyboard and then stopping. These transitions are detected and correlation between two connections determines if they are part of the same connection chain; traffic volume is also suggested as an alternative to timing. A main challenge to this kind of approach is the intentional (and malevolent) introduction of entropy to the specific properties being monitored (e.g., introduction of timing delays).

Yet another network based approach is proposed by Choi *et al* [91], but which does not seem to 'fit' with the classification given by [6] (summarised in Figure 3.12). Choi *et al* provide their own classification, and their 'Network based real time connection traceback system' (NRCTS) is termed as 'active network-based' (thumbprinting and timing based measures are termed 'passive network-based'). The approach used by NRCTS is to insert an easily detectable mark into reply packets, sent from the victim back towards the attacker. The true path back to the attacker's machine is then reconstructed by detecting the presence of marks at various points in the network. The marking approach is not unique to NRCTS but rather appears in the earlier 'Sleepy Watermark' proposal by Wang *et al* [95] (though one can see the foundations of this approach in the packet marking method of IP traceback, pioneered by Savage *et al* [1, 10]).

Before moving onto host based proposals, this seems an appropriate point to mention a relatively new and related marking scheme, that is not aimed at 'connection-chains' in the traditional sense of remote logins, but rather at tracking through "anonymous communication systems". We considered these earlier in our discussion of post-send spoofing and specifically mix networks such as Tor, where data is encrypted in addition to it being 'mixed' through a number of intermediate hosts. The 'DSSS-Based Flow Marking Technique' proposed by Yu *et al* [100] requires an 'interferer' to embed marks into the original data (before submission to the mix network) and a 'sniffer' which recovers the marks before passing the original data to its intended recipient. The marks are generated using the Direct Sequence Spread Spectrum technique, where "pseudo-noise code" is used to "spread the signal

over a bandwidth greater than the original data signal bandwidth". Though this novel approach is restrictive in requiring the modulator and demodulator at each end of the monitored communication, it successfully addresses the issue of traceback invisibility, when a marking approach is employed (the 'marks' in this case are not easily detectable).

3.4.2 Host based connection chain traceback

In general, host based approaches try to establish causality between the incoming and outgoing connections within the host itself. The implicit assumption that traceback functionality installed in a given host has not been compromised, even though the (compromised) host itself forms part of a connection chain, seems counter-intuitive. Nonetheless, Kang *et al* [99] adopt a process searching approach, which is predicated on the existence of a parent-child relationship between the process receiving an attacker's data and the process forwarding this onto the next compromised host (or to the attack's victim). The authors make use of existing Operating System tools (such as the linux 'lsof' utility) to search for a process's parent for establishment of causality; if the search ends with the 'super process' (`init` in linux), then the current host was the chain's origin. This of course assumes the attacker's machine participates in the traceback scheme, in which case this interesting approach is limited to controlled network environments. This is true for any approach that assumes host participation, including proposals in the system class which we consider next.

3.4.3 System based connection chain traceback

The system class refers to proposals requiring collaboration between various network or host based components. The earliest work cited in discussions of connection-chain traceback proposals is the 'Distributed Intrusion Detection System' (DIDS) proposed by Snapp *et al.* [93]. This scheme assumes that all hosts are "C2 or higher rated computers" which, as per the U.S. Department of Defense standard have "controlled access protection" with dedicated user logins and "security-relevant" auditing [101]. A host monitor running on each host communicates with the centralised DIDS director, attributing relevant events to the unique 'network-user identification' assigned to each user upon login. Network activity is scrutinised by dedicated LAN monitors which also communicate relevant events to the Director, that in turn forwards them to the rule-based 'expert system' for analysis. The DIDS system was not designed with connection-chain traceback as its primary goal, but is rather a broader security monitoring scheme for controlled network environments. Its applicability to the field of connection-chain traceback was identified retrospectively by Staniford-Chen *et al* [24] (who first introduced the term 'connection-chain').

Another early (system based) proposal that is often cited in the literature, but which like DIDS above does not directly use the term 'connection chain', is the Caller Identification System (CIS)

[7] by Jung et al. It is not entirely clear to us why CIS should be classed as 'system based' in [6], rather than 'host based'. Though the scheme is distributed, it is deployed only in network hosts, without central direction as was the case with the DIDS proposal above. Jung *et al* use an 'extended TCP wrapper' (ETCPW) which is an adaptation of the TCP Wrapper first proposed by Venema [102]. The ETCPW and the Caller Identification Server (CIS) are installed in all network hosts, and work together to verify the origin of inbound connection requests, before these are granted. A connection request from host $n-1$ to host n is intercepted by the $ETCPW_n$ (i.e., The ETCPW in host n), which makes an authentication request to (the local) CIS_n as is shown in Figure 3.13. Then, CIS_n queries CIS_{n-1} for the origin of the incoming connection request. The subsequent response from CIS_{n-1} indicates whether host $n-1$ was the 'home base' (instigating origin in our terminology) of the connection's originator or otherwise provides the list of hosts through which the remote connection request has passed (from a locally maintained table, updated by earlier queries and responses). In this latter case, CIS_n will independently query all of the indicated hosts' (local) CIS for the given connection request's origin, before granting or refusing access (by appropriately signalling the local $ETCPW_n$), and updating its own local 'host table'.

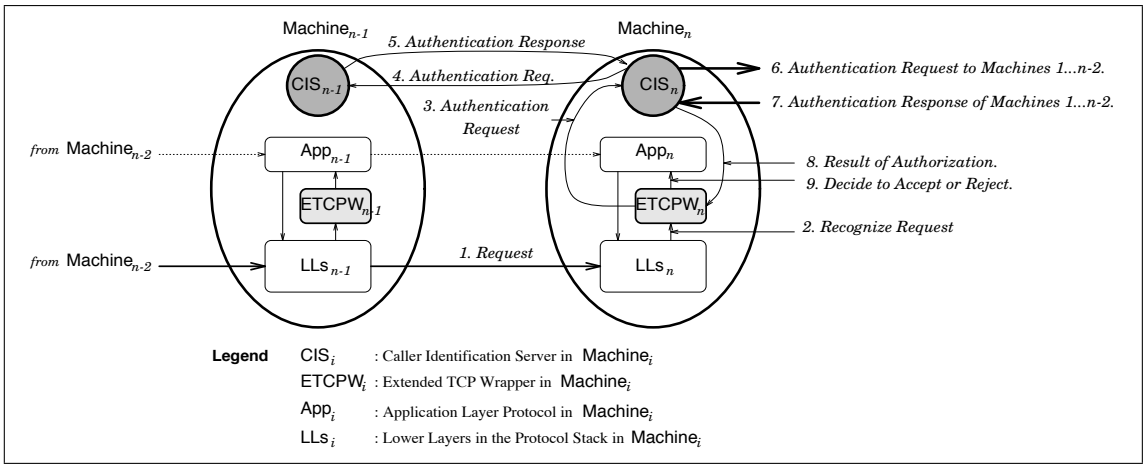


Figure 3.13: Typical sequence of events in the Caller Identification System - diagram is taken from [7]

The Session Token Protocol (STOP) proposed by Carrier *et al* [94] performs a similar functionality to the Caller Identification System. The STOP is essentially an extension of the IDENT protocol proposed by M. St. Johns [103]. Each host is expected to run the STOP daemon, which responds to trace requests from other hosts. Unlike the original IDENT, replies sent from the STOP daemon do not identify a user but rather provide a token, which can be used to request information from the system administrator (thus preserving user privacy). Furthermore, STOP can capture "user and application level state" about the originator of a connection request and so provide more information than IDENT. As with the DIDS and CIS above, STOP would be most useful in strictly controlled

network environments, in which assumptions can be made regarding the software installed at each host.

Finally and for completeness, we briefly consider some important work by Sekar, Xie *et al* [96, 97, 98]. The authors address the more general case of network-fraudulent spoofing, where new service data and user data are actively generated from each compromised host (i.e., the attacker's user data is no longer only relayed). Specifically, the authors address the issue of "epidemic spreading attacks" such as an automated DoS or a speading 'worm' (malevolent software that replicates and spreads by some automated mechanism). Their 'random moonwalk' approach [97] requires a directed 'host contact graph', depicting the communication between the hosts involved in the progression of a given attack. The assumed universal view of the network and an attack's progression somewhat limits its practicality. However, the authors also propose a 'Framework for Internet Forensic Analysis' [96] where it is argued that the Internet infrastructure should be extended to include auditing mechanisms that allow 'Attack Identification' and 'Attack Reconstruction'.

3.5 Chapter Summary

This chapter has provided an overview of related work that in some way addresses the issue of spoofing. We have seen that IP traceback and Layer 2 (L2) traceback address sender spoofing whilst connection chain traceback addresses (fraudulent) post-send spoofing. Most IP and L2 traceback proposals adopt the packet accounting paradigm, where information is captured regarding the identity and route of each packet. This allows a-posteriori traceback of packets, that is, after these have been delivered to the recipient host. Connection chain traceback aims to identify which two connections form part of the same connection chain, though typically this can only be done whilst the packet flow is still active.

We consider that a true end-to-end message traceback system will inevitably require elements of all of these traceback types. IP traceback can reveal the instigating origin's first hop router. Then L2 traceback is required to identify the instigating origin's point of access to the network (e.g., the ethernet switch to which this host is connected). Connection chain traceback is required when IP and L2 traceback reveal the effective rather than the instigating origin. Figure 3.14 is intended to aid the reader by summarising the sections and subsections of chapter 3. We will now present our own contributions in the area of Layer 2 traceback.

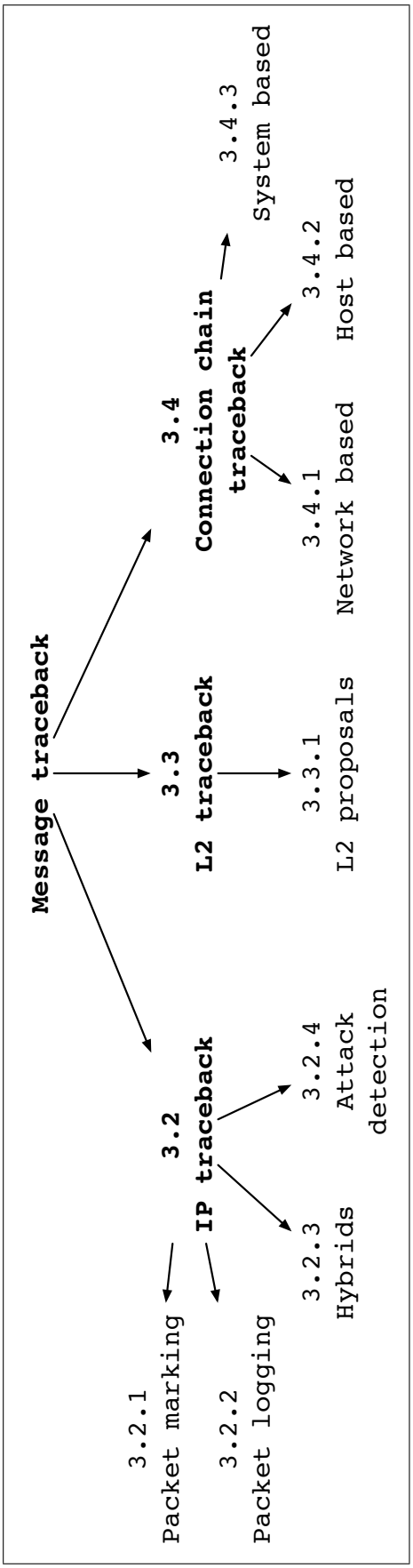


Figure 3.14: Idea map for chapter 3 showing the placement of sections

Chapter 4

switch SPIE

4.1 Introduction

This chapter presents the first of the two Layer 2 traceback systems we developed. As we saw in the previous Chapter, there are a great number of proposals for IP traceback and by comparison much less work in the field of L2 traceback. Part of this chapter was presented as "Logging based IP Traceback in switched ethernets" at EUROSEC 2008 [15].

We decided to pursue L2 traceback for switched ethernet (sw-ether). The only other L2 traceback proposals for sw-ether networks that we are aware of are the adaptation of SPIE by Hazeyama et al [5, 87] and Tagged Frame Traceback (TRACK) by Snow et al [9], that were discussed in the previous chapter. Both of these are designed with switched ethernet in mind and we identified areas to which we might offer improvement. Furthermore we had access to an ethernet switch and 8 dedicated network hosts on which to develop our system.

Given an IP packet that has been traced to its source network, L2 traceback systems aim to identify the origin host. The granularity of the L2 traceback result depends on the physical transmission medium of the deployment network. Switched ethernet allows us to identify the unique access point through which traced data first entered the network. This is achieved by using the switch identifier (sID) and switchport number (pNO) to distinguish each host on the L2 network. Thus given a traced IP packet that originated from the deployment network, switch-SPIE aims to provide the pairing {sID, pNO} that identifies the instigating origin host.

4.2 Requirements

Our system, switch-SPIE is a logging based L2 traceback system for switched ethernet. We now identify and explain the requirements of switch-SPIE before a brief discussion to consider how these are addressed by other known systems:

1. Identifies the instigating origin sID, pNO of any given packet originating within the deployment network (within a bounded traceback window)
2. Requires no changes to existing network infrastructure or protocols.
3. Traceback enabling procedures (i.e. logging) should be considerate of user privacy.
4. The traceback result must be reliable.
5. Support partial deployment and be adaptable to any network topology.
6. Realistic expectations of memory requirements at packet logs.

Requirement one needs no explanation as it forms the primary aim of switch-SPIE. Requirement two states that in order to be useful in a 'real world' deployment, L2 traceback systems must not require any modifications to ethernet switches. That is, any functionality assumed to be provided by the ethernet switches should be widely available. This is not the case in the $TRACK$ proposal by Snow et al [9] where it is assumed that switches 'co-operate' in the packet marking procedure. In $TRACK$ it is assumed that each "TRACK-enabled switch" in the network contains a 'TRACK Frame Tagger' (TFT). The TFT is responsible for determining the sID and pNO though no further details regarding how this is achieved are provided.

Requirement three states that the traceback system should not reveal details regarding the content of network data. Even in a logging based system such as switch-SPIE this is achievable by passing each packet through a cryptographic hash function (e.g., $MD5$) and logging all or part of the output (i.e., the 'packet hash' is logged, rather than the packet itself). This approach was pioneered in the 'Trajectory Sampling approach' by Duffield et al [2] and also used by Snoeren et al in SPIE [3]. By logging packet hashes rather than packets themselves it remains possible to confirm or deny the existence of a given packet at a logging node. However, the (hashed) packet logs themselves are useless to an attacker, even if these are entirely compromised.

Requirement four means that it should not be possible for a malicious network host to 'incriminate' an innocent third party. The procedures by which the sID and pNO are determined for each logged packet should assume that a source `MAC` or source `IP` address are easily spoofed (sender spoofing, as we have seen). Hazeyama et al [5] deploy a single logging node that must maintain local copies of all switch MAC address tables to determine the sID and pNO based on frame source MAC addresses. However, due to this centralised architecture, the single xDGA only 'refreshes' local MAC address tables every 60 seconds. This allows ample time for an attacker to create frames that are attributed to an innocent tertiary host. Snow et al avoid this issue altogether by assuming an 'in switch' process as we have seen. In their more recent proposal, Hazeyama *et al* [87] deploy a single packet digesting tapbox that has a direct link to every switch and router in the network. This work was published at the same time as switch-SPIE [15] and the two systems share some similarities in

the use of port mirroring. However, this newer work still faces the same scalability issues as their earlier work [5] by using a single digesting tapbox.

Requirement five implies that there should be no assumption made over the deployment network topology. We have already seen that in their proposal, Hazeyama et al [5] assume that each network host is separated from the network edge by only a single switch (as is illustrated in Figure 3.10).

Finally requirement six is central to all logging traceback systems as storage requirements govern the 'traceback window'. This is the time during which we can traceback a packet after it has been logged (i.e., before older logs are overwritten). Our switch-SPIE system is an adaptation of SPIE which employs bloom filters to achieve significant memory efficiency. Rather than logging packet hashes themselves, SPIE instead uses the hashes to set bits in a bloom filter, as was illustrated in Figure 3.6. We defer a discussion of the memory utilisation and evaluate switch-SPIE with the rest of these requirements after first presenting our design and implementation.

4.3 Design of switch-SPIE

Our L2 traceback system switch-SPIE is a second adaptation of the SPIE IP traceback system. An open source implementation of SPIE is available from [17] and was first adapted for sw-ether networks by Hazeyama et al who proposed the modified "xDGA" as we have seen.

Logging based IP traceback systems such as SPIE require that IP routers retain information regarding each packet they forward, within some bounded interval known as the 'traceback window'. Thus, they can be queried as to whether they have 'seen' a given packet or not, and if so at which time. L2 traceback systems do the same, however L2 traceback for switched ethernet also provides the unique $\{sID, pNO\}$ pairing that identifies the instigating origin. That is, whilst logging nodes at IP routers need only log each packet, L2 logging procedures at ethernet switches require an additional mechanism by which to identify and associate a source switchport to each packet. In switch-SPIE we use the MAC address table maintained by all ethernet switches to deduce $\{sID, pNO\}$. The same approach is followed by all other known proposals of L2 traceback in sw-ether and we will further examine this process in more detail subsequently.

The SPIE system was briefly considered in the previous chapter, however we provide additional details here that are directly relevant to switch-SPIE. We will then consider the particularities of the sw-ether environment with respect to L2 traceback and which bear upon our design. Finally we will provide details of switch-SPIE and its key algorithms of logging and traceback request processing. Implementation details will follow in a subsequent section.

4.3.1 An Overview of the Source Path Isolation Engine

SPIE is a logging based IP traceback system proposed by Snoeren et al [3]. The system consists of a SPIE Traceback Manager (STM) and a number of Data Generation Agent(s) (DGA). As shown in Figure 4.1 a DGA is deployed at each router; this can be achieved by altering the router software as in Router $r1$ where the DGA is 'built in' or as a router tapbox, as is the case for $r2$ and $r3$. As we will see we choose to deploy our adapted 'switch-DGA' as a switch tapbox through switchport mirroring. This decision was driven by requirement one above, i.e., that our L2 traceback system require no modifications to existing network infrastructure (which includes the switch operating system).

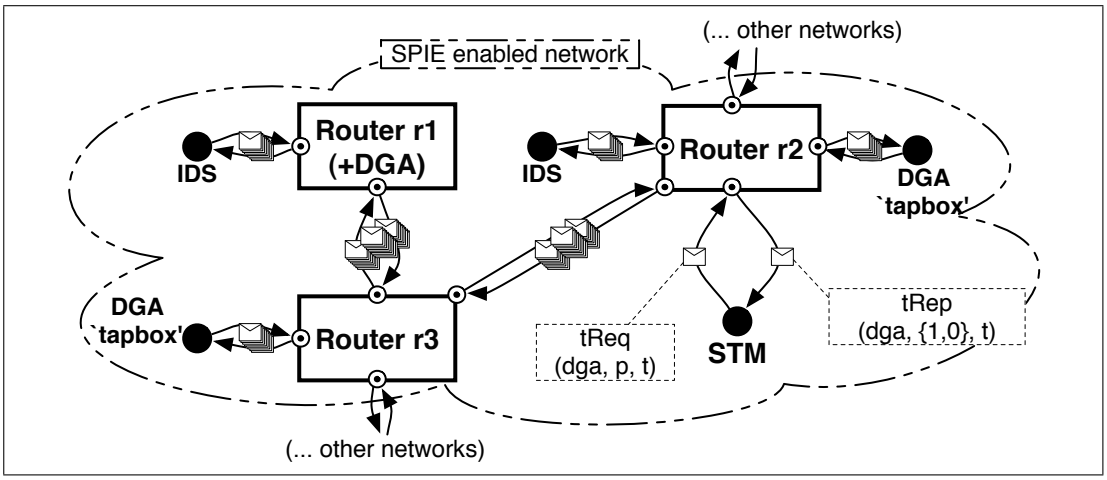


Figure 4.1: A typical SPIE deployment, showing interaction between an Intrusion Detection System (IDS), the SPIE Traceback Manager (STM) and Data Generation Agents (DGA).

The DGA implements the logging functionality and will process every packet forwarded by its router. The n bit results of the k cryptographic hash functions over each packet's content are used as indices into a bloom filter of size 2^n (this is illustrated in Figure 3.6). The bloom filter element specified by each of the k hash results is then set to 1. Thus, in order to check whether a packet is present in the bloom filter log (known as a 'packet membership test'), one must calculate the k packet 'hashes' and verify that all specified bloom filter elements are set to 1.

The open source implementation of SPIE [17] which we adapt uses 20 bytes from the IP packet header and 8 bytes from the packet payload as input to the cryptographic hash function at each DGA. The MD5 algorithm is used, and the 128 bit output of a single hash is used as four 32 bit digests (i.e. k is 4 and n is 32). This supports bloom filters (i.e., bit arrays) of size up to 2^{32} bits, which as we will see in our discussion of memory requirements is sufficient for a switch-DGA logging node. Each bloom filter is only useful as long as it does not become 'saturated'. Once too many elements are set to 1 then the result of packet membership tests becomes unreliable, that is, the false positive rate of the bloom filter increases above an acceptable level of error. Thus a DGA maintains

each bloom filter for a predetermined time interval after which this is replaced with a new bloom filter. The default swap time in the SPIE implementation is 10 seconds, and an 'archive' of 6 bloom filters is maintained at a given time, providing a 'traceback window' of 60 seconds.

As can be seen in Figure 4.1 the SPIE system can be used in conjunction with an Intrusion Detection System (IDS). Each IDS alert generated is translated into a traceback request, $treq(dga, p, t)$ specifying the last hop router DGA (dga), the packet itself (p) and the time (t) that the packet was observed. This is then communicated to the STM which dispatches the tReq to the specified dga for processing. The DGA performs a packet membership test in its archived bloom filters for packet p and then provides a traceback reply $tRep(dga, \{1, 0\}, t)$. The tRep specifies whether the given DGA has seen the packet or not (i.e., one of 1 or 0) and if it has the reply will also specify the time t that the packet was logged. For each DGA that provides a positive traceback reply, the STM will dispatch traceback requests to the given DGA's neighbour routers. After collecting replies from all relevant DGAs, the STM can construct a path of the packet's route and the packet is 'traced back' through the SPIE enabled network.

4.3.2 Characteristics of switched ethernet that impact on L2 traceback

The switched ethernet environment is particularly interesting with respect to L2 traceback. A switched environment means that each host gets a dedicated link to the switch, and so medium access contention is eliminated. This bears upon the design of our L2 traceback system in two ways, one of which is positive and one negative.

The negative consequence of each host's dedicated link to the switch is that there is no single vantage point from which to observe all traffic generated by the switch's connected hosts [104]. That is, each (unicast) ethernet frame is 'switched' from ingress to egress switchport and cannot be observed by a host on any other switchport. In switch-SPIE we use switchport mirroring to provide our modified DGA with all frames transmitted by a switch's hosts. Most switches allow traffic from one or more ports to be duplicated and sent to another 'monitor' port. Though this process is not standardised it is a widely available feature and identified in [18, 19] as one of the monitoring mechanisms for switched ethernet. In switch-SPIE, the egress traffic from all ports to which hosts are attached is mirrored to a monitor port, to which our modification of the SPIE DGA (termed 'switch-DGA') is connected.

Hazeyama et al [5] resolve the traffic visibility issue by logging at the gateway router rather than at each switch. However, this means that frames which are entirely local are not logged. For instance, frames transmitted between two hosts on the same switch, or on separate switches that are not connected via the gateway router. The result of this is a system that can trace the instigating origin of malicious traffic, but not if the victim of said traffic is within one's own network. Moreover and perhaps more significantly a centralized logging approach ultimately sacrifices the trace result's

reliability, as one is forced to update local MAC-address tables less frequently. This conflicts with our second requirement, that is, that the traceback result be reliable. Our second L2 traceback system, COTraSE (presented in the next Chapter), addresses the traffic visibility issue through the alternative mechanism of a passive tap between switches, rather than using switchport mirroring.

The positive consequence of providing each host with an exclusive link to the switch is that ethernet switches need to create and maintain a MAC address table (MAC-table). This table associates the MAC address of each host with the switchport to which that host is connected. Thus to correctly forward frames, switches consult the MAC-table to determine the appropriate egress port. The MAC-table is constructed through 'learning' addresses from ingress (to the switch) frames; a MAC address is reachable over the port from which traffic carrying that address as source was most recently seen. Each address can only be listed once; thus, if a host moves to another port (or the source MAC is forged), the MAC-table is updated to reflect the new mapping. That is, ethernet switches submit all correctly received user data frames to the switch 'Learning process' [105]. Our modified switch-DGA maintains a local copy of the switch MAC-table. This is used to determine the origin switchport of received frames and local MAC-tables are updated from the switch MAC-table at some predetermined rate. Though this approach is also taken by Hazeyama et al [5], the distributed design of switch-SPIE means that local copies of MAC-tables can be updated (much) more frequently. We will further discuss this point in the evaluation of switch-SPIE later in this chapter.

