2012

A deployable IP spoofing defence system

Lei Wang
University of Wollongong

A Deployable IP Spoofing Defence System

A thesis submitted in fulfillment of the
requirements for the award of the degree

Master of Computer Science

from

UNIVERSITY OF WOLLONGONG

by

Lei Wang

School of Computer Science and Software Engineering
July 2012

Dedicated to
My Parents and My Girlfriend

Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Lei Wang
July 12, 2012

Abstract

It is certain that IP spoofing remains a serious issue at present over the Internet. Our work in this thesis is to offer an authentic source identifier which may be employed by a network for IP spoofing packets. In this thesis, we provide an original model together with an examination of a Spoofing Defence System (SDS) in an Autonomous System (AS) level designed for filtering the spoofed IP packets. An Inter-Domain Routing Validator (IRV) handles the secret keys and exchange keys among different ASes via a secure communication channel. As it is a secure system, the SDS is definitely effective and easily employed cooperatively with other defence systems.

Acknowledgement

Dr. Tianbing Xia (my supervisor) has made this Master Degree possible. I would like to extend my regards and thanks to him. I thank him for his understanding of the value of applied research, the difficulty in performing it and his encouragement and guidance throughout the candidature.

Thanks to Prof. Jennifer Seberry (my co-supervisor) for the help she gave me whenever I needed it.

I am also grateful for the help and assistance that the staff, and colleagues in School of Computer Science and software Engineering (SCSSE) provided to me.

Finally, I would like to express my gratitude to my parents, for their support and patience throughout my study.

Publications

The following publication is included in this thesis:

Lei Wang, Tianbing Xia and Jennifer Seberry, **Inter-domain routing validator based spoofing defence system**, *Proceedings of the 2010 IEEE International Conference on Intelligence and Security Informatics, ISI 2010, (Eds) Christopher C. Yang, Daniel Zeng, Ke Wang, Antonio Sanfilippo, Herbert H. Tsang, Min-Yuh Day, Uwe Glser, Patricia L. Brantingham, and Hsinchun Chen, May 23-26, 2010 - Vancouver, BC, Canada,* 153-155. IEEE Catalog Number: CFP10ITI-PRT, ISBN: 978-1-4244-6460-9

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Founded on the Internet Protocol (IP) [Pos81], the Internet is a critical component of modern society. The IP address of the sending host should be included when packets are sent using the Internet Protocol on the Internet. The receiver who replies to the sender using this IP address is not verified by the protocol, which causes the problem that the attacker can forge the source address. The problem is well-known and described by Bellovin [Bel04]. IP Spoofing may be used by attackers as in the following examples. First, Distributed Denial of Service (DDoS) [LRST00] attacks have cost millions of dollars for Internet companies for which IP spoofing is a very popular method of attack. The DDoS attack is hard to terminate due to the hackers' ability to hide their IP addresses by IP Spoofing making them impossible to locate. Second, by assessing a sensitive computer or a network, the attackers can make it appear like a malicious message was from a credible address.

1.1 Motivation and Contribution

It would be valuable for multiple purposes to ensure IP packets are carrying true source addresses [Bel04]. First, services could rely on correct source addresses (source-based traffic control schemes, congestion control, fair queuing). Second, it can simplify the diagnosis of network problems. Third, this helps to solve one of network security's major issues: attackers escaping from their responsibilities for generating malicious packets. Examples of adversarial behavior include DDoS attacks [LRST00], TCP SYN flooding attacks [SKS+97], and smurf attacks [Tea10].

Many solutions have been proposed to defend against IP spoofing attacks. Most of them are based on the incoming interface, filtering packets, and the IP source address [EL09]. In this thesis, we present an alternative solution to the Inter-domain Routing Validator (IRV) [GAG+03] based IP spoofing defence system (SDS) which

is a router based packet marking solution. Unfortunately, similarly with other router based solutions, the efficiency of the system depends on the number of participating routers [EL09]. Thus, the cooperation with different systems is considered in our system. The IRV architecture is used with the Border Gateway Protocol 4 (BGP4) [RLH06] to validate data and thus obtain additional routing information relevant to an Autonomous System (AS) [RLH06]. It also solves some BGP security issues [GAG+03] about BGP neighbor authentication, AS number authentication.

The purpose of our system is to identify the spoofing packets by providing a crypto identifier which can be used by the network. To enable the Spoofing Defence System (SDS) some participants (usually ASes' border routers) are required to mark the outgoing packets with a keyed Message Authentication Code (MAC) [BCK96], and verify the authenticity of the MAC on incoming packets. Our proposed system is essentially a packet marking system which is more efficient than other defence systems. In the SDS architecture a MAC is added to each packet, to validate that the packet has not been spoofed.

The IP layer is the largest common protocol of the Internet, all Internet protocols run over it [Pos81]. Therefore, any spoofing attacks over Internet protocols, such as Transmission Control Protocol (TCP), User Datagram Protocol (UDP) or other protocols, can be captured by SDS by adding the MAC to the IP header.

SDS can work well with other IP defence systems. To simplify the implementation and development of the SDS, the MAC could be used in the IP option field of the IP header. Most router-based systems use IP ID field [EL09] which may lead to problems of collaboration with other IP spoofing defence systems. Thus, we use the IP option field in the SDS to avoid such problems.

SDS can be implemented via digital signatures. A source signs its packets, and routers validate the digital signatures with the source public key. This approach has been discarded as the digital signatures are computationally expensive to generate as well as verify [BCR08]. Instead, our SDS uses an efficient symmetric key MAC (UMAC) [Kro06, BHK+99] as its tag.

Exchanging key information between different ASes routers is required in SDS. It is important in our system to make sure the key exchange is secure. Some IP spoofing defence systems exchange secret keys on BGP UPDATE message, such as the Passport system [LLYW08]. This is insecure due to various BGP security issues, for example prefix hijacking [Mur06]. Thus, we use Inter-Domain Routing Validator Border Gateway Protocol (IRV-BGP) [BBL05] to solve this problem. Considering

the implementation of IRV, it must be simple, robust, and built on widely deployed technology. Obviously, an HTTP server easily fulfills these requirements [SBEL02], so our system implements IRV as a web-based service.

1.2 Overall Structure

Chapter 2 provides an introduction related to the area of IP spoofing defence systems which will be used later in this thesis.

Chapter 3 states details of our IP Spoofing Defence System (SDS). In section 3.1, our general IP spoofing system is proposed. Then in section 3.2 we discuss the details of our system implementation. Finally in section 3.3, the design of the whole system is given.

Chapter 4 gives details and some benefits of SDS. Firstly in section 4.1, implementation of our system in real world is given. Then in section 4.2, our experiment on SDS is described in detail. After that in section 4.3 we detail a prototype implementation of SDS.

Chapter 5 describes SDS security and experiment results: how SDS has been used in the real system. The SDS security is considered under different kinds of attacks in section 5.1. In order to indicate the performance and feasibility of our system, some real world deployment experiment results are analyzed in section 5.2. Section 5.3 uses modeling to study the adoptability and deployment benefit of our system. In the last section 5.4, the SDS characteristics will be analyzed and compared with various spoofing defence mechanisms.

Chapter 6 states the direction of our future work and some discussions of SDS.

The work in Chapter 3 has appeared in ISI 2010 [WXS10].

Chapter 2

Literature Review

This chapter provides an introduction related to the area of IP spoofing defence systems, which is used in the subsequent. According to Bellovin's research, the IP spoofing problem as being a protocol-level problem is extremely difficult to solve, particularly if the compatibility while using a pre-installed base has to be maintained [Bel04]. Additionally it is widely known that in the present Internet, opponents can easily forge just about every type of the source address in IP packets [TL03]. Whenever opponents inject packets with spoofed source addresses into the Internet, routers forward those packets to their destination similarly to any other packets without verifying the validity of the packets' source addresses. These types of spoofing packets use up network bandwidth from routers to their destinations, and are generally parts of several malicious activities, including DoS and DDoS [LRST00] attacks. Routers over the internet presently are unable to successfully filter out spoofing packets. They either have no idea of any way to identify valid packets from spoofed packets, or even do not obtain this kind of information [Bel04].

2.1 Background

The research community has considered tackling IP spoofing for many years. Early work such as ingress/egress filtering [Fer00], was continuing through to the present with modern solutions such as Passport [LLYW08] and Inter-Domain Packet Filtering (IDPF) [DYC08] which have been proposed for IP spoofing prevention. Although many solutions have shown great promise, the IP Spoofing problem has not been solved.

For a better understanding of the spoofing abilities of attackers, we firstly describe results from the MIT Spoofer Project [MIT10]. The volunteers (located in different places through the Internet) participate in this project trying to send spoofing

packets to Spoofer Project servers which measure the spoofing ability of the hosts. From the project data in *Figure 2.1*, we notice that IP source address spoofing remains a severe problem on the Internet. With the widespread adoption of ingress and egress filtering, people may feel that the spoofing problem has been solved. However, the truth is attackers can still spoof a significant portion of existent IP addresses, often any IP address [MIT10].

As Ehrenkranz and Li mentioned in their paper, spoofing defence mechanisms should also maintain a set of desired properties [EL09].

1. The system cannot rely on any value which attackers can easily modify.

2. The system should be easy to deploy, and its protocol should be able to ensure the ability to deploy across all current and future networks.

3. The system should perform low overheads, in other words, it cannot affect network performance.

We would like to point out that this thesis studies IP spoofing, definitely not IP prefix hijacking. Even though they both equally are conducted by attackers pretending to use an incorrect identity, the difficulties are usually different. Murphy determined that once an attacker effectively hijacks a prefix, the hijacked IP addresses would be held by both the attacker and the established owner [Mur06]. However some packets on the way to the hijacked IP addresses may possibly still be received by the established owner, lots of packets will certainly reach the attacker [Mur06]. Whenever an attacker employs IP spoofing, the spoofed source addresses are actually out of the attacker's control. Most of the IP spoofing defence mechanisms believe that attackers cannot get replies, and would not succeed in defending against attackers which employ IP prefix hijacking [BQS09]. Prefix hijacking is definitely a significant network security issue, however it usually requires methods completely different from those for IP spoofing [EL09].

2.1.1 Spoofing

Some may possibly believe that IP spoofing is no longer a problem while using universal deployment with ingress/egress filtering [Fer00]. Enhance in Botnes which often indicates the attackers who will no longer require spoofing [IH05]. To demonstrate previous conclusions have not really been correct, we have simply to check out

the results from the MIT Spoofer Project, as well as how attackers employ Botnets. Actually, IP spoofing remains a method to make successful malicious attacks [EL09].

MIT Spoofer Project

The Spoofer Project measures the IP spoofing ability of hosts through the Internet and generates the granularity of any ingress/egress filtering spoofing packets from those hosts [MIT10].

Volunteers in the Spoofer Project can download and run a testing program on their end-hosts. The program could be downloaded at: `http://spoofer.csail.mit.edu/software.php`. According to Beverly and Bauer's analysis [BB05], this program transmits forged source addresses IP packets to a Spoofer Project server, where the forgery is able to use different allocated addresses. When selecting allocated addresses to spoof, this program attempts to iterate throughout nearby netblocks from the volunteer host (all the traffic coming from a neighboring /32 to a /9 netblock) [BB05]. Note we use Classless Inter-Domain Routing (CIDR) to specify IP subneting in this thesis. For example [MIT10], 192.168.1.0/23 applies the network mask 255.255.254.0 to the 192.168 network, starting at 192.168.1.0. This notation represents the address range 192.168.1.0 - 192.168.2.255. In the IP Spoofing project, for example, consider a host with IP address 207.76.187.165 that runs the testing program. The program will send a packet with spoofed source address 207.76.187.166 (the "neighbor" /32 block in 207.76.187.165/31). Then it will send packets with source addresses 207.76.187.163, 207.76.187.159, etc., to test the "neighbor" blocks in 207.76.187.165/30, /29, /28, etc. Interacting through neighboring netblocks gives the filtering granularity, in the other words, how large of a netblock a volunteer end-host can successfully spoof [MIT10].

The Spoofer Project keeps an eye on how many hosts can successfully spoof addresses, and the filtering granularity for such hosts. Table 2.1 indicates data from the Spoofer Project on Jun 7 2010. Assuming the volunteer end-hosts make up a representative sample of all Internet hosts, this project can then estimate the amount of spoofing possible on the Internet [BB05].

The outcomes from the Spoofer Project demonstrate that hosts allowed to execute spoofing constitute a large number of Internet addresses. At the time of June 7, 2010, the Spoofer Project reports hosts at 17.0% of all Internet addresses are capable of doing spoofing, or around 488,000,000 out of 2, 880,000,000 addresses. Be aware

Table 2.1: Summary of the Spoofer Project on Jun 7 EST 2010 [MIT10]

	Spoofable	Unspoofable
Netblocks	15.2% (877)	84.8% (4910)
IP Addresses	17.9% (962447)	82.9% (467778824)
Autonomous Systems	24.4% (567)	75.6% (1760)

that as 17.0% are capable of doing spoofing, it dose not mean that attackers can spoof 100% of all source IP addresses.

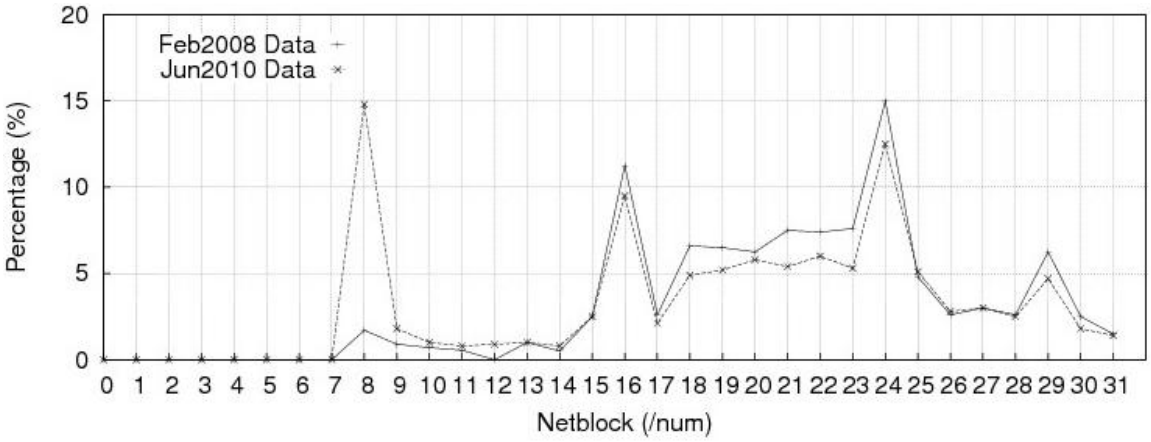


Figure 2.1: Data from the Spoofer Project [MIT10]

Looking at the filtering granularity helps us to know how many source IP addresses attackers can spoof. If we compare the original results to the current online results (*Figure 2.1*), we can see that less than 2% of the spoofing was originally occurring at /8 granularity, but has currently increased to around 15% at /8 granularity. Unfortunately, the number of hosts that could spoof any address within a /8 netblock has significant increased in recent years.