4.3.3 The switch-SPIE logging process

As stated above we deploy logging of IP packets at each switch with a tap-box. Each tap-box will host our switch-DGA adaptation of the open source SPIE DGA. Rather than one bloom filter per time interval, we employ an array of n bloom filters, one for each switchport. We will evaluate the practicality of this in our discussion of memory requirements later.

The switch-DGA tapbox is installed on a switchport to which the switch is instructed to mirror all network traffic. This is illustrated in Figure 4.2, where the switch-DGA is at host $h4$. For our switch-SPIE adaptation we decided to log frames that originate from access ports, but not from link ports. As shown in the diagram below, link ports are used to connect to other switches or routers and access ports are used by end hosts. Thus, in Figure 4.2, the switch is instructed to mirror traffic from ports $1, 2, 3, 5, 6, 7, 8$ but not from ports 9 and 10. This decision was driven by the fact that ethernet switches will typically provide a small number of higher bandwidth ports (e.g. 1Gbit/s as opposed to 100Mbit/s switchport) and these are used as the link ports. For instance, the CISCO Catalyst $3524XL$ switch that we used in our implementation offers two Gigabit ethernet ports and 24 Fast ethernet ports. The monitor port, port 4 in Figure 4.2, will be receiving the aggregated traffic from all other access ports. Given that link ports can transmit and receive an

order of magnitude more traffic than access ports, we expected that to mirror that traffic as well would simply overwhelm the monitor port.

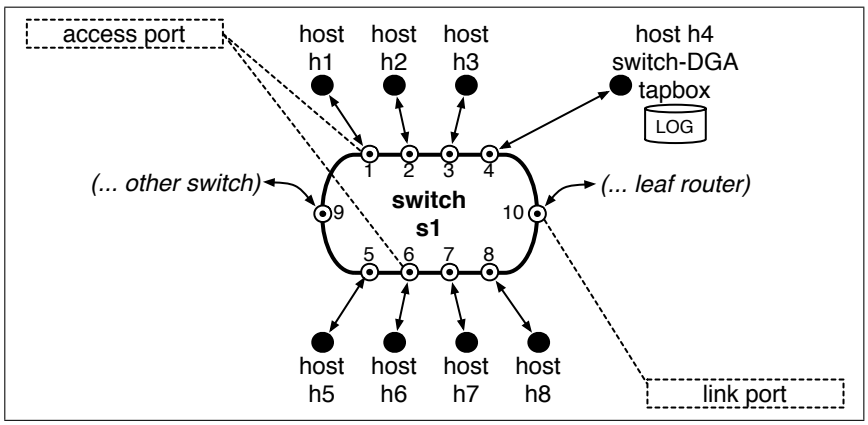


Figure 4.2: Placement of a switch-DGA tapbox at each switch, logging traffic from all access ports.

The implication of this design decision is that switch-SPIE cannot provide a graph of a traced packet's route through the network, as is the case in the original SPIE. Each switch-DGA tapbox will log traffic from its switch's access ports. Thus, each packet is logged only once in the switch-SPIE enabled local network and only a single switch-DGA tapbox will provide a positive traceback reply to traceback requests (more on this subsequently). The primary goal of switch-SPIE is to identify the instigating origin (sID, pNO) and this is achievable by monitoring only access ports. This decision is in line with requirement 1, that is, no modifications are required to existing network infrastructure or protocols. Whilst we considered attaching switch-DGA to a Gigabit ethernet port, it is more realistic to expect that these ports are already used as link ports, and not available for the purposes of logging. We will see how switchport mirroring is configured for the CISCO IOS software in the implementation section that follows.

Our design requires the switch-DGA to maintain an array of n bloom filters, where n is the number of access switchports. Logging of packets occurs as in SPIE, with the additional step of identifying which of the bloom filter array elements is to be used to log each packet. This is achieved by consulting the switch MAC-table based on the frame source MAC address. Given the local proximity of each switch-DGA to the switch itself, it is possible to use the dedicated 'console port' provided by ethernet switches for local switch management (and which gives access to the MAC-table). This is ideal as it eliminates the need to communicate MAC-table values from the switch to the switch-DGA over the already congested monitor port. An alternative approach is to use SNMP to retrieve the switch MAC-table at a predetermined interval. Time constraints meant that we were unable to establish the feasibility of the SNMP approach and as we will see our implementation uses a text file to simulate the switch MAC-table.

Thus, with switchport mirroring enabled and access to the switch MAC-table, the switch-DGA

```
Logging
-------
receive ethernet frame f
extract source MAC address f.src_MAC
extract SPIE packet signature f.sig from f


consult MAC-table to identify port pNO:
for(every entry addr.i in MAC-table)
{
     if(addr.i == f.src_MAC)
     {
          extract pNO;
          break loop;
     }
}


store packet into bloom filter array
bfArray[]:
calculate k hash digests using f.sig
for(every digest k)
{
     set element in bfArray[pNO];
}
```

```
Traceback request processing
----------------------------
receive traceback request tReq(p, t)
extract SPIE packet signature p.sig from p
retrieve archived bfArray[] for time t
calculate k hash digests using p.sig

check bloom filter array for packet p:
set found to false;
for(every bf.i in bfArray[])
{
    if(every k is 1 in bfArray[bf.i])
    {
        source port pNO was bf.i;
        set found to true;
        break loop;
    }
}
if(found)
{
     send tRep(sID, 1, pNO, t) to STM;
}
else
{
     send tRep(sID, 0) to STM;
}
```

Figure 4.3: switch-DGA algorithms for logging and traceback request processing

logging process is as shown in Figure 4.3. For each frame received at the switch-DGA monitor port, the origin pNO is established from the latest MAC-table based on the frame's source MAC address. The k hash digests are calculated based on the packet signature as per SPIE, and these are inserted into the bloom filter array at the element representing the given pNO. Given a traceback request, $tReq\{p,\ t\}$, with packet p and timestamp t, the switch-DGA checks archived bloom filter arrays based on t. Each bloom filter is checked by computing the hashed digest of p as explained earlier. When a match is found, the array index of the current bloom filter reveals the source port. This can then be communicated to the STM as a traceback reply, as is illustrated in Figure 4.3. We now turn to examine how this functionality was implemented by adapting the open source version of SPIE, available from [17].

4.4 switch-SPIE implementation details

We begin by describing our deployment network and the installation of the relevant SPIE components (STM and DGA). This is followed by details of changes we made to the DGA software to produce our switch-DGA adaptation. Changes were also required at the STM to allow receipt of the sID and pNO data in positive traceback replies from a switch-DGA (the original SPIE has no notion of pNO and sID). The SPIE components used in our implementation are from SPIE version 3.0.81 [17]. Finally we will consider how the switch MAC-table was simulated and used by switch-DGA and also how switchport mirroring is configured on the deployment ethernet switch.

4.4.1 Deployment network and installation of SPIE components

Our deployment topology is composed of eight hosts connected by a Cisco switch. All hosts have at least 2.4GHz processor, 512MB of RAM and a 100Mb/s ethernet interface. The switch, 'cs-daysh4' is a Cisco Catalyst 3500XL series running Cisco IOS v.12.0(5), with a total of 24 Fast ethernet ports (100Mbits/s) and 2 Gigabit ethernet ports (1000Mbits/s).

One of the hosts, 'redmire', served as switch-DGA and another, 'falstone', hosted the STM and IDS (Snort [106]). Both redmire and falstone had linux kernel version 2.4.2-2 as this was a requirement for installing SPIE. The remaining six machines ('erble1' - 'erble6') were used as network hosts for generating test traffic. Figure 4.4 shows the switchports to which these hosts were attached on the cs-daysh4 switch.

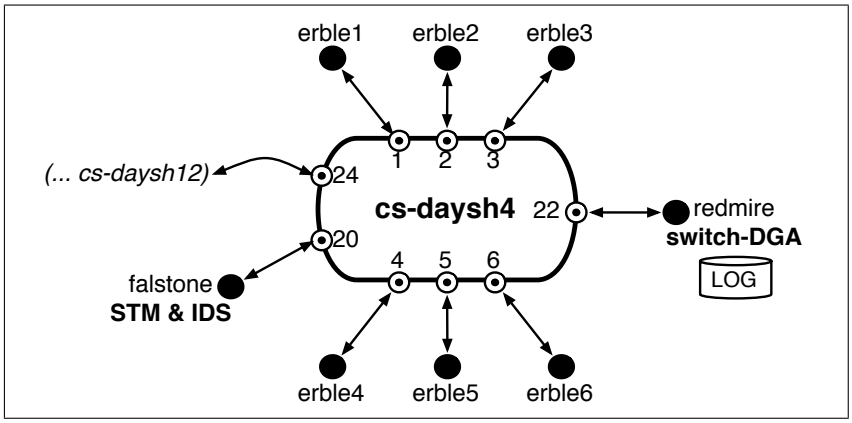


Figure 4.4: Deployment network showing placement of switch-SPIE components

Installation of the SPIE Traceback Manager (STM) on falstone was relatively straightforward as this is a user space application. However, installation of the Data Generation Agent (DGA) on redmire required a kernel rebuild, as the SPIE logging functionality is implemented as a kernel module (more on this subsequently). For the interested reader, further details regarding installation procedures can be found in the SPIE release documentation [17].

In addition to the STM, falstone also hosted the IDS and the spiemon application (included with the SPIE release). The spiemon process is responsible for receiving IDS alerts and sending these to the STM as traceback requests. The spiemon also receives the traceback replies from the STM and displays results to the user. We used the open source Snort IDS version 1.9.0 [106] as this was recommended for the supplied version of spiemon.

4.4.2 Implementing switch-DGA

The SPIE DGA decomposes into two core components directly relevant to our switch-DGA implementation. One of these is a loadable kernel module written in the `C` language called 'spie_mem' and

the other is 'dgad', a user space process written in C++. The kernel level spie_mem is responsible for processing each incoming packet and inserting this into the current bloom filter, whilst the user space dgad code retrieves the filled bloom filters from the kernel and archives them for subsequent traceback request processing.

Understanding the very complex SPIE system in order to implement our required functionality of 'one bloom filter per switchport' proved to be quite a challenge, compounded by the fact that this was our first exposure to C and C++. The spie_mem component is written as a device driver, interfacing with a virtual SPIE 'device'. As spie_mem is executed from within the linux kernel, the 'Kernel Source Level Debugger kgdb' [107] proved invaluable in helping us to effect the switch-DGA functionality; falstone was connected to redmire (the switch-DGA host) through a "null modem" (RS-232 serial cable), which allowed us to 'pause' the kernel on redmire and set breakpoints to step through lines of kernel code.

The user space dgad maintains a fifo ring buffer of size max_records where each ring buffer element is a trace_record structure defining a bloom filter and timing information. The default value for max_records is 6, which equates to a 60 second traceback window as bloom filters are swapped every 10 seconds (these are user configurable parameters - discussion is available in [3]). The dgad process maps each new bloom filter into the kernel's memory space for spie_mem using the kiobuf mechanism (kernel input/output buffer), avoiding costly copy operations [108]. The spie_mem driver maintains two bloom filters at any time, *current* and *next* in order to avoid delays in swapping these (every 10 seconds).

In switch-DGA the dgad instead uses a ring buffer of size $max\_records * numberofports$, as we need a separate set of trace_records for each switch port. This is illustrated in Figure 4.5 which shows the ring buffer when there are 4 switchports. Each switchport requires $max\_records = 6$ elements to provide the one minute traceback window, giving a total of 24 elements. Furthermore, an extra dimension is added into the kiobuf array maintained by spie_mem to represent the current and next records for each switchport. We use the SPIE DGA configuration file to input the number of ports; the configuration file is parsed when the DGA is started and a call to sysctl creates a spie.dga.max_ports entry into the /proc (virtual) file system; /proc is used as a way of communicating values between the user and kernel level code.

The DGA configuration file is also used to supply the current values of the switchport MAC addresses (i.e. the local copy of the switch MAC-table). These are parsed and stored into a character array by the user space dgad. Once the kernel spie_mem driver receives the number of switchports it creates the appropriate number of entries into the /proc filsystem to hold the MAC address of each port. The dgad waits for spie_mem to create the /proc entries and then fills these using sysctl and the MAC addresses that were previously parsed from the configuration file.

Thus both the user space dgad and kernel space spie_mem have the number of switchports and

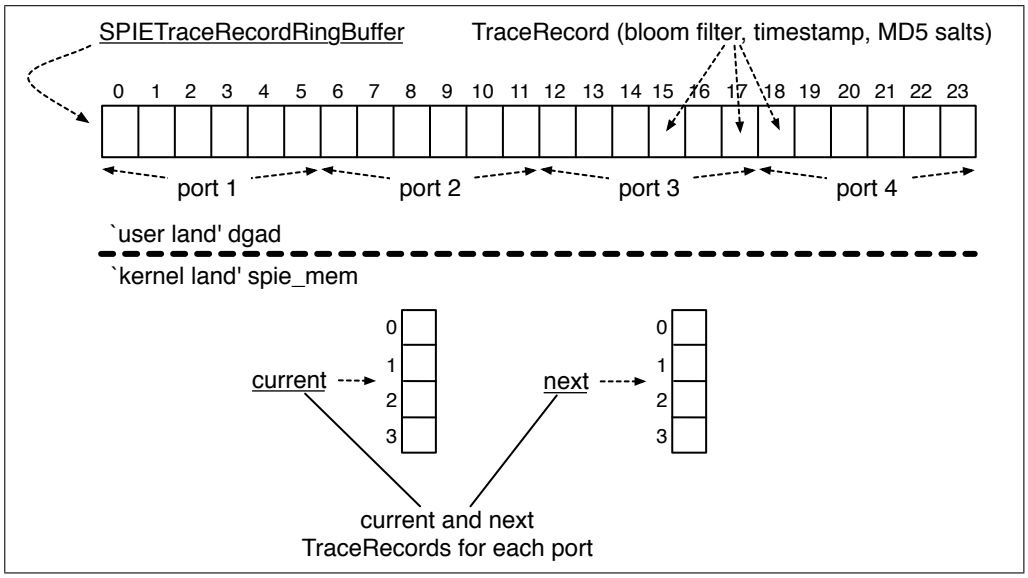


Figure 4.5: User and kernel space structures that hold bloom filter information, when there are four switchports

the MAC address associated with each of these, as well as an array of bloom filters per time interval (one for each of the switchport MAC addresses). For each frame received the `spie_mem` code first checks the (local copy of the) switch MAC-table to determine the appropriate bloom filter. The code then proceeds to calculate the MD5 digest and store this in the same manner as the original SPIE. That is, no changes were required at the MD5 digesting functions beyond specification of the bloom filter array index for storing the hash output.

When searching archived bloom filters for a given packet (i.e. during traceback request processing), a loop is used to check the bloom filter for each switchport. When a match is found (i.e. all bloom filter elements are set to 1), the index of the current bloom filter also reveals the instigating origin switchport. It was necessary to make alterations to the `STM` and `spiemon` components since as mentioned earlier, the original SPIE has no notion of pNO and sID. The sID is implied by the switch-DGA sending a given traceback reply (as there is one switch-DGA per switch) and a new type of SPIE traceback reply was created to accommodate the value of pNO.

As mentioned earlier, we used the port mirroring function that switches provide to overcome the traffic visibility issue inherent to switched ethernet. Our switch, `cs-daysh4` runs Cisco IOS v.12.0(5) and we managed this remotely via `telnet`. In Cisco parlance, port mirroring is known as 'SPAN' (switchport analyser) and our 3500XL series switch offers a port monitor function to mirror m source ports to n monitor ports (though in our deployment and as illustrated in Figure 4.4 we mirrored m to 1). The monitor port receives both egress and ingress traffic for each switch port assigned to it, and this is not configurable (even though switch-DGA only requires egress traffic). Figure 4.6 shows the commands used to mirror traffic to switchport 22 that hosts the switch-DGA

(shown in Figure 4.4).

Enter configuration mode:
```
configure terminal
```

Specify the switchport to be configured:
```
interface FastEthernet0/22
```

Specify switchports to be monitored:
```
port monitor FastEthernet0/1
port monitor FastEthernet0/2
port monitor FastEthernet0/3
(...)
end
```

Check the current configuration to verify the changes:
```
show running-config
```

Figure 4.6: Commands for enabling port mirroring under Cisco IOS v.12.0(5)

4.5 Evaluation of switch-SPIE

As explained earlier, the host 'redmire', served as switch-DGA whilst host 'falstone', hosted the STM and IDS (Snort [106]). The remaining six machines ('erble1' - 'erble6') were used to generate test traffic. The switch was instructed to mirror traffic from ports 1 through 6 (the erbles) to port 20 (IDS and STM on falstone) and also to port 22 (switch-DGA on redmire). The switch is not usually expected to have two monitor ports active as the STM is typically at a central network location. The STM of course does not play any part in the logging processes of switch-DGA.

The implemented switch-DGA was tested using the six erble machines. The snort IDS running on falstone was configured to generate an alert whenever an ICMP echo request was seen between any two erble machines (e.g., erble1 *ping* erble2). This is of course an arbitrary trigger but was sufficient for our purposes of establishing whether switch-DGA could provide the instigating origin pNO for a given IP packet. The STM on falstone created and sent trace requests for every alert generated by snort, and the switch-DGA confirmed the originating machines in every instance.

Of course, it is important that the system does not incur too much overhead when deployed at each switch. In a logging based system such as switch-SPIE memory requirements are a primary concern; one of our L2 traceback system requirements identified earlier directly addresses memory utilisation. Thus we evaluate memory requirements at individual switch-SPIE logs and examine the interplay between memory use, the number of hash digests per packet, false positive rate and bloom filter size (explained subsequently). We will see that memory requirements can be kept within reasonable bounds in current networks. However, it would seem that further investigation of efficient logging mechanisms is warranted, especially when considering larger and faster networks (e.g. 10 Gigabit Ethernet).

We start however by presenting our experiments, designed to establish the reliability of the

port mirroring mechanism. Given the reliance of switch-SPIE on this functionality we felt it was important to establish the 'real world' limits of this approach. Our first set of tests shows that the switch is able to mirror traffic without adverse effect to the 'normal' switching functions. However, as we will see our second set of tests show that the monitor port and logging host can be quickly overwhelmed.

4.5.1 Reliability of the port mirroring mechanism

The equipment used here is as described in subsection 4.4.1. We repeat the specifications here with some relevant additional details: The six erble hosts run Cent OS 5.1 (linux kernel 2.6.18), whilst redmire and erble run an older redhat distribution (kernel 2.4.2). The erble machines have dual Intel Xeon processors (2.4GHz for erble1 and erble2, 2.8GHz for erble3, 4, 5 and 6) with 512Mbytes of RAM. Redmire and falstone have a Pentium 4 processor (redmire 2.4GHz, falstone 3GHz) with 512Mbytes RAM and a 100Mbit/s ethernet interface. The erble machines have a 1Gbit/s interface though they are connected to the switch via a 100Mbit/s switchport. Finally, the switch itself is a Cisco WS-3524XL running Cisco IOS v.12.0(5) with 24 'fast ethernet' ports (100Mbit/s) and 2 'gigabit ethernet' ports (1Gbit/s).

Our experiments are intended to establish whether switch port mirroring is an efficient and reliable means of providing traffic to the switch-DGA. The first set of measurements is concerned with how well the switch keeps up with simultaneous frame switching and mirroring. We simulated short bursts of traffic between the erble machines using 'ping' in flood mode. During a five second burst each erble sent and received an average of $23,000$ frames. That is just under $5,000$ frames per second, equating to around 4Mbit/s bandwidth per switchport as each frame occupies 98 bytes 'on the wire'. With six source ports active this approaches a quarter of the total 100Mbit/s link into a switch-DGA (assuming a FastEthernet port is used).

We do not claim that our experiments here 'stress' the switch to capacity. Whilst limited by the configuration of the testbed (in terms of available hosts) the first investigation described below shows that the given (mid-sized) CISCO switch can duplicate the traffic sent from 6 source switchports to 3 different mirror ports without affecting the time taken for the source traffic to be delivered. Comparing the average round trip time (RTT) reported by ping when mirroring only 3 source ports, with that reported for mirroring 6 source ports shows that the RTT is not increased at all. Furthermore, comparing the number of packets sent/received by ping with the number of packets captured by snort on a given monitor host shows that the switch duplicates and forwards all packets correctly in both the 3 and 6 source port cases.

First, three 'flooding' erble machines, erble2, erble4 and erble6 instigated their five second ping flood simultaneously. The switch was instructed to mirror traffic from each of these to a seperate monitor host, as depicted in Figure 4.7. The snort IDS was running in 'binary logging' mode on

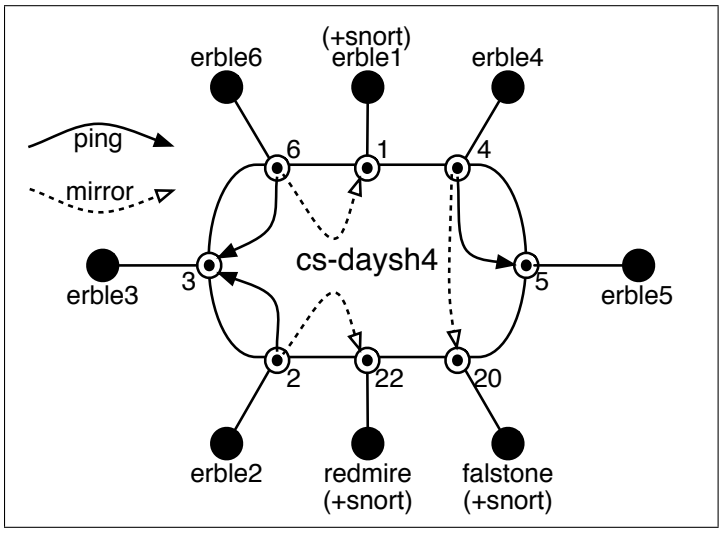


Figure 4.7: Conceptual view of configuration when testing the switch's ability to mirror traffic

the three monitor machines, redmire, falstone and erble1. A comparison was then made between the packets sent and received as reported by 'ping' at each flooding machine, with the packets seen in the monitor machine snort logs.

In all cases the switch was able to keep up the mirroring with a 0% drop rate. That is, the number of packets sent and received by each flooding host was the same as the number of packets received at each monitor host and the switch did not 'drop' any of these due to resource exhaustion. This is shown in Figure 4.8 which gives the number of packets sent/received by ping and those reported by snort on each monitor. Performance was maintained after we repeated the measurement and configured the switch to mirror traffic from both source and destination erble machines to each monitor host, shown as "six simultaneous mirrors" in Figure 4.8. As can be seen in Figure 4.8 when mirroring from 3 sources (left hand column) host erble2 sent and received 60,004 packets with an average RTT of 0.145; the switch was mirroring traffic to redmire which logged 60,004 ICMP ('ping') packets. The corresponding values for the 6 sources run are shown in the right hand column. Thus, as shown in Figure 4.8, erble2 sent and received 60,013 packets. However, redmire received and processed 100,023 packets; this difference is due to host erble3 smultaneously receiving a ping flood from erble6 and so redmire logged those further 40,009 frames. In this configuration of six source monitor ports, the average frame round trip time reported by ping was essentially the same as with only three source ports.

The above indicates that the mirroring process does not adversely affect the 'primary' switching function. In these circumstances (which represent perhaps higher than normal load at about 25 percent of link capacity) switch capacity is sufficient so that mirroring is not noticeable. We would expect round trip times to have increased for instance, had frame switching been adversely affected. Obviously, we could push the switch load further by increasing the number of sources, but our

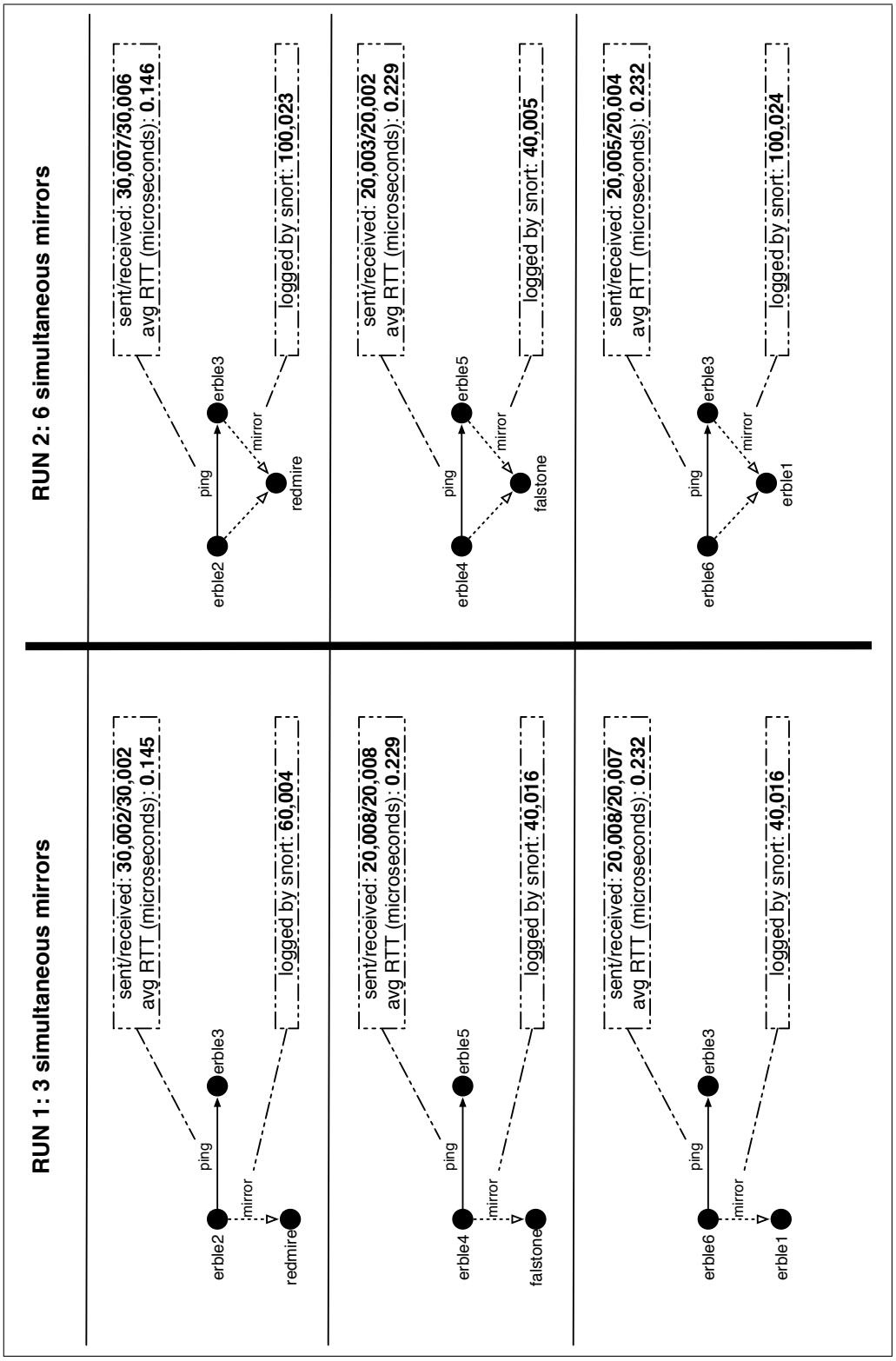


Figure 4.8: Investigating the switch mirroring ability: number of packets sent/received by ping and logged by snort, and RTT reported by ping

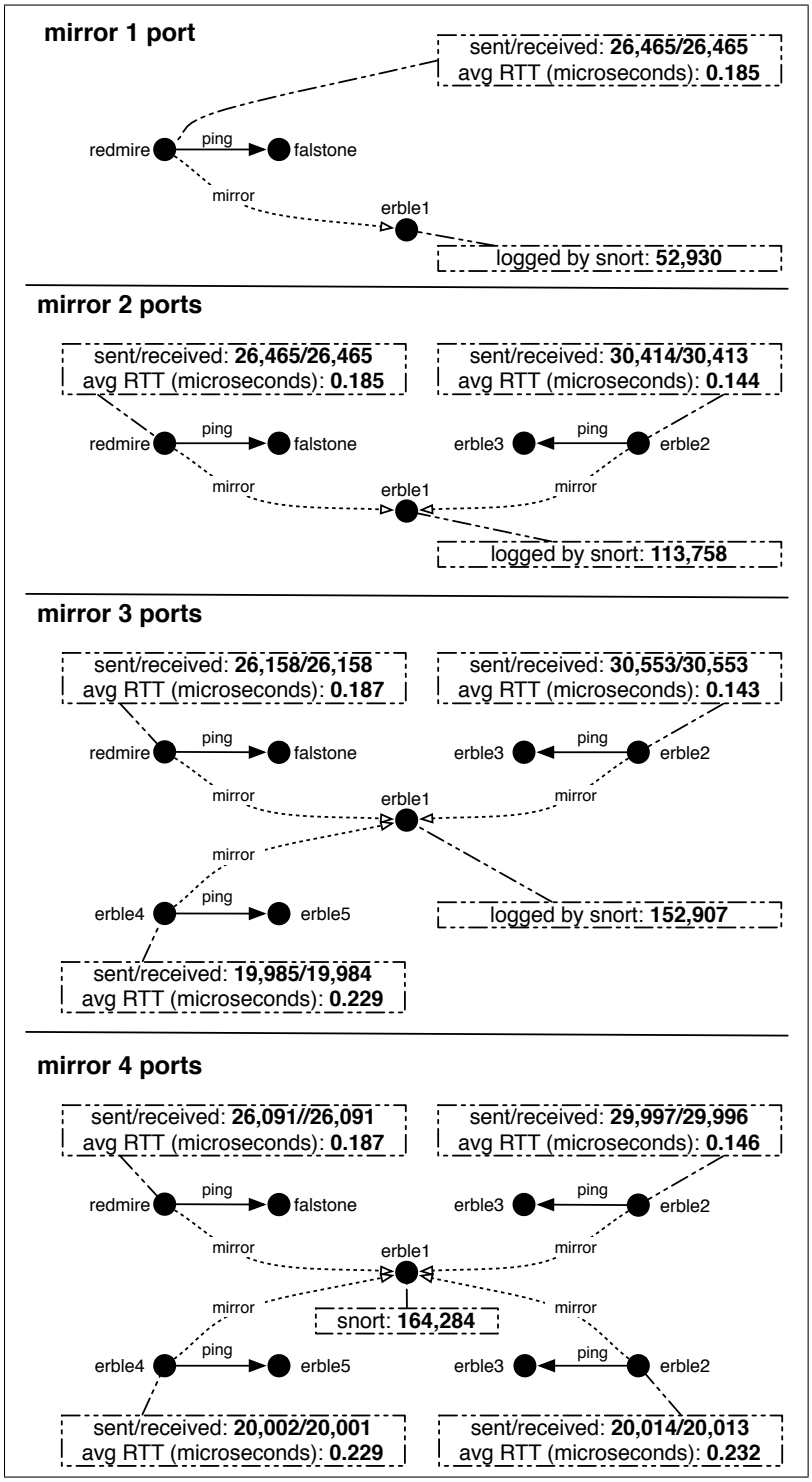


Figure 4.9: Number of packets sent/received by ping and logged by snort whilst increasing the monitored hosts

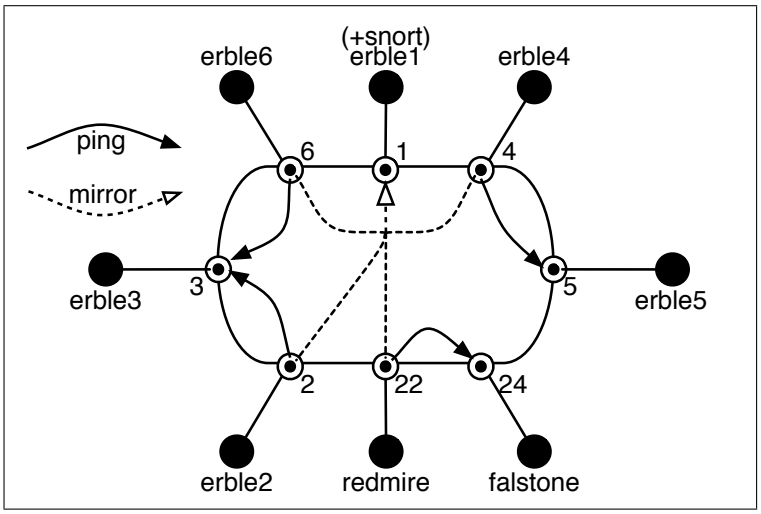


Figure 4.10: Conceptual view of mirroring configuration to establish ability to capture a number of sources at the monitor host

experiments were limited by the configuration of the available hardware. It should be noted that although the implementation of port mirroring will vary, we expect that on a highly loaded switch the mirroring processes become a secondary concern; in trying to keep up with frame switching the switch will (eventually) drop frames from ingress and egress mirroring queues [104]. Hence, the switch will in any case be protected against compute-intensive mirroring.

Our experiments highlighted that the bottleneck of our solution is typically not the processing overhead at the switch, but at the monitoring machine, which must keep up with the traffic sent to it by the mirroring processes. Thus we explored the limits of our equipment with respect to the number of active sources that can be mirrored to the tap-box before it is forced to drop frames. We designated erble1 as monitor host, running snort in binary packet logging mode. A number of traces were taken, with traffic load again being provided by 5 second simultaneous ping floods. Starting with only one source host (falstone *ping* redmire), the number of sources being mirrored to erble1 was then increased for each trace until we began to observe a packet loss. That is, a comparison was made in each case of the packets sent and received by the flooding hosts (reported by ping) and the packets received by snort on erble1. Figure 4.9 gives details of the number packets sent and received by ping on the source machines with the reported average RTT values and the number of packets logged by snort in each case. Table 4.1 summarises the percentage packet loss in each case and Figure 4.10 gives a conceptual view of the switch setup when mirroring 4 switchports to the monitor switchport 1 (host erble1).

We observe a packet loss at the monitoring node when we have more than three sources. Of course our work does not consider 'specialised' hardware, and our prototype is constructed with available equipment. Mechanisms that could limit the number of dropped frames include 'rate limiting' each switch port, or equipping the monitor host with more than one network interface.

Table 4.1: Summary - packet loss at the monitor host

Sources	*Frames* (total)	*Captured by snort*	*%Loss*	*RTT* (ms)	*Frames* (avg/host)
1	52,930	52,930	0	0.185	26,465
2	113,758	113,758	0	0.1645	28,439
3	153,391	152,907	0.32	0.186	25,565
4	192,205	164,284	14.5	0.198	24,026

Another obvious approach is to use one of the Gigabit-Ethernet ports to connect the switch-DGA, and a more powerful machine to process the data. Although such a solution should resolve the issue, it is also expensive. Thus, the need for more efficient logging mechanisms served as a central motivation for our second L2 traceback system COTraSE presented in the next chapter.

4.5.2 switch-SPIE memory requirements

An interesting property of bloom filters is that they cannot produce false negatives. That is, membership tests for an element e whose hashed digest is represented in a bloom filter will never be negative. Thus the following is concerned wholly with false positives, which are used as a measure of performance for bloom filters. A false positive in this context is that an innocent host is implicated as the source of malicious traffic. That is, given a packet p whose packet digest was not stored in a bloom filter, a membership test for p is positive.

The bloom filter false positive rate P is expressed as a function of its size (m bits), number of elements inserted (n packets) and number of hash functions used (k):

$$P = \left(1 - \left(1 - \tfrac{1}{m}\right)^{kn}\right)^k$$

The false positive rate can be lowered by increasing m and limiting the number of insertions n. Both however increase memory requirements (the latter results in more bloom filters overall). The memory efficiency factor $\frac{n}{m}$ is used in conjunction with P to indicate performance. By increasing k (with an equivalent increase in m) the false positive rate is reduced with only slight reductions in memory efficiency [3]. That is, increasing the number of bloom filter elements 'set to' 1 to represent each packet can produce a lower false positive rate, if bloom filter size is adjusted accordingly. However, we must limit the number of digests due to the obvious processing overheads in computing these.

For the purposes of our discussion we first establish some bounds. The number of insertions, n, is bounded by the link speed. We assume that bloom filters are used for 10 seconds before being swapped (this is the default 'paging rate' in [17]). For performance purposes, we also assume that no more than $k = 4$ digests are used for each packet 'inserted'; in SPIE [17], the 128 bit output from a single MD5 hash is used as four 32-bit digests. This supports bloom filters up to 2^{32} bits, which as we will see is sufficient for our purposes.

Querying each switch-DGA for a given packet is an independent Bernoulli trial (success - true positive/true negative or fail - false positive). Thus, the number of false positives overall follows a Binomial distribution. In a network with x 'SPIE-DGA' enabled switches, each maintaining false positive rate P, the expected number of false positives overall is xP. The Newcastle University School of Computing Science network consists of just under 30 switches; for the purposes of this discussion we arbitrarily select $\frac{1}{3000}$ as a reasonable upper bound on the false positive rate. That is, if each switch maintains $P \leq \frac{1}{3000}$, we expect false positives in the set of trace replies roughly 1% of the time. It should be noted that this is a much stricter bound on the false positive rate than the 0.125 used in the analysis of [3].

Assuming full utilisation of a $100Mbit/s$ link into a switch-DGA, we expect $1,000,000,000$ bits in 10 seconds. Taking an average frame length of 1000 bits, $n \approx 1,000,000$ frames are stored in each bloom filter before it is replaced. Using $m = 2^{25}$ bits (4Mbytes) a false positive rate of 0.00019 is achievable with $k = 4$ hash functions [109]. In this configuration, the optimal value for k is 22, giving vastly improved performance with $P = 0.00000021$.

The bloom filter size above ($m = 2^{25}$ bits) equates to 24Mbytes of storage per switchport per minute. At the very high end, a CISCO Catalyst 6513 Ethernet switch provides $1,152$ FastEthernet ports, and so requires 27Gbytes for a one minute 'traceback window'. That is, we can 'traceback' packets generated up to one minute ago; we consider this sufficient for an automated deployment in which an IDS interfaces with the STM to send requests to switch-DGA. At the lower end, a 24 port switch would require 576Mbytes.

This is of course a worst case scenario, that is, we assume full link utilisation with all traffic originating from a single access port (and so stored in a single bloom filter). In practice however, memory requirements can be relaxed. If we assume that frames originate from each switch port with equal probability, we can reduce the expected number of insertions in 10 seconds. Distributing the estimated $1,000,000$ frames over 1152 ports gives an average of 868 insertions per bloom filter, requiring only 2^{15} bits (i.e. 24KBytes/min/port) to maintain $P \leq 0.00033$. A smaller switch needs to provision more memory per bloom filter in this scenario as each accounts for a greater proportion of frames. In the 24 port case we can assume $42,000$ insertions per bloom filter, requiring 2^{21} bits to maintain the same expectation on P. That is 1.5Mbytes per minute per switch port.

In a larger network each switch-DGA must maintain a lower false positive rate to achieve the same performance (e.g. false positives in roughly 10% queries). The link speed is a significant factor in determining memory requirements for a given false positive rate. In Table 4.2 we show requirements for 100Mbit/s and 1Gbit/s Ethernet with corresponding values of P. When the optimal value of k for a given m is greater than 4, we show P for both the optimal case and $k = 4$.

From Table 4.2 we can see that in the GigabitEthernet case we need roughly 192 Mbytes per minute (per switchport). With $k = 4$ this produces a false positive rate slightly higher than our

Table 4.2: Memory requirements (m) and corresponding false positive rates (P)

n (frames)	m (bits)	P (k=4)	P' (optimal k)	$Mbytes/min$ (per port)
100Mbit/s:	2^{23}	0.024	0.022	6
$\approx 1,000,000$	2^{24}	0.0024	0.00046	12
	2^{25}	0.00019	0.00000021	24
1Gbit/s:	2^{27}	0.0052	0.0021	128
$\approx 10,000,000$	2^{28}	0.00044	0.0000046	192

expectation of $P \leq 0.00033$. In the worst case, a CISCO Catalyst 6513 switch can provide 577-GigabitEthernet ports equating to rougly 100 Gbytes for a 1 minute traceback 'window'. Of course, one can produce a lower false positive rate by using the optimal number of hashed digests k for each packet (rather than increasing memory allocation), though the feasibility of this will depend on the performance of available hardware.

4.5.3 Discussion of Switch-SPIE

In [5] the deployment of xDGA within gateway routers alleviates the traffic visibility problem, as the router sees all traffic leaving the network. Traffic sent between hosts on the same switch is not observed and so xDGA cannot traceback malicious activity (wholly) within the local network. A key element of Switch-SPIE compared to [5] is the fact that logging takes place at the switches. As a consequence, switch-SPIE identifies the host even when more than one switch separates a host from the gateway router, as will usually be the case. As a side-effect, the switch-SPIE configuration allows traceback even for frames that are wholly local.

An additional benefit is increased scalability. The processing load is distributed and so traceback request processing is an order of magnitude lower at each switch-DGA compared to the (single) xDGA. Furthermore, in [5] the xDGA must maintain 'up to date' MAC-tables for all switches in the network.

There is an inherent race condition in using the MAC-tables for resolving origin port from frame source addresses; if a forged frame is sent, the MAC-table mapping at the origin switch will correctly identify the source port, as correctly received frames are always submitted to the learning process [105]. However, the local switch-DGA MAC-table may not yet contain a valid entry (or a different earlier mapping). This is illustrated in Figure 4.11, where the switch-DGA local MAC table contains an earlier entry for MAC address `y`, identifying switchport `2` as the origin. However, the switch MAC-table is already updated to identify switchport `1` as most recently sending frames with address `y`. In [5] the authors take into account the high costs of the MAC-table update, suggesting a 60 second update interval, which as we explain below gives ample opportunity to an attacker wishing to exploit the race condition. Our approach challenges the attacker by moving the logging modules closer to

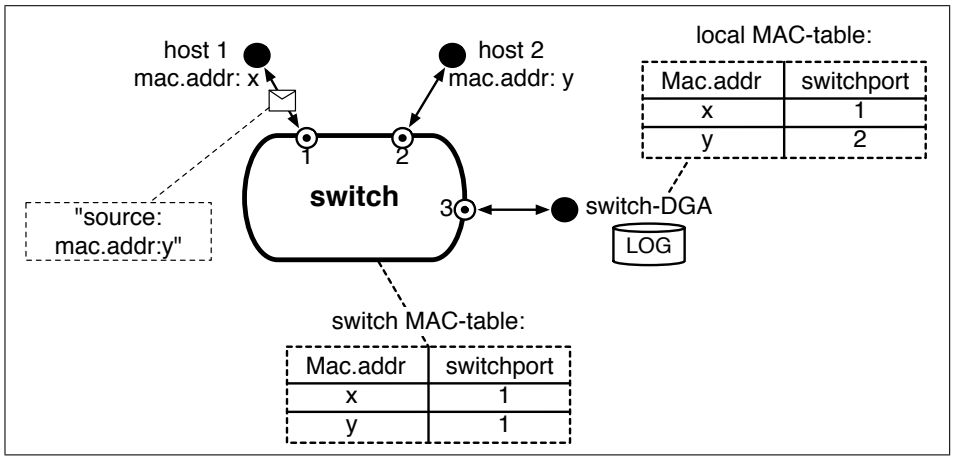


Figure 4.11: Potential discrepancy between the switch-DGA local MAC-table and the switch MAC-table

the switches, making a more frequent MAC-table update feasible.

In order to exploit the race condition an attacker needs to carefully time packet transmission. Two scenarios are possible; the attacker can use the MAC address of another switch host, or an altogether different address. The former is potentially the more 'successful', as the attack is attributed to an innocent tertiary host. However, it is also non-trivial; the attacker must first determine the tertiary host's MAC address (requiring some effort given the switched environment). Furthermore, the attacker needs to 'know' when the tertiary host is transmitting. Lets assume the attacker on port 1 and the tertiary host (MAC address y) on port 2, as is depicted in Figure 4.11. Once attack frames are sent, the switch MAC-table will contain a mapping for address y on port 1 (i.e. the attacker alters its source MAC to be the same as host 2). If the tertiary host transmits a frame before the switch-DGA MAC-table update, then the mapping at the switch MAC-table is overwritten. Thus once the switch-DGA MAC-table update occurs, the attack packets would be mistakenly attributed to port 3. A more frequent MAC-table update, as is possible in switch-SPIE makes this attack more difficult. Another possibility is for the update interval to be randomised, further challenging the attacker. The approach taken by Hazeyama et al in [5] allows a wide window of opportunity for the attacker with the MAC table update time set at 60 seconds, due to the centralised architecture of a single logging xDGA.

If on the other hand the attacker uses an altogether different address and transmits just after a table update, the switch-DGA (local) MAC-table wont yet contain an appropriate mapping for that address. Under 'normal' conditions, we expect that a mapping wont exist for an address when a host has just joined the network. If addresses are consistently not being resolved, then this warrants investigation by network administrators. It is trivial to add a further bloom filter to the switch-DGA for 'unresolved' frames (with an alarm attached to a threshold frame number). A further refinement to our system which we are currently exploring (and that further challenges the attacker) is to

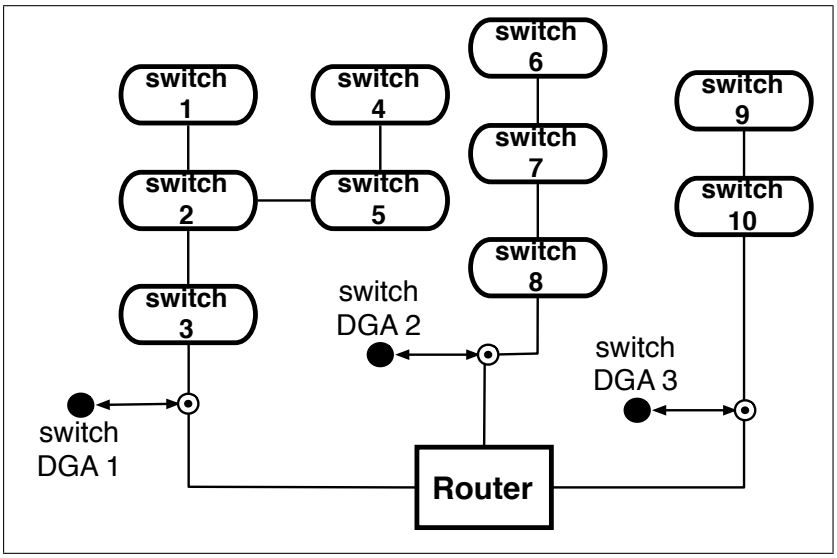


Figure 4.12: An example of switch-SPIE partial deployment

'buffer' incoming frames until the local MAC-table is 'refreshed', before resolving the switchport. This feature has in fact been incorporated into our second L2 traceback system, COTraSE presented subsequently.

Though our approach addresses many of the issues of [5] it is more expensive in terms of required hardware. A dedicated logging node is necessary at each switch. Partial deployment is possible, whereby a switch-DGA is responsible for traffic from a number of switches. As we can see from Figure 4.12, the switch-DGAs move from a switch monitor port to a network tap. However, we begin to lose the advantages afforded by logging at each switch. Specifically, we sacrifice local traffic visibility, as it is not possible to capture traffic unless this traverses a logging node (for instance sent between two hosts on the same switch). More significantly however we also start to sacrifice local MAC-table validity. In Figure 4.12 switch-DGA1 must maintain MAC-tables for 5 switches, incurring higher update costs. Each switch-DGA must maintain an array of bloom filters for each switch. Furthermore, switch-DGA2 will not see packets sent from hosts on switch6 to hosts on switch 7. However, this still improves on the single xDGA approach, and switch-DGA placement can be adapted to the deployment network.

4.6 Chapter Summary

This chapter has presented our first proposal for an L2 traceback system, that is an adaptation of the Source Path Isolation Engine SPIE proposed by Snoeren et al [3]. Our work, switch-SPIE is designed for switched ethernet networks and takes advantage of the MAC address table maintained by ethernet switches to attribute each frame to the sending switchport. Whereas SPIE DGA used

a single bloom filter for logging packets, switch-DGA uses an array of n bloom filters, one for each switchport. An additional step in the logging process identifies the instigating origin switchport from a local copy of the switch MAC-table and stores a given packet into the corresponding bloom filter. Our proposal improves over an earlier adaptation of SPIE for switched ethernet [9] by deploying logging at each switch, rather than at a single network location. This allows for traceback of frames that are entirely local (i.e. that do not traverse the gateway router) and also allows the local MAC-tables maintained by each logging node to be updated with a higher frequency (thus increasing the reliability of the traceback result).

Our evaluation of switch-SPIE memory requirements shows that memory utility is manageable in current switched ethernet networks (100Mbit/s or 1Gbit/s ethernet) but that more efficient logging mechanisms are warranted for faster networks (such as 10Gbit/s ethernet). Empirical evaluation of the switchport mirroring mechanism on our deployment switch showed that the switch itself can handle the extra load incurred by mirroring traffic. However, the recipient switchport and network host (the logging switch-DGA host) are quickly overwhelmed by the traffic load. These two issues served as motivation in our development of COTraSE, our second L2 traceback proposal which we present in the next chapter. Figure 4.13 summarises the placement of sections in our description of switch-SPIE to aid the reader.