If we look at the number of participating hosts in the Spoofer Project, the situation may be even worse. There are only 7, 980 [BB05] hosts which took part in the Spoofer measurement. *Figure 2.2* illustrates the location of these hosts participate in the Spoofer Project. The United States as well as Europe appear effectively represented, on the other hand small number of hosts come from other locations of the world, the final results could possibly be highly biased to U. S and European networks. If the networks in other part of the world are less secure, the actual number of spoofing addresses might be significantly greater.

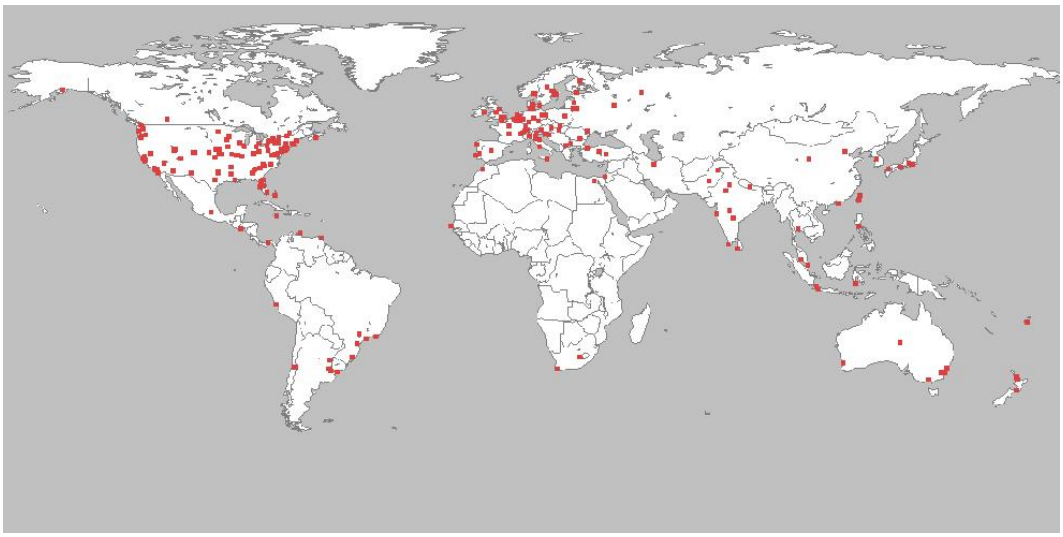


Figure 2.2: Location of spoofable networks [MIT10]

Botnets

Some believe attackers do not have to employ IP spoofing once they can easily control tens of thousands of zombie nodes. The fact is, when considering Botnets, IP spoofing is still a challenge for defenders [IH05]. According to Ianelli and Hackworth's research, many Botnet-based attacks, including a DNS DDoS amplification attack, the IP Spoofing is definitely the key towards the attack's success [IH05]. In some other attacks, IP spoofing provides Botnet owners another layer of anonymity, and guards their zombies anonymous as long as possible [BQS09]. In such situations, it really is mission impossible to identify or even disconnect zombies for network defenders.

Even if Botnets did not employ IP spoofing, the actual threat of IP spoofing would remain in existence. As network defenders and network security researchers, we have not only to research the best way to prevent recognized attacks, we need to discover ways to reduce any kind of possible threat, as well as handle any fundamental drawback in the network infrastructure [BQS09].

2.1.2 Internet Protocol (IP)

The Internet Protocol (IP) is designed for use in interconnected systems of packet switched computer communication networks as Postel mentioned in his paper [Pos81]. It provides for transmitting blocks of data called datagrams from sources to destinations. Sources and destinations are host identified by fixed length addresses. The

Internet Protocol also provides for fragmentation and reassembly of long datagrams [Com00].

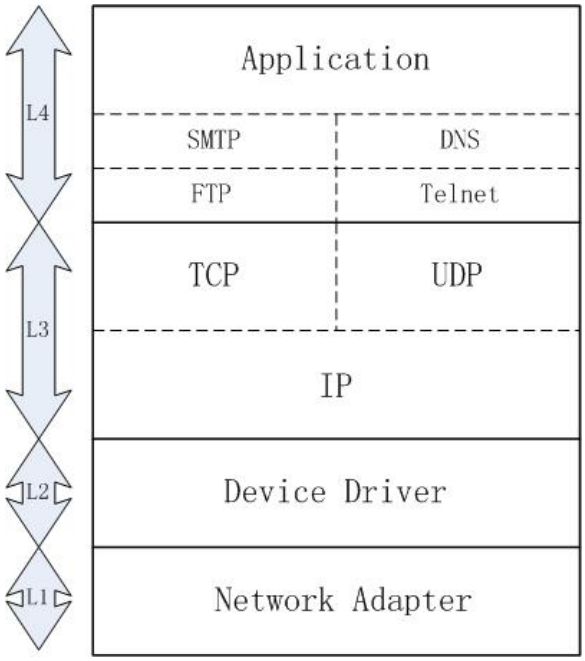


Figure 2.3: The TCP/IP model overview[1]

There are two basic functions in the Internet Protocol: addressing and fragmentation [Pos81]. The Internet modules use the addresses carried in the Internet header to transmit Internet datagrams toward their destinations. The selection of a path for transmission is called routing [Hed88].

The fragmentation and reassembly happen when large datagrams are transmitted though the Internet. Specific fields are used in the Internet header to achieve those proposes [Com00].

In the Internet Protocol, each Internet datagram is an independent entity, in other words, it is unrelated to any other Internet datagram [Com00]. *Figure 2.3* shows the details of Internet Protocol used in our Internet today. It is commonly known as TCP/IP [Bra89b, Bra89a], named from two of the most important protocols in it, the Transmission Control Protocol (TCP) and the Internet Protocol (IP). The IP is in layer 3 of TCP/IP model where TCP and UDP are running over on it.

[1]The contents of the figure come from TCP/IP documentation [Bra89b, Bra89a].

2.1.3 Bound Gateway Protocol (BGP)

The Border Gateway Protocol (BGP) is the most widely used protocol and has been a part of the Internet architecture for many years. The current version used in the Internet is Border Gateway Protocol version 4 (BGP4) [RLH06, RL95]. The protocol has faced new requirements and new problems during these years. Therefore many changes, developments and security additions have been included to enhance the original functionality. The extended version of BGP4 is BGP4+, which is not only supported for IPv4 but also for IPv6.