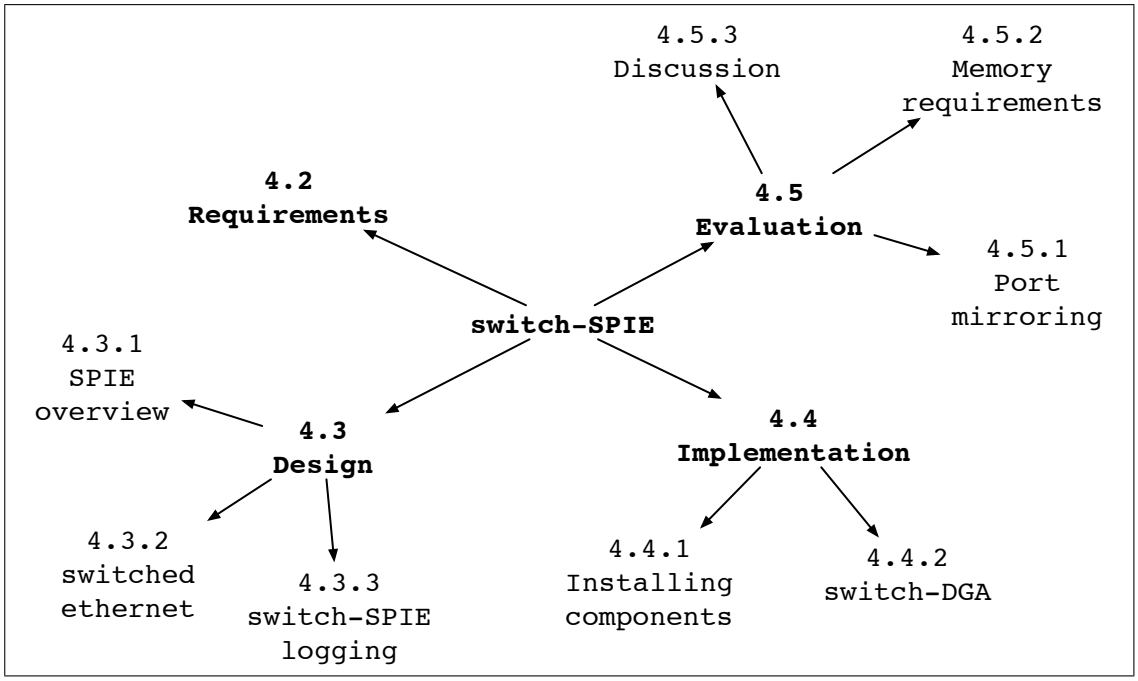


Figure 4.13: Idea map for Chapter 4 showing placement of sections

Chapter 5

COTraSE

5.1 Introduction

This chapter concerns our second L2 traceback system which was presented as "COTraSE: **C**onnection **O**riented **Tra**ceback for **S**witched **E**thernet" at IAS 2008 [13, 12]. Whilst interesting and more challenging than at first appears, logging or marking of network traffic in order to identify the instigating origin of that data can be regarded as a rather academic exercise. That is, in practice one may find it difficult to convince network administrators of the need to provide such fine grained accountability over the traffic that they handle. IP traceback systems such as SPIE and Layer 2 traceback systems such as switch-SPIE are expensive to install but also to maintain (consider securely managing large volumes of logged data).

None the less, in 2006 the European Parliament passed Directive 2006/24/EC "on the retention of data ..." [16]. We consider this directive in some detail in the next chapter. The core requirement is for providers of "publicly available electronic communications services" to maintain "communication records", akin to those maintained by providers of mobile telephony services. That is, the source, destination, duration and type of communication service used. Given that such a "data retention" system is essentially a packet log, then logging based IP and L2 traceback are particularly relevant and offer valuable insights of the practical difficulties in the implementation of such a system. The EU data retention directive in part inspired the development of COTraSE, which adopts a "connection oriented" approach to logging. As we will see, all ethernet frames are processed to deal with sender spoofing, but only a subset are eventually logged as "connection records". Crucially this does not sacrifice the ability to traceback any given packet.

We first revisit our L2 traceback system requirements as used in switch-SPIE and discuss how COTraSE improves on these. This is then followed by the design and implementation details in subsequent sections.

5.2 Requirements

The experience gained when developing switch-SPIE was instrumental in the design of COTraSE. Switch-SPIE like COTraSEis a logging based layer 2 traceback system designed for switched ethernet networks. Furthermore the two systems share the same set of requirements (see 4.2), though COTraSE offers improvement in multiple dimensions. We now revisit these requirements for convenience, followed by a discussion of the advancements claimed by COTraSE.

1. Identifies the instigating origin sID, pNO of any given packet originating within the deployment network (within a bounded traceback window)
2. Requires no changes to existing network infrastructure or protocols.
3. Traceback enabling procedures (i.e. logging) should be considerate of user privacy.
4. The traceback result must be reliable.
5. Supports partial deployment and be adaptable to any network topology.
6. Realistic expectations of memory requirements at packet logs.

With regards to identifying the instigating origin, COTraSElike switch-SPIE assumes that the source MAC address of all frames has been subjected to sender spoofing and so cannot be taken "at face value". However COTraSEgoes further to allow for post-send legitimate spoofing, such as when Network Address Translation (NAT) is employed. We will see how this is achieved by maintaining 'Translation Records' at the gateway router. Other logging based L2 traceback systems including switch-SPIE do not take this type of spoofing into account. Thus, it is not possible to match the IP data delivered to the intended recipient with the same data prior to any alterations made at the gateway router. This of course greatly decreases the utility of such systems in 'real world' deployments, given the ubiquity of legitimate post-send spoofing mechanisms such as NAT and web caching as was discussed in Chapter 2. This ability of COTraSEto traceback regardless of any legitimate post-send spoofing is also relevant to our second requirement as it removes the conflict between logging based L2 traceback systems and widely implemented mechanisms such as NAT. It should be noted that translation logging is orthogonal to the issue of L2 traceback; that is, it is possible to adapt the translation logging process of COTraSEfor other L2 traceback proposals such as switch-SPIE.

With regards to the third requirement, COTraSEprotects user privacy through encryption of logged data. This was also the case in switch-SPIE, where encryption served the dual purpose of decreasing memory requirements as well as protecting user privacy. COTraSEdoes not rely on encryption to decrease memory requirements and instead adopts a "connection oriented" approach to logging. Thus, the use of encryption is entirely for the preservation of user privacy. It could even

be argued that the use of encryption in COTraSEis "overkill" as the data logged is drawn only from the headers of the Data-link, Network and Transport layers, with the payload not being processed at all. This means that in situations where performance of logging nodes is an issue then connection records may be stored in cleartext, avoiding costly cryptographic hashing routines.

The reliability of the trace result is an area where COTraSEmakes significant contributions. We feel that this requirement is particularly important in light of the intended use of retained data under EU directive 2006/24/EC. It **must not** be trivial to employ sender spoofing to create traffic that implicates an innocent third party. COTraSEdoes not rely on bloom filters to decrease memory requirements and so false positives are eliminated alltogether. Furthermore, our switchport resolution algorithm (attributing a frame to a source switchport based on the source MAC address) makes use of two switch MAC tables to make it much harder for an attacker to manipulate the MAC-table race condition, as discussed in the previous chapter and illustrated in Figure 4.9. Finally, COTraSEdoes not rely on switchport mirroring to overcome the switched ethernet traffic visibility issue. The unreliability of port mirroring as the number of switchports being mirrored to the monitor port is increased was illustrated in our experiments. Rather, COTraSElogging procedures are deployed at a passive tap between network switches which means that all frames are processed regardless of load. Even when link ports are an order of magnitude faster than access ports (e.g., 1000Mbit/s vs 100Mbit/s), traffic carried by the link ports will still be less than that from the amalgamation of n access ports to a monitor port (it is often the case that there are more than 10 access ports on a mid range ethernet switch).

COTraSEoffers increased support for partial deployment. In switch-SPIE we logged data from access ports, whilst switch link ports were not monitored at all (as illustrated in Figure 4.2). However, this is reversed in COTraSEwhere we exclusively log at switch link ports. An ideal deployment of COTraSErequires logging between every network switch as only the logging node adjacent to a given frame's origin switch can provide us with the unique sID, pNO that identifies the instigating origin. However given that we monitor link ports, a frame will be processed by all logging nodes en route to the gateway router. Thus, all logging nodes will 'point' towards the instigating origin within the local network so that even in a partial deployment COTraSEcan aid in identifying this host.

Finally, a decrease in memory requirements is another important contribution of COTraSE. The aforementioned EU Directive 2006/24/EC requires that network data be retained for "periods of not less than six months and not more than two years". This is a much larger "traceback window" than that considered by any known L2 traceback systems; in switch-SPIE for instance we discussed memory requirements in terms of minutes, not months! Thus, the efficiency of logging is a crucial factor in the real world utility of L2 traceback systems. In COTraSE, though all frames are processed only a subset are eventually logged as connection records. We have implemented the core logging algorithms and used anonymised WAN traces from [110, 111, 112] as input to these, to enable a

discussion of memory requirements later. We will now turn our attention to the design details COTraSE.

5.3 COTraSE

In COTraSE, rather than switchport mirroring we instead log between switches with a passive 'tap'. Memory requirements are decreased by attributing frames to 'connections' and logging representative 'connection records' (*conRecs*) for each interval rather than explicitly logging all frames. We note that a similar approach is used in router-deployed NetFlow [20], which is tailored to performance management, not traceback. We will examine the important differences between NetFlow and COTraSEin our evaluation later in this chapter.

The basic realisation behind connection oriented logging is that packets exchanged between peer network processes at a given time will have the same source and destination Data-link (MAC), Network (IP) and Transport layer (TCP or UDP) addresses. This data is used as a connection identifier (*conId*) for a given communication. One can group packets in 'connections' even in the absence of TCP or UDP at the transport layer, as we will see. A separate process at the network edge maintains 'translation records' (*transRecs*), addressing the issue of post-send legitimate spoofing (as in NAT).

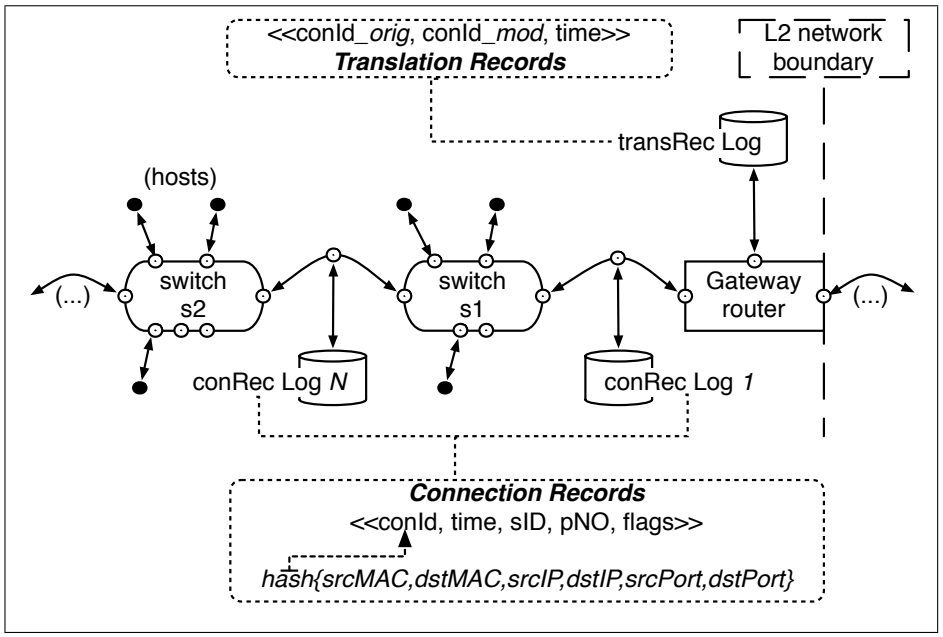


Figure 5.1: COTraSE - *conRec* and *transRec* Logs

An overview of the COTraSEsystem is given in Figure 5.1. As can be seen a conRec log is deployed between ethernet switches, processing traffic from the switch link ports. This may be achieved using hardware dedicated to this purpose, or with an inexpensive ethernet hub placed between the two

switches. Though all ethernet frames are processed and undergo "switchport resolution", only a subset is eventually logged as connection records as we will see. While there may be a number of conRec logs only a single translation record log is required per connection and placed at the gateway router.

For the remainder of this section, we first examine the logging procedures at the connection record logs. This is followed by a discussion of the key subroutine of the logging process, that of switchport resolution. Finally we discuss the two key timing parameters used by COTraSEand explain the translation record logging process at the gateway router.

5.3.1 The Connection Record Logs

Each connection record (*conRec*) consists of connection identifier (*conId*), timing and instigating origin information. As in switch-SPIE each host on the layer 2 network is identified by the unique pairing of switch identifier sID and switchport number pNO.

We use a cryptographic hash function over the data-link, network and transport layer addresses (`MAC, IP, TCP/UDP` data) to derive the connection identifier (*conId*) for each received frame. These addresses can uniquely identify a specific communication between two peer processes only if they are taken together.

For each frame received at a conRec log, *conId* is computed and used to make a new working record (*wRec*). After switchport resolution (explained subsequently), each *wRec* becomes 'active' only if there is no existing active *wRec* with the same connection identifier. This means that at a given time only one active *wRec* exists for each unique communication between peer network processes.

The set of active working records $\{^{A}wRecs\}$ represents all currently ongoing communications. The active wRecs are cleared as per the 'active connections purge interval'. At the end of each purge interval, for every active *wRec* $w \in \{^{A}wRecs\}$ a permanent connection record is created and logged. The next frame for each connection will then become the new active *wRec* and a new set of active working records is populated and maintained until the end of the next purge interval.

A conceptual overview of this process is given in Figure 5.2, which depicts the progression from ethernet frame to connection record, for 12 frames of the same connection. The horizontal axis shows time expressed in terms of purge intervals of length t. During the first interval $0t \leftrightarrow 1t$ two frames are received and a *wRec* created for both. However, only the *wRec* of frame 1 becomes active whilst the *wRec* of frame 2 is discarded (shown as X). After the first purge interval has elapsed ($1t$) the information contained within active *wRec* 1 is used to create conRec A. Thus at the end of the depicted time period ($6t$) a total of five connection records are required to represent the 12 frames.

When processing traceback requests, COTraSEcan provide the time periods t during which a given frame was processed (if at all). There is an inherent space/time tradeoff in setting the length

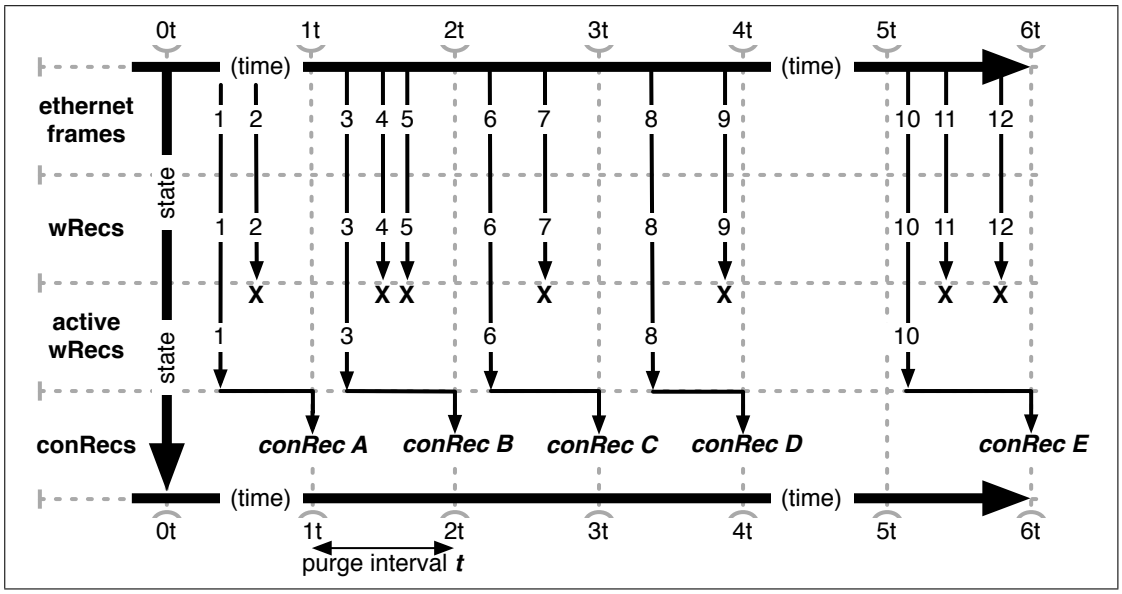


Figure 5.2: Conceptual view of the progression from ethernet frames to connection records. The Figure depicts conRecs from 12 frames of the same connection

of the active connections purge interval. A larger interval means that more frames are represented by a single connection record, thus decreasing memory requirements. However this also means that traceback replies will become less precise. COTraSEcan only confirm that a frame was processed within a given purge interval but cannot identify the precise time this occurred. We will examine the bounds of the purge interval in our discussion of timing parameters later.

For non `TCP/UDP` traffic and in the absence of Transport layer port numbers, deriving the *conId* requires us to find a different means of grouping consecutive frames from the same communication. Source and destination MAC addresses are used in conjunction with any other available data. Of course the grouping of consecutive frames is protocol dependent and we give two characteristic examples as it is impossible to provide an exhaustive list. For `ARP` one can use the `Sender` and `Target Protocol Address` together with the `Opcode`, whilst for `ICMP` the `IP` source and destination addresses are available, and used in conjunction with `ICMP Type` and `Code`. We have been unable to identify a networking protocol for which some means of grouping is not possible. However, even in such a case the logging procedures will degenerate to no worse than an explicit frame log, which is the de facto modus operandi of all other known logging L2 traceback systems.

The *conRec* logging algorithm is given later in Figure 5.5, as we must first introduce the key subroutine of switchport resolution. At each *conRec* log and for each frame received the connection identifier is computed and a new working record created. COTraSElike switch-SPIE determines the origin sID and pNO by consulting local copies of the switch MAC address table. We now turn to the details of this "switchport resolution" process.

5.3.2 Switchport Resolution

As mentioned earlier the source MAC ($srcMAC$) address cannot be taken 'at face value' as it is easily spoofed at the instigating origin host. Switches 'learn' the $srcMAC \leftrightarrow pNO$ mapping from each forwarded frame and update their MAC-table accordingly [104, 105]. As in other proposals, we maintain local copies of the switch MAC-table to determine the switch identifier (sID) and switchport number (pNO) based on the source MAC address. As in switch-SPIE earlier, this process is termed 'switchport resolution'. The novelty of the COTraSEapproach is to use two MAC-tables. Whilst this requires an extra MAC-table lookup it also enables us to perform detection of sender spoofing activity. We classify the return of each MAC-table lookup and show that some combinations are erroneous. A 2-bit flag field is required in all working and connection records to store the outcome of switchport resolution. The value of this field directly depends on the combination of the two MAC-table queries. Importantly the 2-bit flag allows us to reason about the reliability of a given connection record when querying logged data. We now introduce our simple classification and discuss how this is used to detect spoofing.

We correlate the MAC-table values from both adjacent switches to determine $\{sID, pNO\}$. Each switchport is either an access port, providing network access to end hosts, or a link port which leads to another switch or router. This designation was also used in switch-SPIE though COTraSEfurther differentiates between internal and external link switchports. This is shown in Figure 5.3, where ports $\{s1, 10\}$ and $\{s2, 9\}$ are the internal link ports for the shown conRec log, whilst ports $\{s1, 9\}$ and $\{s2, 10\}$ are the external link ports. Of course this is relative to a specific connection record log; at the next conRec log beyond switch s2 (not shown), port $\{s2, 10\}$ would be an internal link port and port $\{s2, 9\}$ would be an external link port.

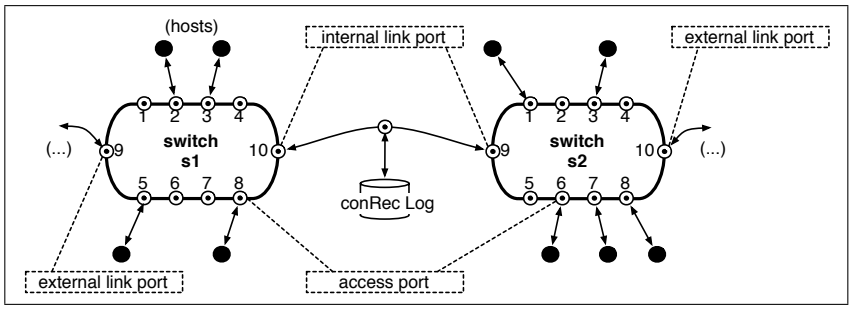


Figure 5.3: COTraSEclassifies all switchports as one of access, internal link or external link

This classification is configured at each conRec log, allowing the return of MAC-table lookups to be classed as one of: null, external link (*ext_link*), internal link (*int_link*), access port (*axs_port*). It is likely that the MAC-tables from both switches will contain an address mapping for a given address. Based on our classification, some combinations are not acceptable, and generally indicative of an error as will be explained below. One legitimate combination for instance is for the (local)

MAC-table of switch $s1$ to report an internal link whilst the MAC-table for switch $s2$ gives an access port. This is shown as case 8 in Table 5.1 which gives the set of all possible outcomes. In the Table the number of each case matches the corresponding circled number in Figure 5.4. The meaning of the 2-bit flags, which are also shown in Figure 5.5 will be explained subsequently.

Table 5.1: Summary of possible switchport resolution outcomes at a given conRec log

Case	Switch 1	Switch 2	Flags	Outcome
1	null	null	01	Alert
2	null	axs_port	00	Normal
3	null	int_link	11	Alert
4	null	ext_link	00	Normal
5	ext_link	axs_port	10	Alert
6	ext_link	int_link	00	Normal
7	ext_link	ext_link	10	Alert
8	int_link	axs_port	00	Normal
9	int_link	int_link	10	Alert
10	axs_port	axs_port	10	Alert

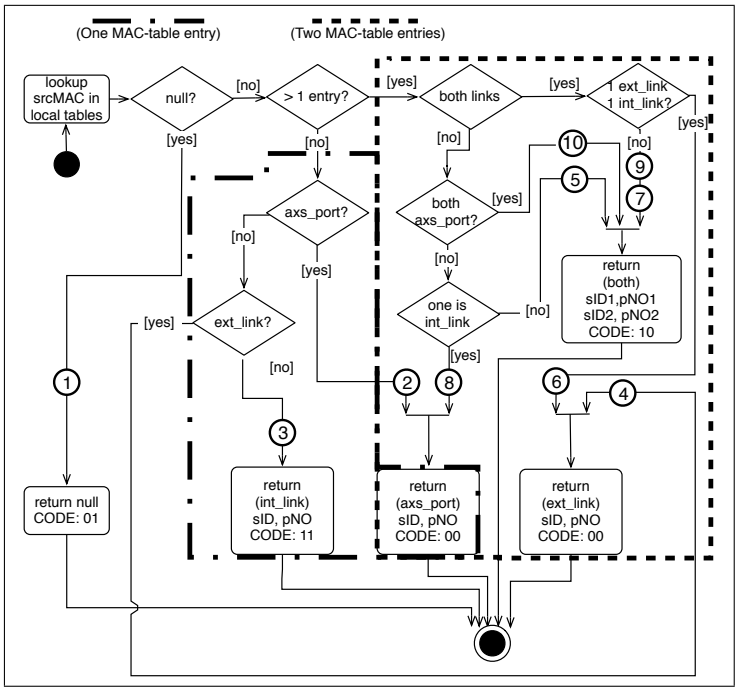


Figure 5.4: The switchport resolution algorithm

When both switches return a non-null value (cases 5 to 10 in the Table), switchport resolution must 'decide' which of the two should be used as the $\{sID, pNO\}$ identifier. An axs_port gives a specific network access point and so is always chosen if available. If both tables return links, then

an ext_link is preferred over an int_link, as in the former case neither adjacent switch was the origin.

Figure 5.4 shows the switchport resolution algorithm. As can be seen, the algorithm makes use of a 2-bit flag to signal the outcome of MAC-table lookups (corresponding to the Flags column in Table 5.1 above). Code 00 indicates 'normal' switchport resolution, that is, a single axs_port (cases 2 and 8) or ext_link (cases 4 and 6) is returned. Code 01 means a port mapping was not available for an address, corresponding to case 1. If MAC addresses are consistently not learnt by switch MAC-tables, then this may indicate an oversubscribed switch [104].

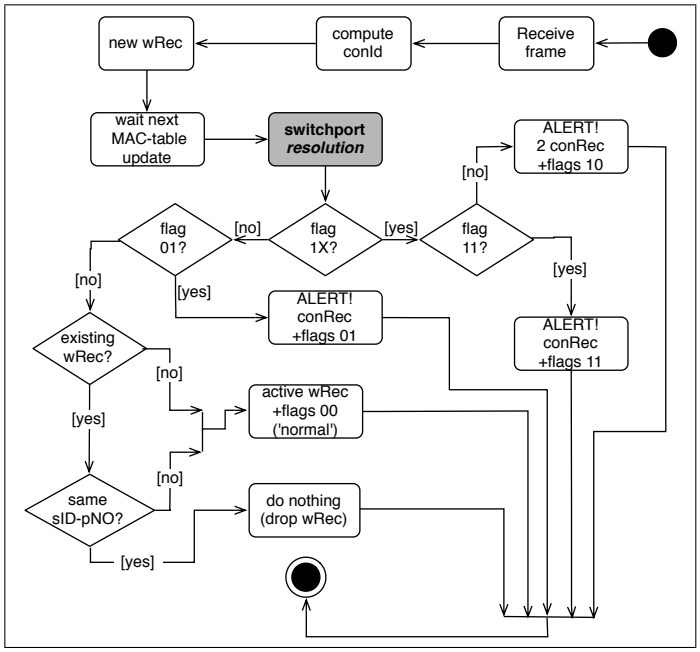


Figure 5.5: Activity diagram showing the conRec Log algorithm

Figure 5.5 depicts the entire connection record logging algorithm, where switchport resolution is shown as a shaded activity box. Those wRecs with codes other than 00 are never made 'active', as they indicate 'abnormal' switchport resolution. Depending on available information one or more conRecs are immediately created, with the error codes indicating their status, as in Figure 5.5. Codes 10 and 11 signal 'erroneous' switchport resolution. In Figure 5.4 and Table 5.1 cases $5, 7, 9$ and 10 produce error code 10 and case 3 produces code 11. The former indicates that we cannot 'choose' between conflicting or equivalent values. For instance, two ext_links or two int_links would mean that the same MAC address was 'seen' as a source address from two opposing directions. This is impossible as switched Ethernet does not permit cycles[104].