The Internet is a vast global computer network. As the Internet keeps on growing, administrative control of the network at the global level is extremely difficult. Thus the whole Internet is divided into several organizations called Autonomous Systems (ASes) for administrative convenience. The concept of Autonomous System (AS) was developed to divide the Internet into small parts [IAN09]. As a single administrative authority, an autonomous system contains a set of networks and routers. The Internet Assigned Numbers Authority (IANA) [IAN09] assigns an AS number to every single AS as its identification. For example, the current IP addresses and AS numbers are assigned by APNIC in Asia-Pacific countries.

The communication between two ASes is often called as inter-domain or external routing protocol. The first inter-domain routing protocol was Exterior Gateway Protocol (EGP) and its specification was defined in RFC 827 [Ros82]. The EGP can be seen as the basis for the BGP. The communication in an AS is often called as intra-domain or internal routing protocol. The intra-domain protocol is known as Interior Gateway Protocol (IGP) which is used to exchange routing information among routers within a single AS. *Figure 2.4* shows where IGP and BGP running on the Internet. Moreover, *Figure 2.5* illustrates the AS, IGP and EGP relationship. AS100 is running Open Shortest Path First protocol (OSPF), while Intermediate System to Intermediate System Routing Protocol (IS-IS) is running in AS200 and AS300 is running Routing Information Protocol (RIP) to exchange routing information.

BGP is a typical representative of an exterior gateway protocol, which is intended to replace the earlier Exterior Gateway Protocol EGP (Note, EGP here is a specific external gateway protocol, it is different from the entire class of exterior gateway protocol as we mentioned before.) Like many other routing protocols, BGP has also experienced different development stages. The earliest version of BGP (known as

BGP-1) was proposed by the Internet Engineering Task Force (IETF) in 1989. It was not until 1995, there was a mature version of BGP (known as BGP-4) whose specification was defined in RFC1771 [RL95] by IETF. BGP-related research has been going on, and according to the findings, in January 2006 the latest BGP4 standard has been defined in RFC4271 [RLH06]. In order to enable BGP to support IPv6, IETF has extended the BGP4 protocol, and in 2000 the corresponding standard for IPv6 was defined in RFC2858 [Rek00] (known as BGP4+). After that, in January 2007 IETF proposed a new specification RFC4760 [BCKR07] to replace RFC2858.

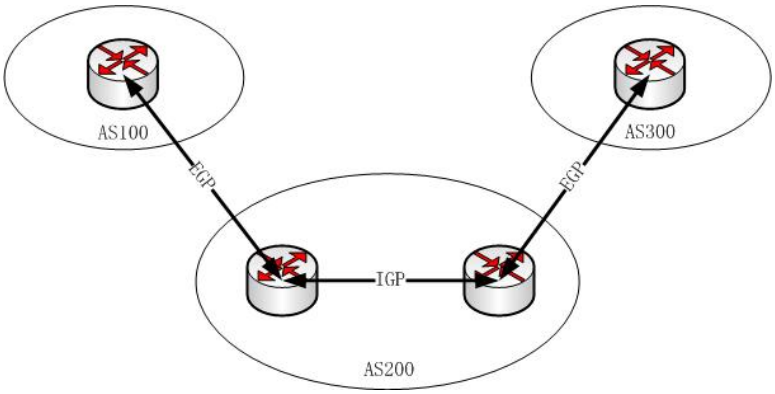


Figure 2.4: EGP and IGP[2]

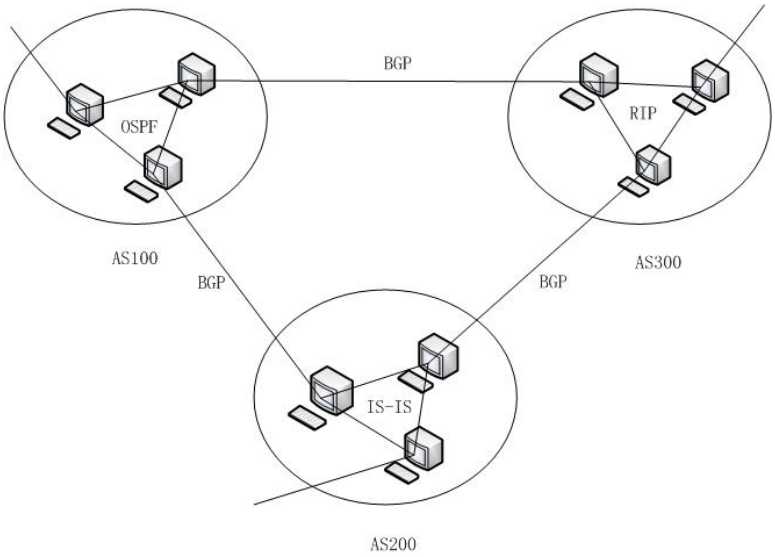


Figure 2.5: AS, IGP, BGP[3]

[2]This relationship figure is summarized from document [Ros82].
[3]This figure is abstracted from document [RL95].

How BGP works

BGP is a path vector protocol used to pass network reachability information among ASes. In other words, BGP builds up an AS reachability information topology through exchanging reachability information among BGP peers. For BGP, the entire Internet is a large AS map, the way to reach any router on the Internet can be given by an AS path. TCP is the transport protocol of BGP, and the port number is