Code 11, produced in case 3 means there is insufficient information as only an int_link is returned. In this case, switch $s2$ points to switch $s1$ as the origin, by returning the int_link from the MAC-table lookup. Given that we perform switchport resolution after local MAC-tables are updated we would expect switch $s1$ to return either an axs_port or an ext_link.

As is shown in Figure 5.5, after local MAC-tables are updated and switchport resolution occurs, the conRec logging algorithm 'decides' what should occur to each working record based on the 2-bit flags. We now turn our attention to the two important timing parameters of the MAC-table update time and the active working records purge interval, before considering the (much simpler) translation records maintained at the gateway router.

5.3.3 Timing Parameters

The MAC-table update time dictates when the local MAC-tables maintained at the conRec logs are refreshed from their corresponding 'master' tables on each of the two adjacent switches. As was shown in Figure 5.5 once an ethernet frame is received, its resulting working record is buffered to undergo switchport resolution only after the MAC-table update. It is possible that the local MAC-tables do not yet contain an entry for the source address of a recently received frame, such as when a new host joins the network. Its likelihood is reduced by deferring switchport resolution until after the local MAC-tables are updated as the switch MAC tables will contain an entry for all recently 'seen' source MAC addresses [104, 105].

Thus, we must ensure that the update time is less than the MAC address Table 'aging time'. All switch MAC-tables will "age out" entries for efficiency, but also to ensure that hosts which move to different parts of the network are not permanently prevented from receiving frames [105]. That is, if the MAC-table update time is too large it is possible to 'lose' the $srcMAC \leftrightarrow pNO$ mapping from the switch MAC-tables so that only `null` is returned by the switchport resolution process (case 1 in Figure 5.4 and Table 5.1).

The IEEE suggest an aging time of between 10 and $1,000,000$ seconds [105], with a suggested standard of 300 seconds. However, since unprocessed wRecs are buffered until the next MAC-table update, the logging process's memory utilisation ("working" memory - RAM) becomes an important and practical concern. With this in mind we choose a MAC-table update time of 5 seconds, which also stays within the minimum aging time of 10 seconds. This creates a worst case requirement of $\approx$ 300Mbytes of RAM on a full duplex 1Gbit/s link to buffer unprocessed wRecs, as we will see in our discussion of memory requirements later (assuming 1000 bit frames).

The second important timing parameter we consider is the active working records purge interval. This is the time after which the set of active working records is cleared and a corresponding set of connection records created and archived. As was mentioned earlier, the purge interval (*purgeInt*) is governed by an inherent time/space tradeoff. A larger purgeInt means a smaller proportion of frames become conRecs overall with each conRec representing a larger time interval. As can be seen in Figure 5.6, with a purgeInt of $1t$, conRec A 'represents' frames 1 and 2. If however the purgeInt were set to $2t$, then conRec A would have represented frames $1, 2, 3, 4$ and 5.

When replying positively to traceback requests (i.e., "yes I processed that frame") COTraSEwill

express the time that the given frame was processed in terms of purge intervals. Thus, whilst a larger purge interval decreases conRec storage requirements, it also decreases time precision of traceback replies. The purge interval is therefore bound to be no greater than is acceptable for the purpose of traceback request processing. However, to limit the search for records to only two purge intervals, it is also bound to be no smaller than the time we expect between processing a frame and its arrival at the intended destination as we explain below (equivalently not less than half the expected round trip time).

Traceback requests provide the time when a given frame to be traced was observed, either at the destination or at some suitable vantage point within the network (e.g., at an Intrusion Detection System). Based on this time we must search the connection record logs for those conRecs, if any, that represent the requested frame. If we ensure that the purge interval is no smaller than half the round trip time (RTT) then we can limit the search to only two purge intervals. As can be seen in Figure 5.6, frame 1 is received and processed at the conRec log at time $0t + \delta$, where $0 \leq \delta < t$. Frame 1 reaches its destination at time $(0t + \delta) + \chi$ and so χ represents the 'delivery time'. If we ensure that $t \geq \chi$ then frame 1 will be delivered during the current or next purge interval (i.e., $0t \leftrightarrow 1t$ or $1t \leftrightarrow 2t$ respectively). That is, with a purge interval of $t \geq \frac{RTT}{2}$ the conRec for time $(0t + \delta) + \chi$ or the previous one 'represent' the tracked frame.

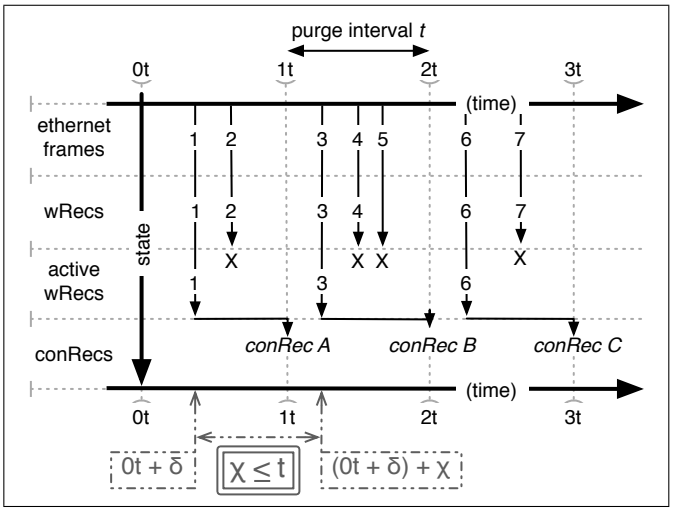


Figure 5.6: The purge interval t must be greater than the time χ we expect a frame to take in reaching its destination

We conservatively choose 10 seconds as the lowest bound on the purge interval. We sampled data from the Skitter "Macroscopic Topology Project" [113] for 3 arbitrarily chosen days that are each one year apart. Each data set shows the RTT for packets sent from a number of skitter nodes that were active on that day. For each node, we use the RTT distribution by continent, that is, the RTT from the given skitter node to servers around the globe (Africa, Asia, North America, South

Table 5.2: Summary - Skitter [113] RTT for all active servers on 3 arbitrary days

Day	Participating skitter nodes	RTT for 99% of packets (ms)
02 Mar 2002	13	2050
02 May 2003	23	4709
02 Mar 2004	21	3941
RTT for 99% of packets over 3 days: 3566		

America, Middle East, Europe, Oceania).

We conservatively estimated that (at the $99th$ percentile) the round trip time is ≈ 3.5 seconds, as summarised in Table 5.2. For instance for data from 02 May 2003 and for each of the 23 available measurements we took the worst case (largest) round trip time at the $99th$ percentile and then calculated a mean value for all servers. Thus we feel that our choice of 10 seconds as the lowest bound on the purge interval is sufficient for $t \geq \chi$. Of course as 10s is the lowest bound, it also produces the greatest number of connection records and so represents the worst case in terms of memory utility. We use this value in our implementation of the conRec log so that our discussion of memory requirements given later realistically portrays this worst case scenario.

5.3.4 The Translation Record Logs

Traceback requests will specify a traced packet in the format it was received by the recipient. As we have seen, when legitimate post-send spoofing such as NAT is in place, the source IP address, source TCP port and destination TCP port are all subject to change at the gateway router. The translation records are a mapping between the connection identifiers before and after any of the address fields are re-written. The original SPIE IP traceback system addressed this issue by way of a 'Transform Lookup Table' [3] and COTraSEis the only L2 traceback system we are aware of to address this. Each translation record contains *conId_orig*, *conId_mod* and a timestamp. Our approach is simple in that a direct mapping is maintained between data before and after modification. We are confident that a more elegant solution is possible and this is deferred to future work.

At the translation record (transRec) logs, the original connection identifier (conId_orig) is computed in the same way as the *conId* at the connection record logs. That is, a hashed digest over the concatenated source and destination addresses from the Transport, Network and Data-link layers (`TCP/UDP` ports, `IP` and `MAC` addresses, respectively). However, the modified connection identifier cannot include any data that is not received by a given frame's recipient. When servicing traceback requests, we need to match the data supplied by the trace initiator (i.e. the traced packet) with data we have stored in our logs. For frames that leave the local network, we expect the application, transport and network layer data to arrive unchanged (i.e. in the same form with which it left the local network). However we can make no assumptions about the physical (and by extension) data

link layers employed to reach the intended destination (for instance ADSL or DOCSIS fibre links may be used). Thus, the source and destination MAC addresses are *not* included in the computation of *conId_mod*. The transRec logging algorithm is shown in Figure 5.7.

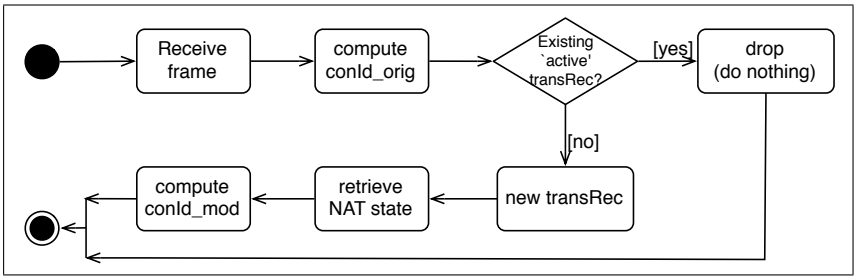


Figure 5.7: The Translation Record logging process

The transRec log maintains an 'active connections' set in a similar manner to the conRec logs. The processing algorithms however are much simpler, in anticipation of the fact that the logging procedures at a leaf router may handle a greater volume of traffic. As can be seen in Figure 5.7 no attempt is made to perform switchport resolution as this is done at all conRec logs en route to the leaf router and doing it again provides no new information. When a frame arrives at the router, the original connection identifier (conId_orig) is computed and a check made for an existing translation record in the 'active transRecs' set with the same conId_orig. If there is a match then the given frame is no longer processed and 'dropped'. If however there is no existing active translation record, then a new transRec is created. The connection's state is retrieved (e.g., from the NAT processes or from the web proxy) and used to compute the modified connection identifier (conId_mod). A time-stamp is then added to this and the new transRec is placed in the leaf router 'active connections'. After the active connections purge interval has elapsed (as discussed above), the active connections set is cleared with the transRecs being archived to storage for subsequent traceback processing.

For traceback requests from beyond the local network, transRecs are searched for conId_*mod*. If a match is found, a local traceback request is dispatched, and conRec logs queried based on conId_*orig*. Thus we 'traceback' packets regardless of post-send legitimate spoofing at the leaf router.

5.4 Memory Requirements

We now turn our attention to a discussion of the memory requirements created by a COTraSEdeployment. This is an obvious and pressing issue for any logging based L2 traceback system. As we will see our implementation of the connection record log demonstrates the drastic reduction in the storage needs of a connection oriented approach, compared to explicit frame logging.

Memory requirements are an important concern for logging based IP and L2 traceback systems as they govern the 'traceback window'. This is the time during which we can traceback a packet after

it has been logged. That is, the length of time for which it is feasible to retain logged connection records, before the oldest conRecs are overwritten with new data. Below we give details of the various COTraSE records encountered so far as these are used in the discussion that follows (brackets denote size in bits):

wRecs (230): *conId* (128), source MAC (48), sID (10), pNO (10), timestamp (32), flags (2)

conRecs (182): *conId* (128), sID (10), pNO (10), timestamp (32), flags (2)

transRecs (288): conId_orig (128), conId_mod (128), time (32)

You may recall that the MAC-table update time directly impacts the 'working' memory requirements (RAM). All working records created from received frames are buffered to undergo switchport resolution directly after the MAC-table update. Taking a duplex FastEthernet link at 1Gbit/s and with average frame size of 1000 bits, at most one gets $\approx 1,000,000$ frames per second in either direction. Assuming a hypothetical worst case scenario where all frames belong to different connections, this would equate to $\approx 10,000,000$ working records in 5 seconds. This scenario is of course highly unlikely and can only arise in a sustained DoS attack targeted specifically at COTraSE(further discussion of vulnerabilities follows separately). None the less even in this extreme case a conRec log requires RAM of $\approx$ 300MBytes for 5 seconds worth of wRecs, which we do not consider excessive.

Storage requirements for archived connection records will ultimately depend on the number of "conversations" between peer machines at a given time and this will of course vary between networks. Recall that a single conRec is required for each ongoing connection for each purge interval. We have hence established bounds using 'real world' network traffic from available WAN traces [110, 111, 112].

The number of connections C gives the lower bound for the number of conRecs cR we need to store for each purge interval p. How many conRecs we store *overall* depends on the 'traceback window', in terms of number of purge intervals. Though C is not within our control, we can tune p, as explained earlier. A larger p will mean less purgeInts in a given traceback window and so less conRecs overall. As we have argued, to limit the search for records when processing traceback requests, a lower bound for p is 10 seconds. This value is used in our experiments so that our results represent the worst case, in so far as the purge interval can affect cR.

In total we processed over $300,000,000$ packets to derive an upper bound for the number of connections C. We cannot necessarily claim that our results represent a general case since C may vary in different scenarios. Furthermore WAN traces are not necessarily representative of LAN traffic, which is the primary focus for COTraSE. However, 'real' LAN traffic is not easily available due to the associated privacy and security concerns.

We first describe the WAN traces in more detail and explain how these were processed to produce a 'hex dump' for each minute of a given trace. This is followed by a description of our implementation of the conRec algorithm. Our code, written in Java, takes the processed trace files as input and

writes connection records to file at the end of each purge interval. This is then followed by the results of our experiments.

5.4.1 The WAN trace data

Table 5.3 summarises the trace data used here. Source 'OC12' [110] provides data from an OC12 link at the AMPATH Internet Exchange in Miami. We arbitrarily downloaded the data for 09 Jan 2007, with a total of 12 one hour traces taken at 3-hourly intervals from 0900 until 0000 of 10 Jan. Source 'OC48' [111] is an OC48 peering link for a large ISP at NASA Ames Internet Exchange. The data we use is for six 5 minute periods: two on each of 14 Aug 2002 (0900), 15 Jan 2003 (0959) and 24 Apr 2003 (0000).

Table 5.3: Trace data summary

Source	Traces	Total Frames
OC12[110]	$12 * 60$mins	$136,503,068$
OC48[111]	$6 * 5$mins	$114,181,288$
WIDE[112]	$6 * 15$mins	$89,357,967$

All data was available in the form of `libpcap` [114] capture files and all traces are anonymised (e.g., IP addresses are changed) to preserve the privacy of network users. The OC48 and OC12 traces are captured separately in each direction of the monitored link; thus, the six traces for OC48 are in actuality either direction at 3 given times. Finally, source 'WIDE'[112] is a 100Mbit/s Ethernet transit link from the WIDE research network in Tokyo to its upstream carrying mainly trans-Pacific traffic. This source provides 15 minute traces and we used data for the same day (and times) as OC12. The WIDE capture files represent both directions of the monitored link.

Despite the WIDE data being duplex and the OC48 and OC12 simplex, we processed all trace files in the same way and treated the simplex traces as independent captures. Splitting the WIDE trace into two simplex traces would be difficult. However merging the simplex OC48 and WIDE traces was possible using the `mergecap` tool, available as part of the `wireshark` [115] network protocol analyser. In any case we decided that this was not necessary. Given the difference in both link speed and capture length for each given trace, it is not appropriate to directly compare the number of conRecs created even if all traces were either duplex or simplex. Furthermore, the conRec algorithm treats each direction of a given communication independently. A different connection identifier is derived for frames flowing from $A \rightarrow B$ than that derived for frames flowing $A \leftarrow B$. Thus, whether these are processed from a single file (in a duplex capture) or from two files (simplex) does not bear upon the total number of connection records created in the given time. As we will see our analysis compares the number of conRecs as a ratio of the total number of frames, to allow a direct comparison between the results for each source.

Each of the six OC48 traces was split using the `editcap` utility into five files, each representing a minute of the given trace (the full 5 minute traces were too large to be processed). This was not necessary for the 12 hourly (simplex) OC12 traces and the six 15-minute (duplex) WIDE traces that were processed in their entirety. Each `.pcap` file was passed through the `tcpdump` utility to produce a text file complete with a 'hex dump' of each packet's data. An example is given in Figure 5.8 which shows data from an OC48 trace. In particular it shows 3 frame representations numbered $2, 3$ and 3739.

As can be seen the start of each frame is denoted by a frame summary before three lines show the base 16 representation of the packet's content, with the ASCII equivalent to the right (non printing ASCII characters are shown as '.'). Each two digit hex code represents a single byte of raw network data. For the OC48 traces the Cisco HDLC data link layer was employed. As can be seen in Figure 5.8, the code 0800 signals that an IP frame follows and 45 is the first byte of IP header. This represents the first two fields of the IP header, showing the IP version as 4 and the IP header length as 5 (32-bit) words. As these fields each occupy a nibble, they are shown together as a single byte in the hex dump. Furthermore, this is interpreted as the character `E` in the ASCII representation of the data as can be seen.

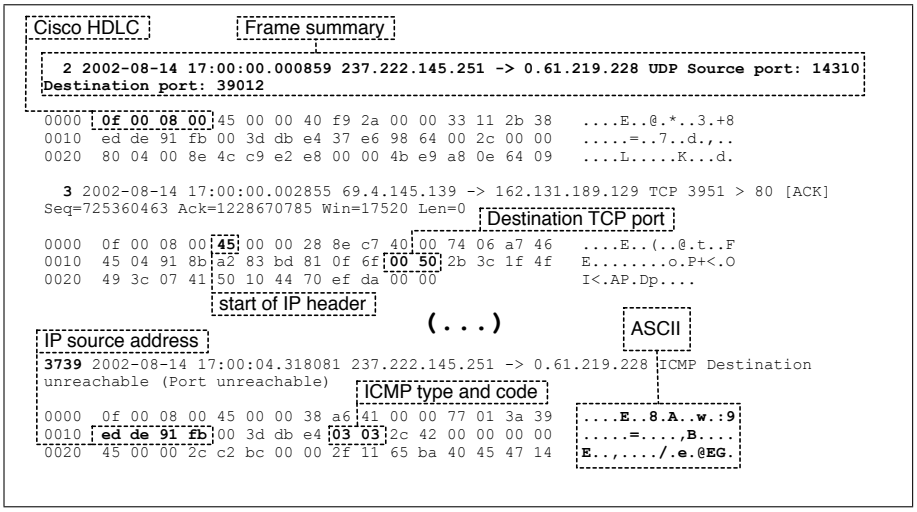


Figure 5.8: Annotated Hex dump of an OC48 .pcap file, ready for processing by the conRec implementation

Each trace file was processed by tcpdump in the same way to produce an equivalent text file complete with the hex representation. Our Java implementation of the conRec logging algorithm takes these files as input and parses the hex byte codes to reproduce the necessary fields for working and connection records. We now turn to the details of this implementation, before presenting our results.

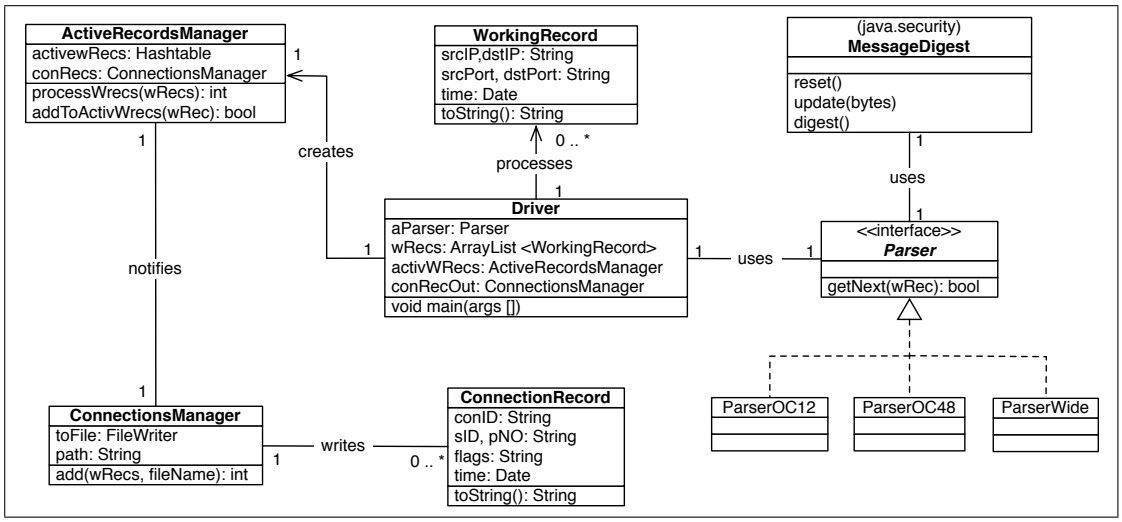


Figure 5.9: UML Class diagram showing the main attributes and operations of our COTraSE implementation

5.4.2 The Connection Record log implementation

Figure 5.9 gives a UML class diagram of our implementation. As can be seen, a main Driver class coordinates all activity. A Parser class was required for each of the three sources due to slight differences in the capture formats. For instance, OC48 employs Cisco HDLC at the Data-link layer (as in Figure 5.8), OC12 is 'raw ip' (no Data-link data) and WIDE uses Ethernet.

Driver takes as input the location of the processed .pcap file for parsing (with hex dump), the purge interval and mac-table update times in seconds and a path to which conRecs are to be written. The Driver then instantiates a Parser and reads the first working record from the input file. We use relative rather than absolute timing; thus, the start time of each capture is taken from the timestamp of the first *wRec* as read from the input file by the Parser. The next and all subsequent purge intervals and mac-table updates are then calculated relative to this.

Our implementation does not perform switchport resolution and so, though present, the sID, pNO and $flags$ fields of a conRec remain as `NULL` . Switchport resolution does not impact upon memory requirements, which is the primary focus of this investigation. The Driver maintains a List of working records, as these are read from the input file by the Parser. The time of each *wRec* is checked against the MAC-table update time which was set at 5 seconds. As soon as a *wRec* is encountered that 'occurs' after the MAC-table update, the current list of wRecs is sent to the ActiveRecordsManager for processing.

The ActiveRecordsManager class maintains the active working records set as a Hashtable, mapping a String connection identifier (MD5 hash calculated by Parser) to the corresponding WorkingRecord object. For each working record sent by the Driver, the ActiveRecordsManager uses the *conId* to update the active wRecs set, as described in Section 5.3.1. Similarly to the MAC-table

update, as soon as a *wRec* is processed that occurs in the next purge interval, the current active wRecs set is sent to the ConnectionsManager to be written to file. The active wRecs are then cleared and the rest of the wRecs are processed if any, thus repopulating the active wRecs.

Our Parser classes processed all `TCP`, `UDP` and `ICMP` frames encountered which on average between the three sources accounted for more than 98% of data. As discussed earlier, the purge interval was set to 10 seconds as, being the lowest bound will produce the greatest number of connection records in a given traceback window. Thus a conRec file was created containing the connection records for each 10 second interval occurring within the given capture time, an example of which is shown in Figure 5.10. We now present the results of this processing and derive estimates for COTraSEmemory requirements.

```
This file (Tue Jan 09 06:08:21 GMT 2007) contains: 23081 conRecs

*****************************************

ConID: bcc598f4983f63f11bf4e1a922f49b1e
Frame no.: NULL time Tue Jan 09 06:08:19 GMT 2007 from 207.232.72.176 to
208.72.155.27 ip_protocol is 01 src TCP/UDP port NULL dst TCP/UDP port NULL
icmp Type/Code 8/0bcc598f4983f63f11bf4e1a922f49b1e*Tue Jan 09 06:08:19 GMT
2007*NULL*NULL*NULL

ConID: 67a050837a6033e40ca8994753d73e24
Frame no.: NULL time Tue Jan 09 06:08:13 GMT 2007 from 164.133.140.130 to
192.240.122.81 ip_protocol is 06 src TCP/UDP port 3124 dst TCP/UDP port
58516 icmp Type/Code NULL/NULL67a050837a6033e40ca8994753d73e24*Tue Jan 09
06:08:13 GMT 2007*NULL*NULL*NULL

                                    (...)
```

Figure 5.10: A conRec file from the WIDE trace data for a given 10 second purge interval

5.4.3 Results

For each source our code created a conRec file for every 10 second purge interval. Our code also created a separate 'debug' file for each processed trace which helped us to verify the values presented here. We used both the conRec and debug files to collate the total number of connection records calculated for each minute (i.e. from each of the six purge intervals occurring therein). Given that the number of connections will vary greatly both due to link speed and network load, it is not appropriate to directly compare the number of conRecs created between the three sources. We summarise some of our findings in Table 5.4.