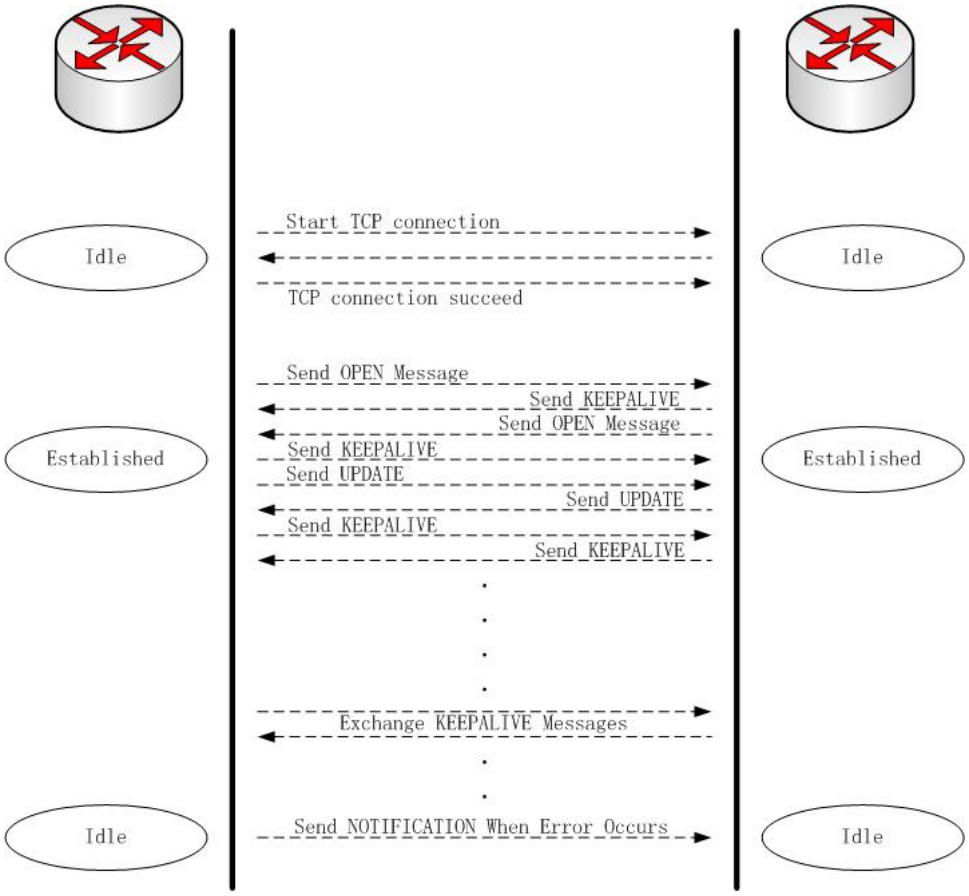


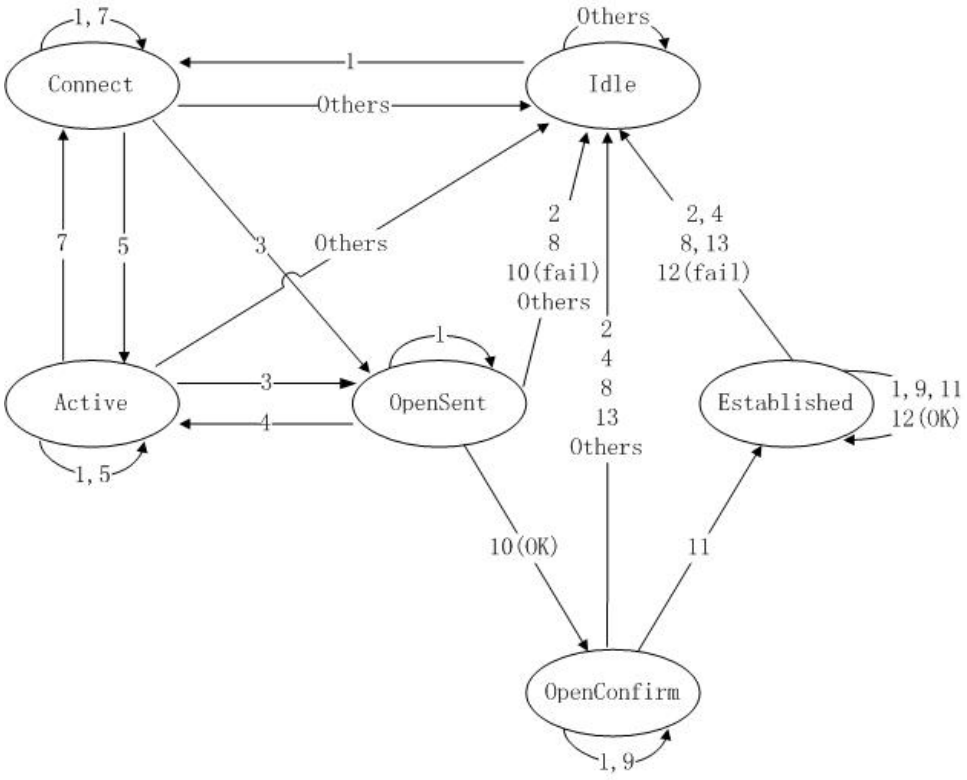
Figure 2.6: The BGP peering state[4]

179. TCP is a connection-oriented transport layer protocol, its performance meets BGP's transport requirements.

BGP4 has five kinds of messages, including OPEN message, KEEPALIVE message, UPDATE message, NOTIFICATION message and the ROUTE-REFRESH message. The first four messages are defined by IETF in RFC4271 [RLH06], ROUTE-REFRESH message is defined in RFC2918 [Che00].

[4]The BGP peering state is summarized from [RLH06, RL95].

Figure 2.6 shows a simple diagram of how BGP works. Two BGP peers establish a TCP connection first, then exchange BGP OPEN message and determine the connection parameters. When the connection has been established, the UPDATE message starts to be exchanged. The whole BGP routing table is exchanged initially. Sending a KEEPALIVE message periodically is to confirm the proper connection. Under error and other exceptional circumstances, the NOTIFICATION message will be sent while the connection is closed down.



1. BGP Start
2. BGP Close
3. TCP connection succeed
4. TCP connection close
5. TCP connection failed
6. TCP connection error
7. Connect Retry Timer overflow
8. KEEPALIVE Timer overflow
9. Hold Timer overflow
10. OPEN message received
11. KEEPALIVE message received
12. UPDATE message received
13. NOTIFICATION message received

Figure 2.7: The BGP FSM [5]

Flowing *Figure 2.7* is a brief summary of BGP operations by a FSM defined in RFC1771 [RL95] and RFC4271 [RLH06]. Before the BGP Connect state, BGP must wait for the transport protocol connection to be completed.

[5]The FSM is summarized from [RL95, RLH06].

Idle The initial state of the FSM. In this state BGP refuses all incoming BGP connections.

Connect In this state, BGP is waiting for the TCP connection to be completed. If the TCP connection succeeds, an OPEN message will be sent to its peer and its state will be changed to OpenSent. If the TCP connection fails, the state stays in the Connect state.

Active BGP is trying to acquire a peer by initiating a TCP connection. If the TCP connection succeeds, an OPEN message will be sent to its peer and its state will be changed to Connect. Otherwise, it stays in the Active state.

OpenSent In this state, BGP is waiting for an OPEN message from its peer. If there are no errors in OPEN message, a KEEPALIVE message will be sent by BGP. Then the state is changed to OpenConfirm. Otherwise, it remains in the OpenSent state.

OpenConfirm BGP is waiting for a KEEPALIVE or NOTIFICATION message in this state. If a KEEPALIVE message is received, the state is changed to Established. On the other hand, if a NOTIFICATION message is received, the state is changed to Idle.

Established The last state in the FSM. BGP can exchange UPDATE, NOTIFICATION and KEEPALIVE messages with its peer in this state.

BGP Vulnerabilities Analysis

As we represented in previous parts, BGP is the *de facto* inter-domain routing protocol, which is widely used in interconnection of Internet Service Provider's (ISP) networks. Our system is working with BGP between different ISPs or ASes, thus the security issues of the BGP must be considered. The vulnerabilities and risks in BGP systems will affect the ISP's network security and stability. RFC 4272 lists three primary limiting factors that lead to the vulnerabilities of BGP [Mur06].

- There is no strong protection of the integrity, freshness, and authenticity of the BGP messages. That means malicious attacks may modify, replay or forge BGP messages in peer to peer BGP communications.

- There is no AS announcement guarantee. An AS can currently announce that it has the shortest path to a destination, even if it is not part of the destination path at all.
- There is no authenticity of the path attributes announced by an AS. A malicious AS can spoof or manipulate the packets' routing path by changing the path attributes in BGP messages.

The following list shows some potential attacks on the BGP. We summarized those attacks from [Bel04, Mur06, MP03, BM92, KS05]. While not a complete list, the attacks discussed here are the most common that are likely to be a concern for BGP.

BGP spoofing attacks The goal of the spoofing attack may be to insert false information into a BGP peer's routing tables. The attacker spoofs the BGP peer by modifying the source address of the peer's IP connection [Mur06].

BGP session resets attacks The sequence numbers of received ICMP messages are not required to be checked by current IETF specifications, which means attackers are easily able to send spoofed ICMP error messages to peer. Hence, it causes loss of BGP peer sessions, and makes it necessary for routing tables to be rebuilt.