'Bytes per packet' was taken from the metadata associated with each capture file as indexed at DatCat [116]. This is significant as in our subsequent calculations we take the average frame size to be 1000 bits. As can be seen in Table 5.4 this is a conservative estimate, with the average bytes per packet from the WAN data being more than five times that at 673 bytes per packet. The 'packets per second' column was populated by taking the total number of working records created by the data from each source, and dividing by the total capture time. For example for OC48 we processed $136,803,068$ packets occurring over 15 minutes (processed as 30 minutes of simplex

captures), equating to 128,868 packets per second. The difference in bandwidth between the sources makes calculation of a mean 'packets per second' redundant (hence n/a). The '% unparseable' represents those frames that our parser classes skipped; recall that our code only processed all `TCP`, `UDP` and `ICMP` data encountered. As can be seen, IPv6, ARP and incomplete frames (e.g. due to an error when capturing) accounted for only 1.43% of total frames on average.

Table 5.4: Values from our results

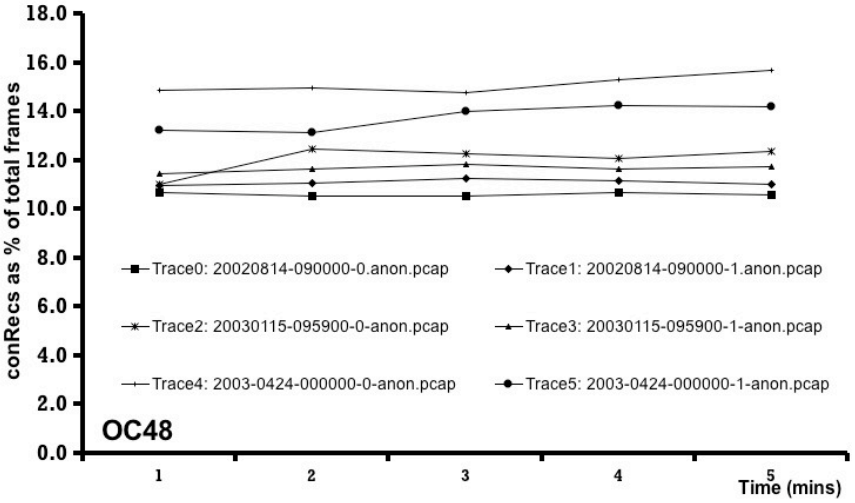
Source	bytes per packet	packets per second	% unparseable !(TCP, UDP, ICMP)
OC48	563	128,868	0.45
OC12	779	6,320	2.76
WIDE	676	16,548	1.07
μ	673	(n/a)	1.43
σ	108	(n/a)	1.20

With a purge interval of 10 seconds, six conRec files were created in each minute of trace data. We took the average number of conRecs for the purge intervals occurring in a given minute and multiplied by 6 to calculate the total no of conRecs in that minute. We then plotted the number of conRecs as a percentage of the total number of frames for each minute of each trace, as shown in Figure 5.11. We see a similarity in the plots and especially so between Figures 5.11(a) and (b). Obviously a lower $\frac{conRecs}{frames}$ ratio signals a greater reduction in memory requirements compared to logging all frames.

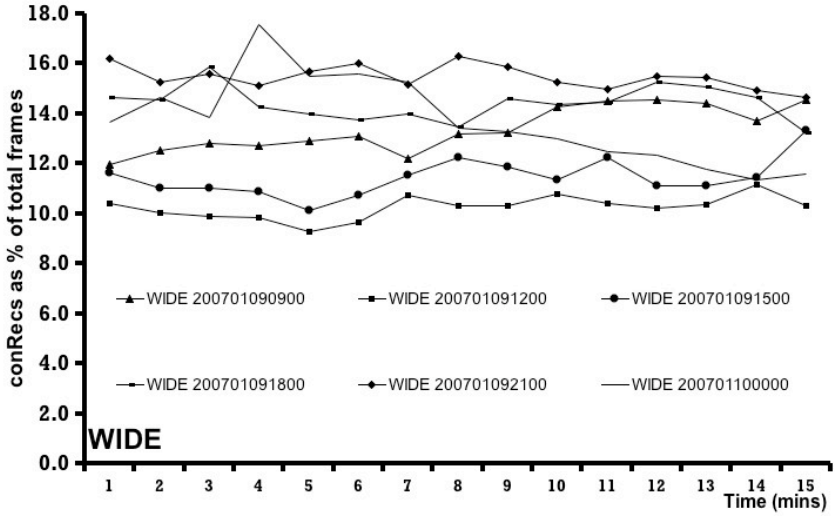
Table 5.5: Predicted COTraSE memory use, with each link at full utility

Source	packets/s (≈)	Megabytes (traceback window) 1 min.	1 hour
OC12	601,300	117	7,045
OC48	2,488,300	486	29,160
WIDE	100,000	19.5	1,171

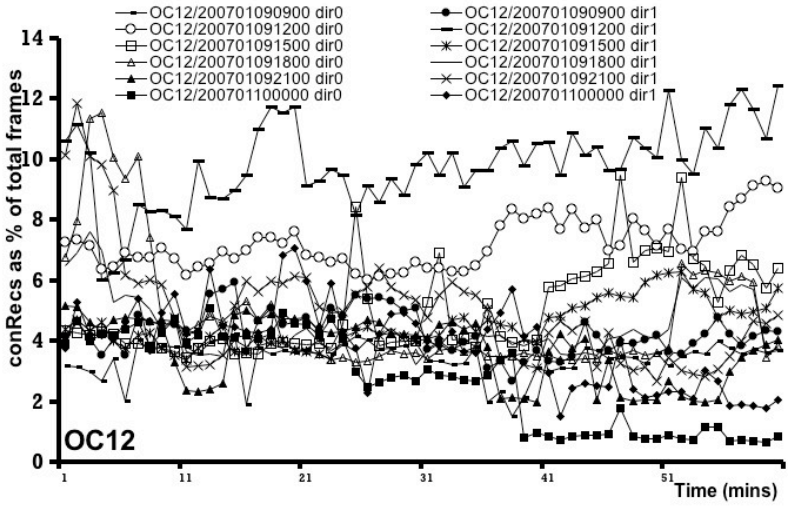
We use $\frac{conRecs}{frames}$ to approximate the number of connections. From Figure 5.11 we take the upper bound for $\frac{conRecs}{frames}$ at 15%. We assume the given link is at full utility and that on average each packet is ≈ 1000 bits, which as we have seen is a very conservative estimate. In Table 5.5 we give the predicted memory requirements of a COTraSE deployment, expressed in megabytes, for providing the given 'traceback window'. For example, in the OC48 case (i.e., 2488.32Mbit/s), taking 15% of 2,488,300 packets per second, and multiplying by 182 bits per conRec gives 486Mbytes for a 1 minute 'traceback window'. The OC48 case is of particular interest as the $\approx 2,000,000$ packets/s throughput is similar to full utility of a duplex 1Gb/s ethernet link (1000 bit frames). As we have already stated it is not possible to derive a general case from this data. The number of connections will vary greatly though it is encouraging that the values we obtained from this 'real world' data are



(a) CAIDA OC48 [111]



(b) WIDE [112]



(c) CAIDA OC12 [110]

Figure 5.11: conRecs as a percentage of frames for each minute of the given trace

consistently low, with the number of conRecs per minute at below 20% of total frames and often closer to 10%.

5.5 Discussion and Deployment Issues

As mentioned earlier, there are similarities between the 'connection-oriented' logging approach taken by COTraSE and Cisco's 'NetFlow' [20, 117]. NetFlow is deployed in Cisco routers and some high end 'multilayer' switches. In this section we briefly consider NetFlow and the associated IPFIX standard [118, 119] and reveal elements of its operational model that make it unsuitable for L2 Traceback. Finally we discuss some of the deployment issues for COTraSE before concluding the Chapter.

5.5.1 Netflow and IPFIX

NetFlow is a general purpose traffic reporting system, widely used by ISPs to determine the composition of traffic on their networks [120]. In its simplest form, a flow consists of source/destination IP addresses, TCP ports and the IP protocol field. IPFIX defines a standard format for exported data and the protocol by which these are delivered for analysis.

The main difference between COTraSE and NetFlow/IPFIX lies in the expiration of flows. In NetFlow, flows are expired if: the end of a flow can be detected (i.e. TCP 'FIN' or 'RST'), when the flow has been inactive for a configurable timeout, or due to resource constraints (e.g. memory exhaustion). Longer lasting flows are also expired periodically (suggested 30 minutes) to avoid 'staleness'. In COTraSE however, all 'current' flows (or 'connections' as per our terminology) are expired at the end of the current 'active connections purge interval' as we have seen. We note that a similar approach is suggested in [120], where the authors suggest a much larger 'time-out' than we do (their focus was on improving NetFlow rather than L2 Traceback). We also note that NetFlow/IPFIX do not make provisions for correlating the flow data before and after potential transformations (e.g. NAT). We note the following of NetFlow/IPFIX flow expiration:

- *Flows are reported after the event.* Flows are reported when they expire, or after 30 minutes (configurable). Thus, the utility of this data by IDS or security monitoring systems in general is debatable. Applying this to logging for traceback it means one cannot bound the time that a frame was processed to any useful level of granularity.

- *Memory constraints often force a sampling approach.* The metering process maintains state for all active flows; with a large number of (longer lasting) flows, memory utilization can quickly increase beyond available levels. Sampling of traffic is suggested as a potential solution, and this is unacceptable for Traceback systems in general.

- *The exporting process is vulnerable to attack.* Flows are reported by the metering processes to the collection host as they expire. Given the high frequency of this operation, strong authentication and encryption of transferred data become expensive. A 'dedicated' link is suggested as a potential solution. Logging based L2 Traceback systems in general do not require that a constant stream of flow information traverse the network. Rather, logs are queried only in response to a traceback request (a less frequent event).

- *Flow expiration of non-TCP flows is inaccurate.* This is well known, and reported in [120]. Periodically expiring *all active flows*, as in COTraSE, improves accounting of non-TCP flows and removes the requirement to examine TCP flags *when* these are present.

5.5.2 COTraSEdeployment issues

Performance: Maintaining the connection identifier as a hashed digest preserves user privacy, even when logs are compromised. However it is also computationally expensive, and its necessity depends on the intended use of COTraSE. As an L2 traceback system the privacy preserving hashed *conId* is preferred. However, the EU 'data retention' council directive [16] mentioned earlier requires that 'data be transmitted upon request'. With a hashed conId, we cannot for example extract connection data for any given source or destination IP address. This point will be further discussed in the next chapter.

The second 'bottleneck' operation is the MAC-table lookup performed during switchport resolution. In *worst case* conditions a dedicated CAM might be a necessity in order to keep up with the packet load. This is a general problem for all logging based L2 traceback systems. Given that a switch already performs a MAC-table lookup for all frames, traceback functions would ideally be performed 'in switch' but such functionality is currently not made available by ethernet switch manufacturers.

Vulnerabilities: Buffering frames until the MAC-table update assumes that address mappings do not change between receipt of a frame and the update operation. An attacker may exploit this to spoof a tertiary host's address. The switch MAC-table would overwrite the mapping for the given address to identify the attacker's pNO as the origin. However, if the tertiary host transmits frames *after* the attacker but *before* the MAC-table update, then switchport resolution will mistakenly identify the tertiary host as the originator of the attack frame.

The attacker first must discover the tertiary host's MAC address which requires effort due to the low visibility of switched Ethernet. The attacker needs to predict when the tertiary host is transmitting, but also when the MAC-table update occurs. As was suggested in Chapter 4, the MAC-

table update time can be randomized (e.g., between 3 and 5 seconds). Furthermore, COTraSEuses the MAC-tables from both switches to perform switchport resolution making conditions even more adverse for an attacker.

Finally, an obvious attack on the COTraSE system is for an adversary to send a large number of frames designed to produce *different* conIds, to exhaust conRec log resources. We note that under these 'worst case' conditions our connection oriented approach will degenerate to no worse than an explicit frame log.

Partial Deployment: It may be prohibitively expensive to provide 'full' deployment with a conRec log between all network switches. However, we can adapt conRec log placement to the deployment network, though a partial deployment ultimately sacrifices 'local traffic visibility'.

A frame is processed by all conRecs encountered en route to the gateway router. However, only the conRec log adjacent to the frame's origin switch will be able to provide the 'axs_port' pNO which identifies the frame's origin switchport. All other conRec logs will 'point' to the origin within the network (i.e., by providing an ext_link pNO) which can at least aid in the identification of the instigating origin.

5.5.3 COTraSEand switch-SPIE

A quantitative comparison of the memory requirements of our switch-SPIE and COTraSEsystems is not possible due to the difference in the key logging algorithms. As we have seen, COTraSElogs only $\approx 15\%$ of all frames, whereas switch-SPIE is an explicit frame log (i.e., logging all frames). Switch-SPIE used bloom filters to lower memory requirements and this introduces false positives, which do not occur in COTraSE. Moreover, the memory requirements of switch-SPIE are governed by the bloom filter parameters but also by the number of switchports (and so will depend on the deployment switch size) as discussed in 4.5.2. However in COTraSEmemory requirements will depend on the characteristics of the network traffic (number of connections) as well as the purge interval as we have seen.

5.6 Chapter summary

We introduced COTraSEa 'connection oriented' logging based layer 2 traceback system for Switched Ethernet. COTraSEallows traceback of ethernet frames that are wholly local but also for IP packets addressed to external recipients, regardless of any address translation mechanisms (e.g., NAT). Rather than explicitly logging all traffic we attribute frames to ongoing connections and log representative connection records in discrete intervals. Our switchport resolution algorithm establishes the origin switch and port for frames by correlating MAC address entries from both adjacent switches.

This algorithm classifies the return of each Table lookup to detect potential errors such as MAC address 'spoofing'. COTraSEoffered improvement over our earlier switch-SPIE system in a number of areas, most importantly increased correctness of the traceback result and decreased memory requirements.

We empirically demonstrated the potential memory efficiency of a 'connection oriented' traceback approach by simulating the conRec log algorithm using data from available WAN traces. Our simulations show a consistently low $\frac{conRec}{totalframes}$ ratio of the order of 15% which is favourable in decreasing COTraSEmemory requirements. Finally Figure 5.12 shows the placement of sections in this chapter as an aid to the reader.

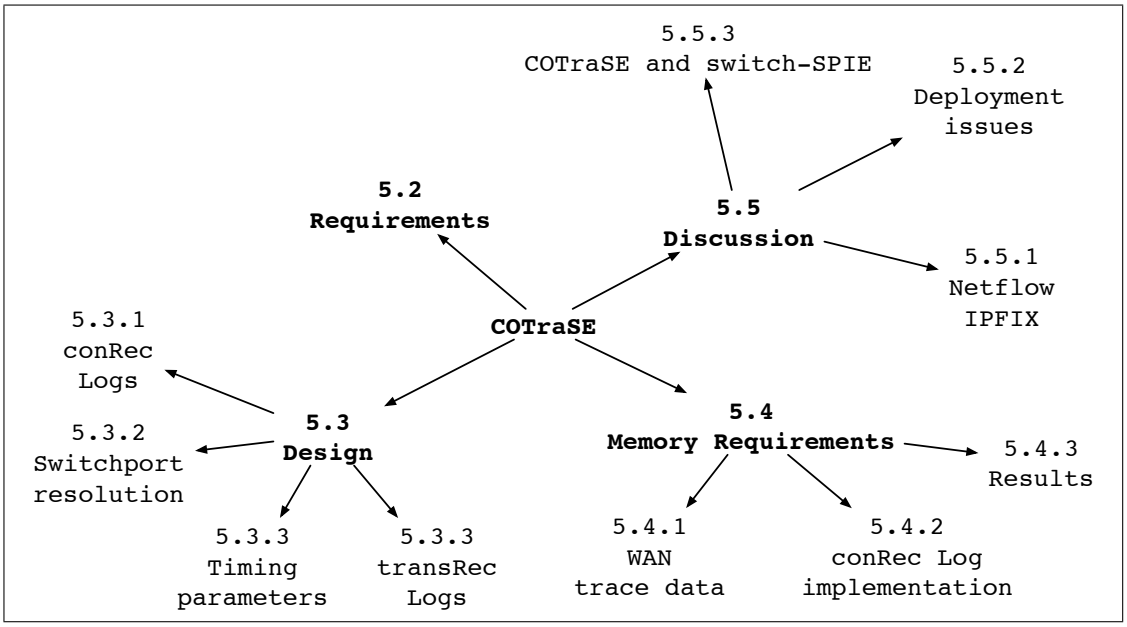


Figure 5.12: Idea map for Chapter 5 showing placement of sections

Chapter 6

EU 2006/24/EC

6.1 Introduction

This chapter examines EU council Directive 2006/24/EC "on the retention of data generated or processed in connection with the provision of publicly available electronic communications services...". As was mentioned in the introduction to Chapter 5, this directive requires that EU member states bring into force provisions at a national level for the logging of communications data, in a manner akin to the 'call records' maintained by mobile telephony service providers.

As we saw in Chapter 5, the connection oriented approach adopted by COTraSE was in part inspired by this EU data retention directive (EUDRD). For instance we consider the connection records produced by COTraSE to be well aligned with the requirements laid out by the EUDRD. More generally however, we regard all work in the areas of logging based IP and L2 traceback as being directly relevant in exposing the challenges to be faced by any implementation of this pan-European data retention system.

We begin this chapter with an executive summary of the EUDRD. This is then followed by a discussion of the main challenges that must be addressed, drawing from our own experiences in developing the switch-SPIE and COTraSE systems. We will also be using the terms developed in Chapter 2, including user or service data, sender or post-send spoofing (fraudulent or legitimate) and instigating or effective origins. We refer the reader to Figures 2.1, 2.2 and 2.3 for a reminder of these terms.

6.2 Executive Summary of Directive 2006/24/EC

The EU Directive 2006/24/EC (EUDRD) was passed on 15 March 2006 and its full title is "on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks". The core requirement is for providers of electronic communications services to retain data that identify the source, destina-

tion, time, duration and type of communications service used, as well as data to identify the user's communication equipment. After introducing some relevant definitions from related Directives, we decompose the EUDRD with brief summaries of why, what, when, how, and by whom data is to be retained. Finally consideration is given to two relevant UK Acts of parliament, the Data Protection Act 1998 [121] and the Regulation of Investigatory Powers Act 2000 [122], with a brief discussion of their relevance to the EUDRD.

6.2.1 Definitions

From Directive 2002/21/EC on a regulatory framework for electronic communications,

- **Electronic communications network**: switching or routing equipment and other resources which permit the conveyance of signals by wire, by radio, by optical or by other electromagnetic means, including fixed (circuit- and packet-switched, including Internet) and mobile terrestrial networks and irrespective of the type of information conveyed.
- **Electronic communications service**: a service normally provided for remuneration which consists of the conveyance of signals on electronic communications networks.
- **Public communications network**: an electronic communications network used wholly or mainly for the provision of publicly available electronic communications services.

From Directive 2002/58/EC on privacy and electronic communications,

- **Communication**: any information exchanged or conveyed between a finite number of parties by means of a publicly available electronic communications service.
- **Electronic mail**: any text, voice, sound or image message sent over a public communications network which can be stored in the network or in the recipient's terminal equipment until it is collected by the recipient.

6.2.2 Why is communications data to be retained?

The primary aim is to make data available for the prevention, investigation, detection and prosecution of criminal offences and in particular organised crime and terrorism. The July 2005 terrorist attacks in london are directly cited by the directive and reference is made to the "research and practical experience of several Member States" to stress the importance of location and traffic data to law enforcement authorities.

A secondary aim is to harmonise provisions for data retention across the European Union. The "legal and technical differences between national provisions" are seen to present an unfair advantage to communications service providers in Member States where data retention is not currently

mandated. Furthermore, by harmonising these provisions it is ensured that the primary aim of making communications data available across all Member States is satisfied and enables inter state cooperation on issues of law enforcement and national security.

6.2.3 When does data need to be retained and for how long?

Member states are required to bring into force "the laws, regulations and administrative provisions necessary" by no later than 15 September 2007. However there is an option to postpone application of the Directive for 18 months, until the 15th March 2009 (16 Member States have exercised this option by way of a declaration published with the EUDRD). At the time of writing and with March 2009 having passed, the UK home office has an open consultation [123] on the UK implementation of the EUDRD.

Regarding the period of retention, the EUDRD requires "not less than 6 months and not more than 2 years from the date of the communication". The exact retention period is to be resolved at a national level.

6.2.4 What data is retained?

The Directive is explicit in stating that the content of communications should not be stored. Rather, only that data which is "generated or processed" by service providers in the supply of their communications services is to be retained. This is potentially problematic as far as e-mail is concerned, as we will see in the next section.

The source and destination of a communication are identified by the user id (where applicable) and name and address of the subscriber to whom an IP address is allocated. The time and duration are identified by recording the log-in and log-off of an Internet access service (including Internet e-mail or Internet telephony), together with the IP address and whether this was dynamic or static and also the type of communications service used. Finally, it is required that a record be made of the users' communication equipment, such as the "digital subscriber line or other end point" and data to identify the location of mobile communication equipment.

6.2.5 How is the data to be retained?

The Directive states that it is not intended to harmonise the technology by which data will be retained and that this should be resolved at a national level. In practice this may contradict one of the core aims of the Directive, namely to eliminate "legal and technical" differences between national provisions. We will further discuss this point in our analysis below.

6.2.6 Who will retain data?

As stated in the EUDRD title, providers of publicly available electronic communications services, or of public communications networks will retain data. An obvious interpretation of this is that European governments will require all Internet Service Providers to log data. However it is not clear if the word "service" includes provision of web content services. Does this mean for instance that anyone who operates a web server be required to retain data regarding all server requests? Furthermore, apart from ISPs, are corporate and university networks required to retain data, or are these not considered "publicly available"? We will further develop this point in our discussion of implementation issues.

6.2.7 Related UK Acts of Parliament

This section considers two UK Acts of Parliament and their relevance to the EUDRD. The first is the Regulation of Investigatory Powers Act 2000 [122] and the second is the Data Protection Act 1998 [121].

The Regulation of Investigatory Powers Act (RIPA) consists of five parts and part 1 is most directly relevant to the EUDRD as this covers interception and acquisition of communications data. RIPA in general outlines the powers available to government agencies with respect to interception of communications, intrusive and covert surveillance and access to encrypted data. For each of these RIPA outlines the purposes for which a given power can be used and by which authorities as well as who can authorise the use of a power, how intercepted data can be used and also provides for independent judicial oversight.

RIPA part 1 describes the interception of communications data and sets out the specific conditions under which this can lawfully occur. There is no distinction between user or service data with interception defined as making "some or all of the contents of the communication available, while being transmitted, to a person other than the sender or intended recipient". Clearly this describes a much more intrusive form of data retention than that mandated by the EUDRD.

Interestingly RIPA defines 'traffic data' as "data identifying any person, apparatus or location to or from which the communication is or may be transmitted". Here this is taken to include the source and destination addresses within service data (e.g., TCP, IP, or MAC addresses). In defining 'interception', RIPA explicitly excludes the logging of 'traffic data'. This means that data retention as mandated by the EUDRD is not considered as 'interception' of communications and thus not subject to the same scrutiny as the powers afforded under RIPA.

RIPA also defines the terms "public telecommunication system" and "private telecommunication system" so that university or corporate networks are considered as private whilst an ISP network is considered as public. Whilst the EUDRD requires data retention in public communications systems,

RIPA provides for lawful interceptions in private networks under certain conditions. Given that UK government agencies already have the legal authority to instigate a very intrusive form of surveillance over electronic communications with RIPA, one might question the need for the implementation of a widespread data retention system (as required by the EUDRD).

The Data Protection Act (DPA) sets out the rights of an individual, with respect to the processing and disclosure of their personal data. The DPA was passed as a result of an earlier European Directive 1995/46 "on the protection of individuals with regard to the processing of personal data" [124]. The EUDRD explicitly states that directive 1995/46 applies in full to retained data and so equivalently it is expected that the DPA will apply in full to data retained under the UK implementation of the EUDRD.

The DPA provides definitions for 'personal data', that identifies or has the potential to identify (e.g., by combining with other available data) an individual, as well as 'sensitive personal data' that reveals information about an individual's racial origin, religious, political or sexual orientation, physical or mental health. Here it is argued that the data to be retained by the EUDRD, even if restricted to service data, constitutes personal and even sensitive personal data as defined by the DPA. For instance, it is possible to correlate an IP address with the address allocation data from the user's ISP to identify the person to whom that address was allocated. Furthermore, by looking at the destination IP addresses of a user's IP packets it is possible to reveal with whom that user has communicated, for instance the web server of a particular religious or political group.

Under Schedule 2 of the DPA, processing of personal data can occur if the subject of that data has given their consent. However, a seperate clause in Schedule 2 allows personal data processing if the data controller (e.g. the ISP) is required by law to do so; this would be the case for instance once the EUDRD has been implemented as a UK Act of Parliament. Furthermore, part 4 of the DPA lists the conditions under which the protections afforded by the DPA are voided. These include national security and crime which are also given as part of the motivation for the EUDRD. That is, the reasons cited by the EUDRD as necessitating the retention of data are the same reasons that allow exemptions from the Data Protection Act.

6.3 Implementation Issues

In this section we identify and discuss a number of key challenges and issues that must be addressed by any implementation of the EUDRD, drawing from our experiences with switch-SPIE and CO-TraSE. As mentioned in the Introduction we will use terms here that were developed in Chapter 2.

6.3.1 General Remarks

Though already controversial, we would argue that in some sense the EUDRD does not go far enough. If we are going to the trouble of putting in place a system that accounts for all our online communications then we should do it right. Yes, communications data is and must remain private. However there are proven ways of 'doing' logging whilst preserving user privacy. Firstly and as the EUDRD makes quite explicit, the user data must be designated as strictly off limits. Taking this into consideration, there is no difference between the communications records from online activity and those already maintained on our telephone calls. The retained data is just 'who called whom, for how long and what service where they using?'

However, if one then invests in the extra computing required to hash all communications data before logging this, then one ends up with a querying database. That is, the data in itself is not readable and so the only way to deduce whether a given communication was made or not is through specific queries. This is the approach taken by COTraSE where the hashed digest of {sourceMac, destMac, sourceIP, destIP, sourcePort, destPort} becomes the connection identifier by which all records are distinguised. In order to check if host A port 1 made a connection to host B port 2 on 21June 2008 around 1200GMT one would need to compute the connection identifier and use this to search archived logs.

Thus, when records are hashed one needs a starting point in order to query the system. For instance an Intrusion Detection System raises an alert based on some predefined rules and you use the alert or any generated logs to issue a query to the L2 traceback system. We will see that storing logs in cleartext allows for much more general querying which is certainly more intrusive.

Another issue is who will be required to log? It is likely that networks maintained by universities or corporations will not be considered as "publicly available" and so will not be required to log data. However, we will see that this will likely create a security problem at these institutions. Firstly, the lack of logging will make these networks a target for compromise (e.g. in order to launch network attacks). Furthermore it might be the case that these networks are forced to deploy logging by proxy, as they will wish to establish personal liability for some malevolent network act that has been proven to originate from their network.

It has been speculated in UK newspapers that a single database would be maintained for all data retained under the UK implementation of EUDRD [125]. Knowing the storage needed for logging from our L2 traceback systems this is an absurd notion, though one can appreciate the allure of such an approach. The most obvious advantage from a government's point of view is the easy access to data from a source exclusivedly controlled by a relevant government agency. This also works for the ISPs as they do not have to take on the cost of securely managing their own databases of retained data, whilst also servicing any requests from the presumably numerous government bodies that will be granted access. With the central database approach the ISPs can simply 'log and ship' without

being concerned with storage. Of course the complexities and cost of transferring that amount of data to a single location makes this approach extremely difficult (though not strictly impossible).

6.3.2 Accuracy and reliability of retained data

Considering the intended use of network data under the EUDRD, it is imperative that the logging procedures be entirely reliable. It must not be possible for the logging and any subsequent querying systems to produce a false positive. That is, it should not be possible to incorrectly incriminate an innocent party. Consider that communications data retained under the EUDRD are intended to be produced as evidence before a court of law. As we have seen in Chapter 2 there are many ways in which both user and service data can be altered, whether fraudulently or legitimately and at the instigating origin or at some intermediary host en route to the intended recipient. Logging procedures must be sure that the identifiers attached to a given communication record (e.g. source IP or MAC address) are accurate.

In switch-SPIE and even more so in COTraSE, we adopted the somewhat extreme approach of treating all packets as 'guilty ́till proven innocent'. That is, it is assumed that the source MAC address is forged and so we employ a process of 'switchport resolution' to assign a source switch identifier and switchport number to a given connection record. This approach however may only be viable in very sensitive networks where the need for fine grained accountability over network data justifies the added procedures and their associated costs.

Thus there must be some middle ground for any general and wide spread implementation. In a switched ethernet network (wherein our experience lies), switches provide a means with which to 'lock down' an access switchport to a given source mac address. That is, the switch is instructed to drop any frames that do not carry a pre-configured MAC address as source. Of course this introduces an additional cost to the overall network administration (e.g., some addresses may be configured manually). However with such mechanisms in place, a source MAC address can be taken as valid and not spoofed, making switchport resolution redundant. Measures such as this should become a part of the local security policy where available. Though our discussion here is specific to a switched ethernet, in general the issue of spoofing and its possibility within the context of a given deployment network must be considered and factored into any design of a data retention system.

6.3.3 Ciphertext versus plaintext logs

Regardless of the technology by which communication records are produced, one must decide whether retained data will be stored as cleartext or as ciphertext. For instance in COTraSEan MD5 hash digest is used to create the connection identifiers by which unique communications are distinguished. This has an obvious impact on the privacy of logged data and inevitably leads to a more secure

management; communication records stored as ciphertext are meaningless even if they are entirely compromised. Of course, producing a hashed digest for all communication records is computationally expensive and so the obvious trade-off is between performance and privacy. It could even be argued that since the EUDRD requires only service data to be retained (and not the more private and sensitive user data) the use of cryptography is 'overkill'.

However, there is a second consequence (and trade-off) of the choice between storing communications data as cleartext or ciphertext which is more subtle but equally, if not more important than that of performance. With ciphertext records, the way in which retained data can be queried becomes more restrictive. You need to 'know' what you are looking for and this was demonstrated by both COTraSE and switch-SPIE. In switch-SPIE, the logging procedures explicitly log all packets into a bloom filter and so only specific queries can be serviced ("have you seen this packet?"). In COTraSE and due to the connection oriented approach and lack of bloom filters, more general queries can be serviced, such as "was there a communication between host A port X and host B port Y on 02 June 2009 at around 19:21 GMT?". However in the case of cleartext records, data mining techniques can be employed to answer much more general queries such as "give me all communication records with X as source IP last week".

Thus, the trade-off is between privacy, performance and importantly traceback 'convenience'. This can be related to the notion of evidence: in order to query a traceback system such as switch-SPIE or COTraSE, authorities will need some initial information, or 'evidence', to serve as a starting point. By storing ciphertext logs one safeguards privacy by way of the lower traceback convenience, albeit at the cost of performance (i.e., logging nodes would need fast and multiple processors and network cards and large storage capacities).

6.3.4 Who and where?

As stated in the executive summary above, the EUDRD is not explicit about who is considered a "provider of public communications services". If one asks the highest level ISPs to log data, then it is not possible to identify the instigating origin host. As we have seen the instigating origin's identity can only be reliably resolved within that given host's local network. This is demonstrated by the fact that IP traceback procedures, deployed in routers, can only identify the instigating origin's source network. Layer 2 traceback systems, such as switch-SPIE and COTraSE can then identify the instigating origin itself, though the granularity will depend on the deployment technology (e.g., switched ethernet or 802.11 wireless); this point is further discussed subsequently. In general the proximity of a logging node to the instigating origin host directly affects one's ability to determine that host's identity, assuming that sender or post-send spoofing is employed. Thus, if only the largest ISPs are asked to log data, there will still be a great number of hosts for whom communications data will only be partially identifiable.

One obvious example that we have seen is Network Address Translation and more generally legitimate post-send spoofing. For instance, a public library is provided with Internet connectivity services by a large ISP, and is assigned a single publicly routable IP address. The library's network administrator therefore configures DHCP and NAT so that appropriate local addresses are assigned and, correctly altered before leaving the LAN. One of the publicly available hosts at the library is used to post messages on an extremist blog about a terrorist attack against U.K. interests. The ISP logs this activity but all packets coming out of the library will carry the same source IP address. Without logging deployed within the origin network (as in L2 traceback) there is no way to know 'whodunit'. This is known as 'the cheerleader defense' as we saw in Chapter 3 when discussing the case of Tammie Marson vs Virgin Records America Inc..

Related to this is whether universities and corporate networks will be required to retain data. One expects that they will not, given the wording of the EUDRD that data be retained only in the provision of a "public communications network". Will this then make these networks, and in particular academic networks an obvious launchpad from which to execute network attacks? It is not inconceivable that student log-in accounts become prized by malicious network users and thus a target for compromise. Moreover, with a widespread, pan-European data retention system deployed at ISPs it will become possible to identify that a given communication was made from within a specific network (e.g., Newcastle university central campus network). Thus even if it is not mandated through legislation, any EUDRD implementation will eventually force smaller network providers to deploy Layer 2 traceback logging procedures. If nothing else this will enable them to remove the liability for prosecution from themselves.

6.3.5 E-mail and "the content of electronic communications"

The EUDRD, as we saw above is explicit in that data retention does not apply to the content of electronic communications but rather only to data "generated or processed" by the service providers. This fits well with our notions of user and service data; data from the Data-link, Network and Transport layers can be logged, but anything from the Application layer (user data) is off limits.

However, the directive is also explicit in requiring that all categories of data (source, destination, time, duration, type, user equipment) be retained for e-mail. This poses a problem as e-mail is by definition user data and not service data. An e-mail, including the source and destination addresses, subject and message body, is carried as the payload of 'regular' IP packets using TCP at the Transport layer. If one were to intercept this data en route to the mail server, and without looking at the user data it is impossible to deduce the `from` and `to` fields of the e-mail message. In fact the only clue that the given data belongs to an e-mail might be the destination TCP port, for instance port 25 for an SMTP server or port 110 for a POP3 server.

There is some wording in the Directive that can be interpreted as addressing this issue and that

is that "data generated or processed" should be logged. In general service data is processed by service providers and it is this data that would be retained (e.g., the source IP address and TCP port number). However, in the case of e-mail one would normally expect some record of each SMTP exchange to be kept at a given mail server. The mail server logs are generated by the service provider in the process of delivering the email service. However this is still user data albeit received by proxy. Whether the user gives up their right to privacy as they have handed over their data to the service provider is, one can be assured, a controversial issue.

6.3.6 Heterogeneous networks

The EUDRD does not address the technology for retaining data. It is understandably difficult to specify the way in which data retention should be technically deployed. This of course will depend both on network topology and physical transmission medium used. Our L2 traceback systems are designed for a switched Ethernet. As we saw this poses the problem of low traffic visibility, addressed by switchport mirroring in switch-SPIE and by a tap between link ports in COTraSE. However, switched ethernet also allows us to be very precise about the instigating origin's identity by providing the switch identifier and switchport number (as always, assuming spoofing is employed).

If on the other hand IEEE 802.11 wireless is used as the transmission medium then there is no problem with traffic visibility but the instigating origin can not be reliably identified so precisely. If sender spoofing of the MAC address is assumed then one can only reliably identify the wireless access point and not the instigating origin host itself.

6.4 Chapter Summary

This concludes our examination of Directive 2006/24/EC on the retention of data. We first provided an executive summary of the EU data retention directive (EUDRD) to aid the reader by answering the questions: why, when, what, how and by whom is data to be retained? We then used the terms developed in Chapter 2 to expose the technical challenges that will be faced by any implementation of the EUDRD. These are mainly the reliability and accuracy of logging procedures as well as the extent to which these respect end user privacy. We saw that the case of e-mail is particularly problematic with respect to logging without retaining user data. Finally Figure 6.1 shows the placement of sections within this chapter as an aid for the reader. We now turn to the final chapter and the conclusions of this work.

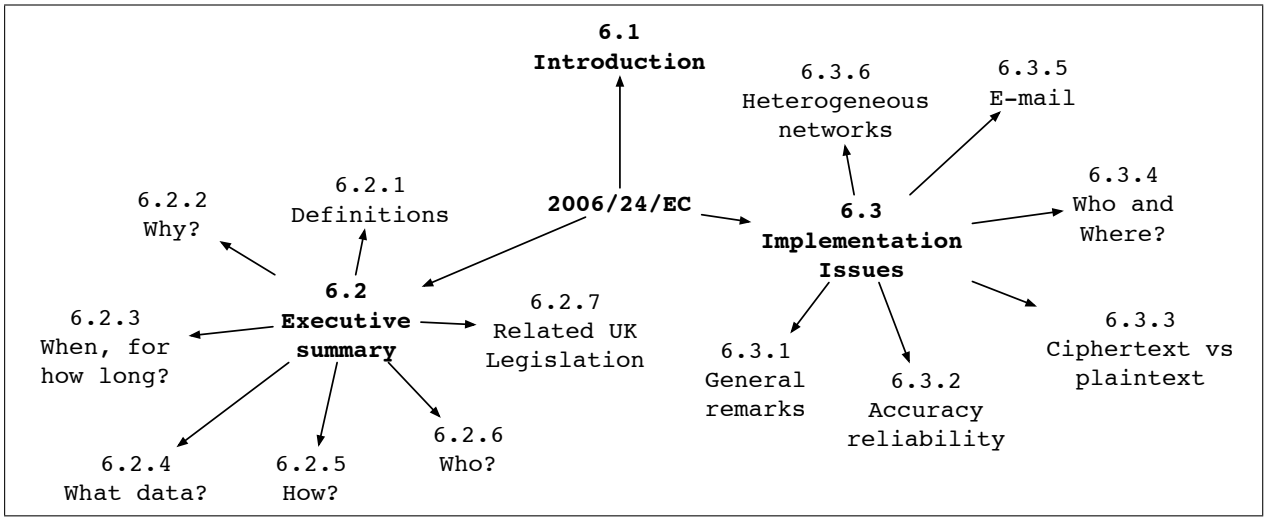


Figure 6.1: Idea map for Chapter 6 showing placement of sections

Chapter 7

Conclusions

This Chapter concludes this work. We first summarise the thesis before discussing its overall contributions. This is followed by a consideration of future research directions and finally our concluding remarks.

7.1 Thesis Summary

Chapter 1 introduced this thesis, outlining the main goals and our approach in satisfying these. We introduced the notions of spoofing and message traceback systems and their relevance to the European data retention directive 2006/24/EC. We also summarised the main contributions and outline the thesis contents.

In Chapters 2 and 3 we unified the relevant threads of research, namely 'IP and L2 traceback' but also 'connection-chain traceback' and 'host fingerprinting'. In Chapter 2 we began this process by providing a unifying definition of 'spoofing' together with a classification of its various forms. We saw that spoofing is the alteration of any network data intended as an identifier (ephemeral or otherwise), that effectively obscures the identity of the instigating origin (sending) host.

Chapter 2 designated spoofing at the instigating origin as 'sender spoofing' and 'post-send spoofing' as alterations made at some intermediate network host (which becomes the effective origin of that given data), depicted in Figure 2.1. We further differentiate between spoofing of user data and service data and our classification is summarised by Figure 2.3. A final distinction was made between fraudulent spoofing, when data is altered intentionally and with intent to defraud and legitimate spoofing, that is brought about by sanctioned network processes such as Network Address Translation. As we saw, sender spoofing is always fraudulent, in so far as we are unable to envisage a scenario in which this could be employed legitimately (beyond experimentation by network administrators for instance). However, post send spoofing can be both legitimate and fraudulent. We then provided an extended discussion of each type of spoofing with examples. Figure 2.4 shows the limitations of sender spoofing when TCP is in play. Figures 2.6, 2.7 and 2.8 show cases of post-send,

user data spoofing and Figure 2.9 depicts an example of legitimate post-send spoofing of service data (NAT).

Whilst Chapter 2 unified the various forms of spoofing, Chapter 3 provided an extensive literature review. We considered proposals that in some way address the issue of tracing back to the origin of network data, in spite of any sender or post-send spoofing. We first saw that IP traceback systems aim to identify the origin of network data in spite of sender spoofing. This is accomplished in the majority of proposals by either marking or logging of IP packets at network routers, as summarised by Figure 3.2. We saw that whilst logging procedures capture the identity of each packet, marking procedures capture the identity of each router. We also examined some hybrid approaches (both marking and logging) as well as related attack detection and mitigation proposals (especially against Denial of Service).

Figure 3.9 shows that IP traceback can only reveal the origin's first hop router and so Layer 2 traceback is required to identify the instigating origin itself. We briefly summarised all known proposals for L2 traceback and the main challenges that these identified. Most work in the area of L2 traceback assumes a switched ethernet network, though we also considered two proposals for traceback in a mobile ad-hoc network. Finally, Chapter 3 considered proposals for connection chain traceback that address the issue of post-send spoofing. Figure 3.11 depicts the example of consecutive ssh connections through compromised hosts with a DoS attack launched from the last host in the chain (the effective origin). We examined a number of connection chain traceback proposals which fall mainly into the categories of host and network based.

In Chapter 4 we presented the first of our L2 traceback systems, switch-SPIE. As we saw this is an adaptation of the open source SPIE IP traceback system for a switched ethernet. For any given IP packet switch-SPIE aims to identify the instigating origin host, through the unique pairing of switch identifier and switch-port number. We first gave an overview of SPIE (briefly encountered in Chapter 3) before considering those characteristics of switched ethernet that bear upon our ability to traceback to the instigating origin, in spite of sender spoofing. This was followed by details of our switch-SPIE design with Figure 4.3 providing the key algorithms for logging and traceback request processing. Our proposal improves over an earlier adaptation of SPIE for switched ethernet [9] by deploying logging at each switch, rather than at a single network location. As we saw switch-SPIE can traceback frames that are entirely local (i.e., do not traverse the gateway router) and our configuration allows the MAC-tables maintained by each logging node to be updated with a higher frequency than in [9].

Figure 4.4 shows the switch-SPIE deployment network and Figure 4.5 summarises the key changes made to the SPIE code for our implementation. Figures 4.7 and 4.10 show the configuration of the switch during our investigation into the reliability of the switch-port mirroring mechanism. As we saw switch-SPIE relies on mirroring to overcome the low traffic visibility in switched Ethernet.

Our experiments showed that whilst the switch can 'keep up' with switch-port mirroring without adverse effect to the primary switching function, the network host and switch-port to which traffic is mirrored are quickly overwhelmed. Furthermore and as we saw in the relevant discussion, memory requirements for switch-SPIE can quickly spiral in larger and faster networks (e.g., faster than gigabit ethernet), even though the traceback window is restricted to only 1 minute. Table 4.2 shows the memory requirements for 100Mbit and 1000Mbit ethernet with different bloom filter sizes and the corresponding false positive rates. Finally we considered the MAC-table race condition, as depicted in Figure 4.11 and Figure 4.12 shows switch-SPIE in a partial deployment.

Chapter 5 presented our second attempt at L2 traceback for a switched ethernet, where we built on our experience with switch-SPIE. Our COTraSE system adopts a connection oriented approach to logging as we saw, which was in part inspired by EU Directive 2006/24/EC. COTraSE decreases memory requirements by only logging a subset of all packets. A conceptual overview of 'connection oriented' logging is given in Figure 5.2 whilst Figures 5.5 and 5.4 give UML activity diagrams for the key algorithms of COTraSE.

Crucially and as was explained, whilst we log only a subset of packets as connection records, this does not sacrifice our ability to traceback any given ethernet frame or packet and provide the unique {sID, pNO} pairing to identify the instigating origin. In COTraSE we deployed logging between network switches rather than on a monitor port as in switch-SPIE, as is shown in Figure 5.1. This allowed us to improve the switchport resolution algorithm so that the origin switchport is resolved more accurately albeit at the cost of an extra MAC-table lookup per frame. This approach of maintaining 2 MAC-tables (one from each adjacent switch) allows us to detect potential sender spoofing activity. We also saw that COTraSE allows for legitimate post-send spoofing mechanisms such as NAT by maintaining translation records at the gateway router and Figure 5.7 depicts the transRec logging algorithm. Finally Chapter 5 provided details of our implementation of the connection record logging algorithm which used 'real' WAN trace data as input. By processing over 300 million packets we were able to empirically deduce that for the given data the number of connection records required per minute is approximately 15% of the total number of frames, equating to much reduced memory requirements, compared to explicitly logging all frames.

Finally in Chapter 6 (this Chapter) we presented EU Directive 2006/24/EC and identified the relevance of work in the areas of IP and L2 traceback to the implementation of any 'data retention' system. We used our own experiences in developing switch-SPIE and COTraSE, as well as the definitions introduced in Chapter 2, to discuss some of the issues that such an implementation must at least consider.

This completes our thesis summary and we now turn to a discussion of the key contributions made by this work.

7.2 Contributions

The contributions of this thesis go beyond the provision of two Layer 2 traceback systems for switched ethernet. We have seen how a unifying definition and simple classification of spoofing can bring together relevant research in the areas of IP, L2 and connection chain traceback. Furthermore, we show that we can directly apply the experience in developing these message traceback systems to the implementation of EU directive 2006/24EC. The main contributions of this thesis are:

- A unifying definition and classification of spoofing,
- An extensive literature review of all known work in the areas of IP, L2 and connection-chain traceback using our own classification from above,
- The switch-SPIE L2 traceback system, adapted from SPIE, presented as 'Logging Based IP traceback in Switched Ethernet' at Eurosec 2008 [15],
- The improved L2 traceback system, COTraSE, inspired by EU Directive 2006/24/EC, presented as 'COTraSE: Connection Oriented Traceback in Switched Ethernet' at IAS 2008 [13] and later published in the Journal of Information Assurance and Security [12],
- Identification of the relevance of message traceback systems to the EU data retention directive,
- Identification of key challenges to be faced by the EUDRD implementations, using our experience with L2 traceback.

Work in the areas of IP and connection chain traceback have traditionally been considered as distinct. In this thesis we argue that IP traceback (and its L2 subdomain) and connection-chain traceback are both fundamentally addressing the same issue of message traceback: How can we traceback a given network message (e.g., an ethernet frame) to its origin host if we assume that the information contained in that message (especially source addresses) has been altered? Our definition of spoofing was created with the primary aim of a classification applicable to all related domains. We showed that IP and L2 traceback address the issue of sender spoofing, whilst connection-chain traceback addresses post-send spoofing. We presented the related work using our classification and definition in order to demonstrate their viability.

The switch-SPIE system was our first attempt at L2 traceback for a switched ethernet, which explicitly logs all packets using bloom filters. We adapted the open source SPIE IP traceback system and assigned an exclusive bloom filter for each switchport. We added the step of switchport resolution to the 'normal' SPIE logging routines, in order to determine the appropriate bloom filter for each received frame. We aimed to improve an earlier adaptation of SPIE for switched ethernet by

deploying logging locally at each switch, rather than at the gateway router. The implications of this are that the traceback result is more reliable as local MAC-tables can be refreshed from the switch more frequently. Furthermore, this topology allows one to traceback frames that are entirely local and do not traverse the gateway router. The problem of low traffic visibility in switched Ethernet is well known and switch-SPIE took the approach of using switchport mirroring. Our experiments showed that this is not a reliable means of receiving network traffic. This is especially true for larger switches where the aggregate traffic from a number of switchports will quickly overwhelm the monitor port. Our discussion of memory requirements showed the limitation of our 'one bloom filter per switchport' approach; memory requirements are excessive in the worst case of a very large switch (1000+ switchports), especially when maintaining a low false positive rate (equating to a larger bloom filter).

The COTraSE system was primarily intended to address the challenges revealed by switch-SPIE. This is evident for instance in the placement of connection record logs at a tap between switches, rather than relying on switchport mirroring. However, at about this time the EU data retention directive came to our attention and we instantly recognised the applicability of logging based IP and L2 traceback systems to this legislation. The connection oriented logging approach taken by COTraSE was in part inspired by the EUDRD as we have seen. In COTraSE we further exploit the switch MAC-tables to make the switchport resolution algorithm even more reliable. We saw that at the cost of an extra MAC-table lookup and some prior configuration at each switch to classify switchports (e.g., access or internal link ports) switchport resolution can even detect malicious activity such as sender spoofing of source MAC address. Furthermore, by maintaining translation records at the gateway router, COTraSE can traceback IP packets that are delivered to external hosts, regardless of any legitimate post-send spoofing such as NAT. The potential memory efficiency of the connection oriented approach was demonstrated using WAN traces from large ISPs that were taken as input by our connection record log implementation. We saw that even in the worst case of a 10 second purge interval, the number of connection records as a percentage of total frames was consistently lower than 15% across all traces (over $300,000,000$ packets).

Finally we introduced the EU data retention directive with an executive summary, before listing the key challenges that any implementation will have to consider. Our discussion drew on our experiences with switch-SPIE and COTraSE and also made use of the definitions that were developed in the earlier Chapters.

We now turn to a discussion of future work and research directions, before our concluding remarks signal the end of this thesis.

7.3 Future Work

The main future research directions are:

1. Implement logging and switchport resolution 'in switch',
2. Develop a full COTraSE deployment,
3. Investigate a better translation record logging mechanism,
4. Survey commonly used Data-link layer technologies and their traceback characteristics,
5. Research legislative and technical differences between EUDRD implementations,
6. Develop a standard querying language and protocol for traceback request processing.

7.3.1 Implement logging and switchport resolution 'in switch'

Our implementation of switch-SPIE used the 'dgad' configuration file to simulate the MAC-table. As was discussed, the console port available on all switches for local configuration could be used for querying the switch MAC-table. This is made possible by the deployment of switch-SPIE at each switch, rather than at a central network location. Furthermore and as was suggested, if this port is unavailable then the Simple Network Management Protocol can be used to query the switch. It is of interest to empirically deduce the feasibility of either approach. For instance, how much faster is the console port compared to SNMP if at all? What is the delay between a MAC address being changed in the switch MAC-table and this being updated in the switch-DGA MAC-table?