Malicious route injection attacks There are three kinds of illegal route injection: DUSA (Documenting Special Use Addresses) route injection, unauthorized route injection and unallocated route injection [MP03]. Disruptions in networks might be caused by malicious route injection of a DUSA addresses. Using unauthorized or unallocated route injection, attackers can make serious DoS and DDoS attacks easily.

Hijacking attacks By announcing a more specific route to a target network, an attacker hijacks the netblock address, then the packets directly flow to the attacker's machine. The YouTube incident [RIP10] is a good example of a netblock address hijacking attack.

We will review and compare different IP spoofing defence solutions. The purpose of this part is to comprehensively introduce IP spoofing technologies, and at the same time to analyze any obstruction of the deployment of solutions. In the following subsection, modern spoofing defence mechanisms will be summarized.

2.2 Related Works

Detection methods can be classified into host-based methods and router-based methods [TL03]. The host-based methods are implemented on a host, and aim to allow a host to recognize spoofing packets, for example IPSec [KS05], the OS fingerprinting [Zal09c, Tal03], TCP probing [Zal09a, Zal09b], SYN cookies [Ber10] and IP puzzles [CKCL05]. The advantages of these methods are that we do not need to change networking infrastructure and they are easy to deploy. On the other hand, they may act too late since the spoofing packets must reach the host before they are detected. The router-based methods are mostly implemented on routers, and the spoofing packets will be recognized before they reach the destination, for example Ingress and egress filtering [Fer00], Reverse Path Forwarding (RPF) [BS04], Router-Based Filtering (RBF) [MJR06], Spoofing Prevention Method (SPM) [BBL05], Passport [LLYW08], Distributed Packet Filtering (DPF) [PL01], Inter-Domain Packet Filtering (IDPF) [DYC08], SAVE [LMMWZ02, LME+08], BASE [LKHP07], Peer-to-Peer Based Anti-Spoofing Method (APPA) [SBWL08], Spoofing Prevention based on Hierarchical Coordination (SP-HCM) [LS07], a packet marking method with Bloom Filter [TMI07], Hop-Count filtering [WJS07], Probabilistic Packet Marking (PPM) [SWKA00], Deterministic Packet Marking (DPM) [BA03], Pi [YPS03] and StackPi [YPS06]. These methods are more efficient than host-based methods, but the efficiency of the system depends on the number of participating routers. It is impossible for every router on the Internet to participate in the system and different ISPs deploy the same method. Thus, the cooperation of different systems should be considered in the new system. Our system SDS is a router-based method and it performs better for cooperation with other methods. We will discuss this in Chapter 3. Table 2.2 summarized those different methods.

Ingress and egress filtering [Fer00] are the most well-known filtering methods which run on a border router. Ingress/egress filtering checks the addresses of packets flowing into and out of an edge network. The border router is easily able to filter out any spoofing packet. If this filtering mechanism was implemented on all networks, attackers would be limited to their own local network. Unfortunately, as we mentioned before, such universal deployment is not a reality. The RPF [BS04] is similar to ingress/egress filtering. The only difference is RPF uses the forwarding table information at a router. Unfortunately, RPF is based on a routing symmetry assumption [BS04], with the high amount of routing asymmetry in the Internet

today, this assumption is not true.

The SPM [BBL05] can also be deployed on a border router. The SPM validates a packet by checking for a secret key embedded into the packet. Any packet with the secret key is valid, and others are spoofing. Unfortunately, packets cannot be checked which come from ASes do not deploy SPM. In the SPM, intermediate routers cannot assist in filtering spoofing packets, only the source or destination AS can filter a spoofing packet.

Routers running the Passport system [LLYW08] define a new header, based on the IP header. This header can be thought as "Passport" which contains lots of "Visas". Each "Visa" indicates one AS on the path that packet passes through. "Visa" in the passport system is a keyed Message Authentication Code (MAC). The Passport system generates MACs embedded in the sending packet, which uses the IP option field that contains a large header. The border router checks the MAC value, if the verification fails, the router will then demote or drop the packet. In order for each pair of ASes to have the secret key, each participating AS performs a Diffie-Hellman key exchange based on BGP update messages. When the packet is too large the router will fragment it. When this happens the passport will be invalid. Obviously key exchange based on BGP message is not secure because it is vulnerable as we analyze before.

The DPF [PL01] assumes that routers maintain incoming direction knowledge of a packet incoming interface. When a spoofed packet arrives at an incorrect interface, the router can detect and filter the packet. The problem in the DPF is the method for routers to learn the incoming direction knowledge. The IDPF [DYC08] attempts to provide an implementation of DPF. The incoming direction knowledge comes from a BGP updates message. The BGP protocol does not provide any way to guarantee that the attributes of a BGP UPDATE message are correct. Thus the IDPF is invalid in this situation.

The SAVE protocol [LMMWZ02, LME+08] operates similarly to the DPF which filters packets based on their incoming direction. The SAVE protocol is the first protocol to help routers to learn the incoming direction knowledge. The SAVE protocol also creates an "incoming tree" which keeps track of the topological relationship of source addresses. When one router change affects the incoming direction, other participating routers will be updated based on the "incoming tree". The SAVE protocol assumes that all routers deploy SAVE protocol which is infeasible.

The BASE protocol [LKHP07] relies on BGP. The basic operation of the BASE

is similar to the SAVE protocol which sends updates to routers to learn the correct incoming direction of packets. So the BASE protocol is path-based packet filtering mechanism. BASE protocol updates must travel the same path as the BGP update. However, the BGP updates do not always travel the same path. AS the BASE protocol uses control messages, it means the BASE routers are only able to respond to the spoofing packet after receiving instructions.

The Pi [YPS03] is originally designed to defend against DoS attacks, but it also provides an IP spoofing defence solution. Pi uses the fragmentation field of an IP packet to identify the path that packet traveled. The destination host cannot identify the attack packet, but if lots of packets contain the same markings it is likely that they travelled the same path. How to identify an attack packet is another problem in the Pi system. The StackPi [YPS06] is the improved version of the Pi system, which added some mechanisms to guard against spoofing packets. The StackPi system store the legitimate packets marking, the host checks the stored marking then filters out any packet without correct marking. The StackPi assumes full deployment which is also infeasible. Moreover, it cannot identify all spoofing packets. The efficiency of low deployment levels is unknown.

The idea of our SDS comes from packet marking systems, such as the SPM [BBL05] and the Passport [LLYW08]. Packet marking seems to be the most efficient method to solve the IP spoofing problem. Most of these systems depend on BGP message e.g. BGP UPDATE message, but the BGP message is not secure, as we analyzed in Subsection 2.1.3. Sometimes we even cannot trust border routers in ASes. The SDS is based on the secure BGP (IRV-BGP) which provides a secure communication channel and authentication to solve both problems. In the following chapter, we will discuss details of the Inter-Domain Validator based SDS.