In switch-SPIE we are essentially trying to implement logging 'in switch'. That is, each switch-DGA logging host is receiving traffic explicitly forwarded to it (through port mirroring) by its local switch. Thus, the ideal location for switchport resolution and logging more generally would be from within the switch operating system itself. As was discussed, a switch already performs a MAC-table lookup for all frame *destination* addresses to determine the appropriate egress port. The switchport resolution algorithm of switch-SPIE consists of a MAC-table lookup based on the *source* MAC address. Thus, a second avenue of research is whether logging and switchport resolution can be implemented by adapting an open source switch operating system. Such an approach would alleviate the reliance of switch-SPIE on the switchport mirroring mechanism for network traffic.

In COTraSE, the issue of implementing logging 'in switch' is not so clearly advantageous. Recall that COTraSE uniquely used the MAC-tables from both adjacent switches for the switchport resolution algorithm. If one were to implement logging 'in switch' then only the MAC-table of the local switch would be available.

7.3.2 Develop a full deployment of COTraSE

You may recall that our implementation of COTraSE as discussed in Chapter 5 was designed with memory requirements in mind. A full deployment of COTraSE would allow us to further establish its feasibility as a traceback system for sensitive network environments. One research direction is to establish the reliability of SNMP for the MAC-table updates from either adjacent switch (the console port approach is ruled out here). Another is to establish the feasibility of a cheap ethernet tap implementation consisting of an ethernet hub; for instance, what is the performance penalty due to the addition of a collision domain between the two switches?

7.3.3 Investigate a better translation record logging mechanism

Recall that in COTraSE, translation records are a mapping between connection identifiers before and after any of the address fields are re-written (e.g., due to NAT). In COTraSEour approach is simple in that a direct mapping is maintained between data before and after modification. We are confident that a more elegant solution is possible and further investigation is warranted.

7.3.4 Survey commonly used Data-link layer technologies

Both switch-SPIE and COTraSE are designed with switched ethernet as a deployment network. We saw that this is interesting because it allows us to perform switchport resolution and so address sender spoofing. Furthermore, this allows us to identify a specific network access point via {sID, pNO}. However, in other physical transmission media one can expect different traceback results. For instance in 802.11 wifi and as we saw in Chapter 3 at best we can identify an access point (the first encountered malicious node in a MANet). Another example is DOCSIS ('cable' Internet) where a common cable network is shared between subscribers in an area (e.g. a residential street). It would be of interest to compile a list of commonly used transmission media and what their characteristics are with respect to traceback. A primary concern is the availability of network traffic (e.g., traditional vs switched ethernet) and whether any maintained state can allow a switchport resolution like process.

7.3.5 Research legislative and technical differences between EUDRD implementations

One can already expect slight variations in the national provisions and technical implementations of the EUDRD. For instance the directive allows a retention period of between 6 months and 2 years, to be resolved at a national level. Denmark has already implemented the EUDRD [126] and as we saw the UK home office currently has an open consultation on the UK implementation. Beyond periods of retention, of particular interest is who will be required to retain data. Furthermore and

as discussed the technical differences between implementations may vary considerably. For instance what processes make the logging accurate (e.g., through security policies) and reliable (i.e., false positives are unacceptable).

7.3.6 Develop a standard querying language and protocol for traceback request processing

What are the queries that the EUDRD will be required to service? In order for this to be standardised there of course needs to be agreement across the European Union. After all one of the primary aims of the directive is to harmonise data retention provisions in order to ensure cooperation and availability of data for law enforcement. Any standard querying language and protocol would need to be preceded by legislative provisions, which are not currently in place as member states are still in the process of implementing directive 2006/24/EC.

7.4 Concluding Remarks

The European data retention legislation 2006/24/EC will impact all citizens of the Union in a very personal way. It is thus imperative that any implementations are 'right' and especially mindful of a person's right to respect for his private life and his correspondence, under Article 8 of the European Convention for the Protection of Human Rights and Fundamental Freedoms. We have remained neutral by avoiding any discussion of the rights and wrongs of data retention in general. We merely underline the importance of all work in the areas of message traceback (IP, L2 or connection-chain) to directive 2006/24/EC. Our L2 traceback work allows us to expose the subtleties of reliably and accurately logging data. We would like to emphasize the importance of the various trade-offs that should be considered in any designs, especially with regards to encryption of logged data.

It is possible to do logging right but this will only occur if we go beyond merely expressing concern. We must be active in averting the negative implications to our privacy that can easily be brought about, if nothing else through lack of understanding of the technical challenges. We must be compelled to get involved and make it right. One part of this is the promotion of much further work in all areas of message traceback. We have no choice but to dance with the devil.

Bibliography

[1] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. In *ACM SIGCOMM*, 2000.

[2] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *ACM SIGCOMM Computer Communication Review*, 30(4):271–282, 2000.

[3] A. C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F Tchakountio, B. Schwartz, S.T. Kent, and W.T. Strayer. Single packet ip traceback. *IEEE/ACM Transactions on Networking*, 10(6):721–734, 2002. preliminary version presented at ACM SIGCOMM 2001.

[4] Hal Burch and Bill Cheswick. Tracing anonymous packets to their approximate source. In *14th USENIX Conference on System Adminimstration (LISA)*, 2000.

[5] H. Hazeyama, M. Oe, and Y. Kadobayashi. A layer-2 extension to hash based ip traceback. *IEICE Transactions on Information and Systems*, E86(11):2325, 2003.

[6] Ahmad Almulhem and Issa Traore. A survery of connection-chains detection techniques. In *IEEE PacRim Conference on Communications, Computers and Signal Processing*, pages 219–222, 2007.

[7] Hyun Tae Jung, Hae Lyon Kim, Yang Min Seo, Ghun Choe, Sang Lyul Min, and Chong Sang Kim. Caller identification system in the internet environment. In *IV USENIX Security Symposium*, 1993.

[8] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher. *Internet Denial of Service (Attack and Defense Mechanisms)*. Prentice Hall, 2005.

[9] M. Snow and J. Park. Link-layer traceback in switched ethernet networks. In *IEEE LANMAN*, pages 182–187, 2007.

[10] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for ip traceback. *IEEE/ACM Transactions on Networking*, 9(3):226–237, 2001.

[11] A. Belenky and N. Ansari. Tracing multiple attackers with deterministic packet marking (dpm). In *IEEE PACRIM Conference on Communications, Computers and Signal Processing*, pages 49–52, 2003. Vol. 1.

[12] Marios S. Andreou and Aad van Moorsel. Cotrase: Connection oriented traceback in switched ethernet. *Journal of Information Assurance and Security*, 4(2):91–105, 2009.

[13] Marios S. Andreou and Aad van Moorsel. Cotrase: Connection oriented traceback in switched ethernet. In *The Fourth International Conference on Information Assurance and Security*, 2008.

[14] T. Kohno, A. Broido, and K.C. Claffy. Remote physical device fingerprinting. *Dependable and Secure Computing, IEEE Transactions on*, 2(2):93–108, April-June 2005.

[15] Marios S. Andreou and Aad van Moorsel. Logging based ip traceback in switched ethernets. In *EUROSEC '08: Proceedings of the 1st European workshop on system security*, pages 1–7. ACM, March 2008.

[16] Council European Parliament. On the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks. *Official Journal of the European Union*, L 105:54–63, April 2006. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32006L0024:EN:NOT`.

[17] BBN Technologies. Source path isolation engine, 2004. `http://www.ir.bbn.com/projects/SPIE/spiehome.html`.

[18] CISCO. Catalyst switched port analyzer (span) configuration example. `http://www.cisco.com/en/US/products/hw/switches/ps708/products_tech_note09186a008015c612.shtml`, 2007.

[19] R. Waterman, B. Lahaye, D. Romanascanu, and S. Waldbusser. Remote network monitoring mib extensions for switched networks - ietf rfc2613, standards track. `http://www.ietf.org/rfc/rfc2613.txt?number=2613`, 1999.

[20] B. Claise. Cisco systems netflow services export version 9 - ietf rfc 3954, informational. `http://www.ietf.org/rfc/rfc3954.txt?number=3954`, 2004.

[21] Houghton Mifflin Company. The american heritage dictionary of the english language, 4th edition. online - retrieved from Dictionary.com website. `http://dictionary.reference.com/browse/spoof`.

[22] K. Egevang and P. Francis. The ip network address translator (nat) - ietf rfc 1631, informational. `http://www.ietf.org/rfc/rfc1631.txt?number=1631`, 1994.

[23] P. Srisuresh and K. Egevang. Traditional ip network address translator (traditional nat) - ietf rfc 3022, informational. `http://www.ietf.org/rfc/rfc3022.txt?number=3022`, 2001.

[24] Stuart Staniford-Chen and L. Todd Heberlein. Holding intruders accountable on the internet. In *IEEE Symposium on Security and Privacy*, pages 39–49, 1995.

[25] Andrew S. Tanenbaum. *Computer Networks, 4th Edition*. Prentice Hall, 2003.

[26] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol http/1.1 - ietf rfc 2616, informational. `http://www.ietf.org/rfc/rfc2616.txt?number=2616`, 1999.

[27] Chris Pederick. User agent switcher :: Firefox add-ons, March 2008. `https://addons.mozilla.org/en-US/firefox/addon/59`.

[28] Sean Michael Kerner. Spoofing googlebot. *internetnews.com*, July 17th, 2007. `http://www.internetnews.com/security/article.php/3689241`.

[29] DARPA. Darpa internet program transmission control protocol - ietf rfc 793 - editor: Jon postel. `http://www.ietf.org/rfc/rfc0793.txt?number=793`, 1981.

[30] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[31] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allolaction for private internets - ietf rfc 1918, best current practice. `http://www.ietf.org/rfc/rfc1918.txt?number=1918`, 1996.

[32] S. Bellovin. Defending against sequence number attacks - ietf rfc 1948, informational. `http://www.ietf.org/rfc/rfc1948.txt?number=1948`, 1996.

[33] Chris Gulker. The kevin mitnick/tsutomu shimomura affair. `http://www.gulker.com/ra/hack/`, 2001.

[34] Salvatore Sanfilipo. New tcp scan method. `http://seclists.org/bugtraq/1998/Dec/0079.html`, 1998.

[35] (unknown) 'm3lt' meltman@lagged.net. The land attack (ip dos). `http://insecure.org/sploits/land.ip.DOS.html`, 1997.

[36] CERT Coordination Center. Advisory ca199728: Ip denial-of-service attacks. online, 1997.

[37] Jani Hursti Lasse Huovinen. Denial of service attacks: Teardrop and land. Technical report, Department of Computer Science, Helsinki University of Technology, 1998.

[38] CISCO. Cisco security advisory: Tcp loopback dos attack (land.c) and cisco devices. `http://www.cisco.com/warp/public/707/cisco-sa-19971121-land.shtml`, 1997.

[39] Malachi Kenney. Ping of death. online - insecure.org `http://insecure.org/sploits/ping-o-death.html`, 1996.

[40] CERT Coordination Center. Advisory ca199621: Tcp syn flooding and ip spoofing attacks. online, 1996.

[41] (unknown) 'T. Freak' tf@tap.net. 'smurf' multi-broadcast icmp attack. `http://seclists.org/bugtraq/1997/Oct/0066.html`, 1997.

[42] Hain T. Architectural implementations of nat - ietf rfc 2993, informational. `http://www.ietf.org/rfc/rfc2993.txt?number=2993`, 2000.

[43] Daniel J. Barrett, Richard E. Silverman, and Robert G. Byrnes. *SSH The Secure Shell The Definitive Guide*. O'Reilly Media, 2nd edition, 2005.

[44] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 2(24):84–90, 1981.

[45] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *13th USENIX Security Symposium*, 2004.

[46] Crowley Michael and Andrew Woodward. Does your wireless lan have criminal intent? In *4th Australian Information Security Management Conference*, 2006.

[47] OUT-LAW.COM. Woman forces us record industry to drop file-sharing case. online - theregister.com `http://www.theregister.co.uk/2006/08/03/industry_drops_filesharing_case/`, 2008.

[48] S. C. Lee and C. Shields. Tracing the source of network attack: A technical, legal and societal problem. In *IEEE Workshop on Information Assurance and Security*, 2001.

[49] Robert Morris. A weakness in the 4.2 bsd unix tcp/ip software. Technical report, AT&T Bell Laboratories, 1985.

[50] S. M. Bellovin. Security problems in the tcp/ip protocol suite. *ACM SIGCOMM Computer Communication Review*, 19(2):32–48, 1989.

[51] A. Koyfman, T. Doeppner, and P. Klein. Using router stamping to identify the source of ip packets. In *ACM Conference on Computer and Communications Security CCS*, 2000.

[52] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for ip traceback. In *IEEE INFOCOM*, pages 878–886, 2001. Vol. 2.

[53] Kihong Park and Heejo Lee. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. In *12th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 338–347, 2001. volume 1.

[54] Michael T. Goodrich. Efficient packet marking for large-scale ip traceback. In *9th ACM conference on Computer and Communications security*, pages 117–126, 2002.

[55] S. Bellovin. The security flag in the ipv4 header - ietf rfc 3514, informational. `http://www.ietf.org/rfc/rfc3514.txt?number=3514`, 2003.

[56] E. Jones, O. Le Moigne, and J. M. Robert. Ip traceback solutions based on time to live covert channel). In *12th IEEE International Conference on Networks ICON*, pages 451–457, 2004. Vol. 2.

[57] M. Adler. Tradeoffs in probabilistic packet marking for ip traceback. In *34th ACM Symposium on Theory of Computing STOC*, 2002.

[58] R. P. Laufer, P. B. Velloso, D. de O. Cunha, I. M. Moraes an M. D. D. Bicurdo, M. D. D. Moreira, and O. C. M. B. Duarte. Towards stateless single-packet ip traceback. In *32nd IEEE Conference on Local Computer Networks*, pages 548–555, 2007.

[59] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[60] André O. Castelucio, Ronaldo M. Salles, and Artur Ziviani. An as-level ip traceback system. In *CoNEXT '07: Proceedings of the 2007 ACM CoNEXT conference*, pages 1–2, 2007.

[61] Gu Hsin Lai, Chia-Mei Chen, Bing-Chiang Jeng, and Willams Chao. Ant-based ip traceback. *Expert Systems with Applications*, 34(4):3071–3080, 2008.

[62] Yi Shi, Yong Qi, and BinXia Yang. Deterministic link signature based ip traceback algorithm under ipv6. *10th Int. Conference on Advanced Communication Technology, ICACT*, 2:1010–1014, 2008.

[63] S. Bellovin, M. Leech, and T. Taylor. Icmp traceback messages - ietf internet draft. `http://www3.tools.ietf.org/html/draft-ietf-itrace-00`, 2000.

[64] A. Mankin, D. Massey, W. Chien-Lung, S.F. Wu, and L. Zhang. On design and evaluation of intention driven icmp traceback. In *10th International Conference on Computer Communications and Networks*, pages 159–165, 2001.

[65] Vadim Kuznetsov, Helena Sandström, and Andrei Simkin. An evaluation of different ip traceback approaches. In *4th International Conference on Information and Communications Security*, pages 37–48, 2002.

[66] T. Baba and S. Matsuda. Tracing network attacks to their sources. *IEEE Internet Computing*, 6(2):20–26, Mar/Apr 2002.

[67] K. Shanmugasundaram, H. Brönnimann, and N. Memon. Payload attribution via hierarchical bloom filters. In *11th ACM Conference on Computer and Communications Security*, pages 31–41, 2004.

[68] T-H Lee, W-K Wu, and T-Y W. Huang. Scalable packet digesting schemes for ip traceback. In *IEEE International Conference on Communications*, pages 1008–1013 Vol.2, 2004.

[69] L. Zhang and Y. Guan. Topo: A topology-aware single packet attack traceback scheme. In *International Conference on Security and Privacy in Communication Networks*, 2006.

[70] C. Gong, L. Trinh, T. Korkmaz, and K. Sarac. Single packet ip traceback in as level partial deployment scenario. In *IEEE Global Telecommunications Conference*, 2005.

[71] V. L. L. Thing, M. Sloman, and N. Dulay. Non intrusive ip traceback for ddos attacks. In *ACM Symposium on Information, Computer and Communicat ions Security ASIACCS*, pages 371–373, 2007.

[72] J. Li, M. Sung, J. Xu, and L. Li. Large-scale ip traceback in high-speed internet: Practical techniques and theoretical foundation. In *IEEE Symposium on Security and Privacy*, pages 115–129, 2004.

[73] R. Chen, J-M.Park, and R. Marchany. Track: A novel approach for defending against distributed denial-of-service attacks. Technical report, Virginia Polytechnic Institute and State University, 2005. TR-ECE-05-02 Dept. of Electrical and Computer Engineerig.

[74] Y-N. Jing, P. Tu, X-P. Wang, and G-D. Zhang. Distributed-log-based scheme for ip traceback. In *International Conference on Computer and Information Technology CIT*, pages 711–715, 2005.

[75] B. Al-Duwairi and M. Govindarasu. Novel hybrid schemes employing packet marking and logging for ip traceback. *IEEE Transactions on Parallel and Distributed Systems*, 17(5):403–418, May 2006.

[76] A. Fadlallah and A. Serhrouchni. Psat: Proactive signaling architecture for ip traceback. *Communication Networks and Services Research Conference CNSR*, 2006.

[77] S. J. Templeton and K. E. Levitt. Detecting spoofed packets. *DARPA Information Survivability Conference and Exposition*, 1:164–175, April 2003.

[78] 'Fyodor'. Remote os detection via tcp/ip stack fingerprinting. Available from `http://nmap.org/nmap-fingerprinting-article.txt` retrieved 11 Aug 2008, 1998.

[79] Honeynet Project. Know your enemy: Passive fingerprinting. Available from `http://www.honeynet.org/papers/finger` retrieved 11 Aug 2008, 2002.

[80] Franck Veysset, Olvier Courtay, and Olivier Heen. New tool and technique for remote operating system fingerprinting. Available from `http://www.lot3k.org/biblio/fingerprinting/english/ring-full-paper.pdf` retrieved 11 Aug 2008, 2002.

[81] Steven M. Bellovin. A technique for counting natted hosts. In *ACM SIGCOMM Workshop on Internet Measurement*, pages 267–272, 2002.

[82] V. Jacobson, R. Braden, and D. Borman. Tcp extensions for high performance - ietf rfc 1323. `http://www.ietf.org/rfc/rfc1323.txt?number=1323`, 1992.

[83] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing - ietf rfc 2267, informational. `http://www.ietf.org/rfc/rfc2267.txt?number=2267`, 1998.

[84] H. Wang, C. Jin, and K. G. Shin. Defense against spoofed ip traffic using hop count filtering. *IEEE/ACM Transactions on Networking*, 15(1):40–53, 2007.

[85] J. Postel. Character generator protocol - ietf rfc 864. `http://www.ietf.org/rfc/rfc864.txt?number=864`, 1983.

[86] R. Stone. Centertrack: An ip overlay network for tracking dos floods. In *IEEE Workshop on Information Assurance and Security*, 2001.

[87] H. Hazeyama, Y. Matsumoto, and Y. Kadobayashi. Design of an fdb based intra-domain packet traceback system. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 1313–1318, March 2008.

[88] Yi-an Huang and Wenke Lee. Hotspot-based traceback for mobile ad hoc networks. In *WiSe '05: Proceedings of the 4th ACM workshop on Wireless security*, pages 43–54. ACM, 2005.

[89] Yongjin Kim and Ahmed Helmy. Attacker traceback with cross-layer monitoring in wireless multi-hop networks. In *SASN '06: Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pages 123–134. ACM, 2006.

[90] L.T. Heberlein, G.V. Dias, K.N.Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *IEEE Symposium on Research in Security and Privacy*, pages 296–304, 1990.

[91] Yang-Seo Choi, Dong il Seo, Sung-Wan Sohn, and Sang-Ho Lee. Network-based real-time connection traceback system (nrcts) with packet marking technology. In *ICCSA*, 2003.

[92] Yin Zhang and Vern Paxson. Detecting stepping stones. In *IX USENIX Security Symposium*, 2000.

[93] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur. Dids (distributed intrusion detection system) - motivation, architecture, and an early prototype. In *XIV National Computer Security Conference*, pages 167–176, 1991.

[94] Brian Carrier and Clay Shields. The session token protocol for forensics and traceback. *ACM Transactions on Information and System Security*, 7(3):333–362, 2004.

[95] Xinyuan Wang, Douglas S. Reeves, S. Felix Wu, and Jim Yuill. Sleepy watermark tracing: An active network-based intrusion response framework. In *XVI IFIP International Conference on Information Security*, 2001.

[96] Vyas Sekar, Yinglian Xie, David A. Maltz, Michael K. Reiter, and Hui Zhang. Toward a framework for internet forensic analysis. In *ACM SIGCOMM Hot Topics in Networks*, 2004.

[97] Yinglian Xie, Vyas Sekar, D.A. Maltz Michael Reiter, and Hui Zhang. Worm origin identification using random moonwalks. In *IEEE Symposium on Security and Privacy*, pages 242–256, 2005.

[98] Yinglian Xie, Vyas SEkar, Michael Reiter, and Hui Zhang. Forensic analysis for epidemic attacks in federated networks. In *IEEE International Conference on Network Protocols*, pages 43–53, 2006.

[99] Hyung Woo Kang, Soon Jwa Hong, and Dong Hoon Lee. Matching connection pairs. In *PDCAT*, 2004.

[100] Wei Yu, Xinwen Fu, Steve Graham, Dong Xuan, and Wei Zhao. Dsss-based flow marking technique for invisible traceback. In *IEEE Symporium on Security and Privacy*, 2007.

[101] United States Department of Defense. Department of defense trusted computer system evaluation criteria. available online from NIST CSRC, December 1985. `http://csrc.nist.gov/publications/history/dod85.pdf`.

[102] Wietse Venema. Tcp wrapper: Network monitoring, access control and booby traps. In *USENIX Security III Symposium*, 1992.

[103] M. St. Johns. Identification protocol - ietf rfc 1413. `http://www.ietf.org/rfc/rfc1413.txt?number=1413`, 1993.

[104] Rich Seifert. *The Switch Book: The Complete Guide to LAN Switching Technology.* John Wiley & Sons, Inc., 2000.

[105] IEEE. Standard for local and metropolitan area networks: Media access control (mac) bridges. Technical Report 802.1D - 2004, IEEE, Feb 2004.

[106] Inc. Sourcefire. Snort - the de facto stadard for intrusion detection/prevention, 2005. `http://www.snort.org`.

[107] LinSysSoft Technologies Pvt. Ltd. Kgdb: Linux kernel source level debugger, 2004-2006. `http://kgdb.linsyssoft.com/about.htm`.

[108] A. Rubini and J. Corbet. *Linux Device Drivers.* O'Reilly Media, 2001.

[109] Panagiotis Manolios. Bloom filter calculator, 2004. `http://www.cc.gatech.edu/fac/Pete.Manolios/bloom-filters/calculator.html`.

[110] C. Shannon, E. Aben, K.C. Claffy, and D. Anderson. The caida anonymized 2007 internet traces, 2007. `http://www.caida.org/data/passive/passive_2007_dataset.xml`.

[111] CAIDA OC48 Trace Project. Caida oc48 traces 2002-08-14, 2003-01-15, 2003-04-24 (collection), 2008. `http://www.caida.org/data/passive/index.xml#oc48`.

[112] WIDE Project. Packet traces from wide backbone, mawi working group traffic archive, 2008. `http://mawi.wide.ad.jp/mawi/`.

[113] CAIDA Macroscopic Topology Project Team. CAIDA skitter Topology Traces (collection), 2007. `http://www.caida.org/tools/measurement/skitter/`.

[114] Van Jacobson, Craig Leres, and Steven McCanne. tcpdump, 2009. `http://www.tcpdump.org/`.

[115] Gerald Combs et al. Wireshark network protocol analyser, 2009. http://www.wireshark.org/.

[116] Cooperative Association for Internet Data Analysis CAIDA. Internet measurement data catalog, 2009. http://www.datcat.org.

[117] CISCO Systems. Cisco ios netflow white papers. http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html, 2008.

[118] J. Quittek, T. Zseby, F. Fokus, B. Claise, and S. Zander. Requirements for ip flow information export (ipfix) - ietf rfc 3917, informational. http://www.ietf.org/rfc/rfc3917.txt?number=3917, 2004.

[119] B. Claise. Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information - ietf rfc 5101, standards track. http://www.ietf.org/rfc/rfc5101.txt?number=5101, 2008.

[120] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *ACM Special Interest Group on Data Communication SIGCOMM '04*, pages 416–428, 2004.

[121] Data protection act 1998. Act of UK Parliament, 1998. http://www.opsi.gov.uk/acts/acts1998/ukpga_19980029_en_1.

[122] Regulation of investigatory powers act 2000. Act of UK Parliament, 2000. http://www.opsi.gov.uk/acts/acts2000/ukpga_20000023_en_1.

[123] UK Home Office. Protecting the public in a changing communications environment, April 2009. http://www.homeoffice.gov.uk/documents/cons-2009-communications-data.

[124] Council European Parliament. On the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Union*, L 281:31–50, November 1995. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML.

[125] Richard Ford. 'big brother' database for phones an e-mails. *Times Online*, May 20th, 2008. http://business.timesonline.co.uk/tol/business/industry_sectors/telecoms/article3965033.ece.

[126] K. Retzer and J.J. Vanto. Data retention: Denmark is first eu member state to implement controversial directive. *Privacy and Security Law Report, BNA*, 6(18):717–719, 2007.