Table 2.2: Summary of spoofing defence mechanisms

	Host Based	Router Based
A packet marking method with Bloom Filter		√
BASE		√
Deterministic Packet Marking (DPM)		√
Distributed Packet Filtering (DPF)		√
Hop-Count filtering		√
Ingress/engress filtering		√
Inter-Domain Packet Filtering (IDPF)		√
IP puzzle	√	
IPSec	√	
OS fingerprinting	√	
Passport		√
Peer-to-Peer Based Anti-Spoofing Method		√
Pi	√	√
Probabilistic Packet Marking (PPM)		√
Reverse Path Forwarding (RPF)		√
Router-Based Filtering (RBF)		√
SAVE		√
Spoofing Prevention Method (SPM)		√
StaticPi	√	√
SYN cookies	√	
TCP probing	√	

Chapter 3

IP Spoofing Defence System

This chapter addresses details of our IP SDS. The purpose of our system is to use cryptographic primitives to provide an authentic source identifier which can be used by the network to identify spoofing packets. To enable the SDS some participating ASes are required to mark their outgoing packets with a keyed MAC, and verify the authenticity of the MAC on incoming packets.

Firstly, the general IP spoofing system is proposed. Then we discuss the details of our system implementation. Finally, the design of the whole system is demonstrated.

3.1 Overview of IP Spoofing Defence System

The proposed SDS uses an efficient symmetric key MAC construction (UMAC) as its tag to verify whether a source IP address is valid or not. Different ASes border routers obtain shared keys by using a protected Diffie-Hellman key exchange protocol with the Inter-Domain Routing Validator (IRV) servers. To enable the SDS some participating ASes are required to mark the outgoing packets with a keyed MAC, and verify the authenticity of the MAC on incoming packets.

In the SDS architecture a MAC is added to each packet, to validate that the packet is not spoofed. *Figure 3.1* shows how the SDS works at AS level. When a packet leaves its source AS (AS_i), the border router stamps one MAC for the next AS (AS_{i+1}) on its path into its header. Each MAC is computed using a secret key shared between AS pairs (AS_i and AS_{i+1}) on the path, e.g. AS_{100} and AS_{200}, AS_{200} and AS_{300}...AS_i and AS_{i+1}.

When the packet enters the next AS (AS_{i+1}) on the path, the border router verifies the MAC value using the secret key shared with the previous AS (AS_i). A correct MAC can only be produced by the current AS (AS_{i+1}) which shared the secret key with the previous AS (AS_i). If the MAC is verified, it is sufficient to

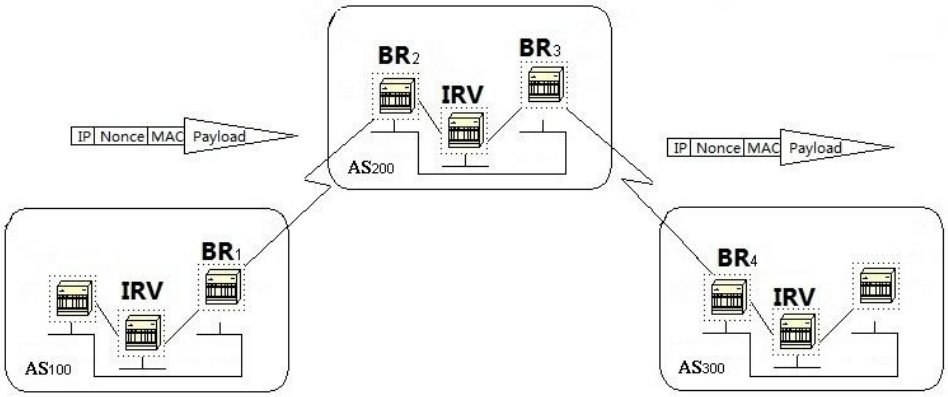


Figure 3.1: An AS level overview of SDS

show that the packet comes from the source AS indicated by its source address, and the border router will generate a new MAC to overwrite the valid one. The new secret key is shared between the current AS and the next AS. Otherwise, it is assumed to be a spoofed packet. The packet with an invalid MAC is discarded at the current AS. We chose to discard spoofed packets because discarding them has the advantage that spoofed packets will not further consume network resources, such as bandwidth. This is an important feature of our SDS, especially under IP Spoofing based DDoS attack. Thus, the spoofed packet will never reach the intended destination. The MAC generation process only happens at the initial stage, once a communication channel is established between border routers then there is no requirement to recompute the MAC and share a new secret key between ASes on the path. The MAC is added into the packet by the current outgoing border router, the next incoming router just compares the MAC value with the previous incoming packet. The time required to verify each packet and share the secret key is short. There are new keys and MAC values generated when a new communication request occurs.

3.2 Spoofing Defence System Implementation

The role of the keyed MAC is to verify that the source address of a packet is not spoofed. As we all know that the IP layer is the largest common protocol of the Internet, all Internet protocols run over it [Pos81]. Thus, adding the MAC to the IP header allows the SDS to capture any spoofing attack over any Internet protocol, no matter whether it is TCP, UDP or any other protocols.

For ease of deployment and implementation of the SDS, the MAC can be placed in the IP option field in the IP header. Most router-based systems use the ID field in the IP header. This may lead to collaboration problems with other IP spoofing defence techniques like SPM [BBL05], Pi [YPS03] and StackPi [YPS06]. Another disadvantage in using the IP ID field is that the IP ID field length is only 16 bits, which is insecure under off-line cryptanalysis attack. So, we chose the IP option field and MAC in the SDS to avoid these problems.

3.2.1 Background on MAC

Obviously, our SDS can be implemented via digital signatures. A source signs its packets, and routers validate the digital signatures with the source's public key. We discard this approach as digital signatures are computationally expensive to generate and verify. Instead, our SDS uses an efficient symmetric key MAC (UMAC) [Kro06] as its authentication information.

Before we move on to the description of our SDS implementation, we need to recall the concept of MAC, as this cryptographic primitive is the core of our SDS construction. After reviewing the basic MAC definitions, we will present two constructions (HMAC and UMAC) and then discuss the UMAC implementation in our SDS.

Message Authentication schemes

A MAC is used to ensure data integrity. Which is a basic requirement of information security preventing data from unauthorized manipulation. With the development of network technology, especially as e-commerce continues to expand, to ensure the integrity of information becomes increasingly important [KS05]. To combat unauthorized data manipulation, ensuring message authenticity, is usually done by a MAC [KBC97]. A MAC verifies the date which the packet claims to have originated from, and the source has not have been altered during transport. Note, encryption does not provide data integrity.

A MAC scheme is specified by three algorithms [KBC97]: a key generation algorithm (K); a tagging algorithm (T) and a verification algorithm (V). The MAC scheme works as follows. The sender and receiver both share the secret key generated by the MAC key generation algorithm and not known to the adversary. The sender processes the key and a tagging algorithm which produces the tag. Then,

the sender sends the message with the tag to the receiver. When the receiver receives a message and tag, he then processes their shared key and the received message with a tagging algorithm. If his generated tag matches the one received with the received message, then the received message is taken to be authentic. Otherwise the message is discarded as unauthentic. A successful message forgery means the adversary is able to create a tag for a new message (without knowledge of the secret key) which together with the message passes the receiver's authentication test.

Compare MACs with digital signatures. In a MAC, two parties share the same secret key which is a symmetric or secret key model. In a digital signature scheme, two parties have the different keys, private key and public key. This is an asymmetric or public key model. Therefore, the digital signature schemes are more flexible than MAC schemes. The fastest digital signature schemes are much slower than the fastest MAC schemes, so MAC schemes are used whenever possible [KBC97].

Let us summarize the above in following definition. A Message Authentication Code scheme **MAC**=(K, T, V) is specified in three algorithms as follows:

Definition 3.1 *The key generation algorithm* K *is a probabilistic polynomial-time algorithm that returns a key* κ*.*

Definition 3.2 *The tagging algorithm* T *is also a probabilistic polynomial-time algorithm which takes key* κ *and message* M*, and then returns a tag* t*, where* $t \leftarrow \mathrm{T}_\kappa(\mathrm{M})$*.*

Definition 3.3 *The verification algorithm* V *is a deterministic algorithm which takes the key* κ*, a message* M *and a tag* t *for* M *to return a bit, where* $\mathrm{V}_\kappa(\mathrm{M}, t)$*.*

We require that $\mathrm{V}_\kappa(\mathrm{M}, \mathrm{T}_\kappa(\mathrm{M})) = 1$ for all $\mathrm{M} \in Message$.

Verification can be performed by computing the correct tag and checking that the transmitted tag matches the correct one. The verification algorithm is simply as follows:

HMAC and UMAC Constructions

We now consider a family of message authentication schemes called HMAC and UMAC due to [KBC97, Kro06].

The keyed-hash based message authentication code is a kind of message authentication coed which uses a cryptographic key in conjunction with a hash function

Require: $V_\kappa(M, t)$
  $t\prime \leftarrow \mathcal{T}_\kappa(M)$
  if $t = t\prime$ **then**
    return 1
  else
    return 0
  end if

[KBC97]. To compute a MAC over the Message(M) using the HMAC function, the following operation is performed:

$$\begin{aligned} & MAC(M)_\tau \\ & = HMAC(\kappa, M)_\tau \\ & = H((\kappa_0 \oplus opad) \parallel H((\kappa_0 \oplus ipad) \parallel M))_\tau \end{aligned}$$

In this function, H is an approved hash function; *ipad* is an Inner pad and *opad* is an Outer pad, they both repeated B times which is the block size of the input to hash function; κ is the secret key; κ_0 is the key κ after any necessary pre-processing to form a B byte key; τ is the number of bytes of MAC and M is the message for which the HMAC was calculated.

Compared with other MAC algorithms, the HMAC has a number of advantages. First, based on Bellare et al's security proof [BCK96], the security of the HMAC is based on the underlying hash function security. Second, the HMAC construction is faster than the MAC which uses a block cipher structure in a software application. The speed of the HMAC is particularly high, because the HMAC uses the key in a very simple way. Compared with the underlying hash function performance, the speed of HMAC is not significantly reduced. Third, the HMAC is free of charge. Last, HMAC is a construction where the underlying hash function can be viewed as a black box.

The disadvantage of the HMAC is that its security of the HMAC is based on the underlying hash function security. However, the security of some hash functions cannot be proved. Second, because the compression function of the HMAC is serialized it does not support parallel processing.

UMAC, as next generation MAC, was designed with two main goals: **extreme speed** and **provable security**. Designers aimed to create the fastest MAC ever. Which justifies our choice of UMAC as a system tag.

UMAC is obviously fast. Determined by Black et al [BHK+99] test on a **350 MHz Pentium II PC**, one version of UMAC (where the adversary seems to have a 2^{-60} possibility of forgery) achieves peak-performance of 2.9 Gbits/sec (**0.98** cycles/byte). A different version of UMAC (with 2^{-30} possibility of forgery) achieves peak-performance of 5.6 Gbits/sec (**0.51** cycles/byte). In contrast, a SHA-1 implementation runs at **12.6** cycles/byte. Note that the SHA-1 speed range is higher than the speed of the HMAC-SHA1 [BCK96]. The fastest among suggested universal hash functions was **MMH** [HK97], which usually runs at around **1.2** cycles/byte (for 2^{-30} possibility of forgery).

In contrast to many MACs, when the sender needs to authenticate some string *Msg* he or she must provide as input to UMAC (using *Msg* along with the shared key Key) a 64-bit string *Nonce*. The *Nonce* should not be used again by the sender in the communication session. Usually the *Nonce* would be a random generator which the sender creates together with every transmitted message.

The way the message, key, and nonce establish an *authentication tag* are actually selected by the UMAC algorithm. The sender is required to provide the receiver the message, nonce and tag. The receiver is able to validate the tag for this specific message and nonce, and finds out if this matches the tag actually received. The receiver might additionally desire to confirm the fact that a nonce has not been used previously, thus avoiding replay attacks.

The UMAC employs a *subkey generation process* similar to several modern ciphers. Inside the subkey generation process the fundamental key is mapped into UMAC's inner keys. In normal applications subkey generation is performed only once, at the beginning of the long-lived session, therefore subkey-generation is frequently not performance-critical [KBC97, Kro06].

Our Implementation

Our design uses UMAC because of its extremely high speed. UMAC takes a *Nonce* as its input, so we generate a random number (pseudo random), place it into the IP header 32-bit nonce field and use it together with the 16-bit IP Identification (IP ID) with Padding to generate a 64-bit *Nonce* for the UMAC computation.

$$Nonce = IP\ Nonce\ field||IP\ ID\ field||Padding$$

where the IP Nonce field=32-bit, IP ID field=16-bit and Padding=16-bit. We describe how we determine the 16-bit padding. We define a function f which can select 16-bit (4-byte) numbers for padding. We define, Padding=$f(K^2)$, K here is the 128-bit secret key. Now as the UMAC input required, the input *Nonce*=32bits+16bits +16bits=64bits. A border router of an AS stamps a MAC for the next AS on the path to the destination. Each MAC is computed using the key which is 128-bit, shared with the next AS on the path. These secret keys are exchanged between ASes as described in subsection *Key exchange among different ASes* on page 29. The MAC computed for the next AS covers the source address (*src*), the destination address (*dst*), the IP Identification (*IP ID*), the packet length field of the IP header (*len*), current and next AS number (*ASN*). Where, src=32bits, dst=32bits, len=5×32bits= 20bytes and ASN=16bits. The IP Identification field is currently used to indicate IP fragments belonging to different packets, but only less than 0.25% of the packets on the Internet actually use this feature [BCR08]. That is why many spoofing defence systems use IP ID field for their marking purpose, such as SPM [BBL05]. Our SDS does not, because both collaboration with different systems and tag security are considered in our system. AS numbers are assigned in blocks by the IANA [IAN09] to Regional Internet Registries (RIRs). A unique ASN is allocated to each AS for use in BGP routing. The ASN is important because it uniquely identifies each network on the Internet. Based on the newer definition in RFC 1930 [HB96], the ISP must have an officially registered ASN when they are running BGP on the Internet.

As we discussed earlier, the hash-function family known as **NH** underlines hashing inside UMAC. It is actually a simplification from the **MMH**: **MMH** families are defined in [HK97]. At this point we are going to describe the way it works. The Message Msg hash is a sequence of an even number l of integers, $Msg = (msg_1, ..., msg_l)$, where each $msg_i \in (0, ..., 2^\omega - 1)$ refers to a ω-bit word (e.g. ω=32). A selected hash function is named by using a sequence of $n \geq l$ ω-bit integers $K = (\kappa_1, ...\kappa_n)$. Now compute $NH_K(Msg)$ as

$$(\sum_{i=1}^{l/2}((msg_{2i-1} + k_{2i-1}) \ mod \ 2^{2\omega}) \cdot ((msg_{2i} + k_{2i}) \ mod \ 2^{2\omega})) \ mod \ 2^{2\omega}$$

In our system, the underlying shared key *Key* (128-bit) is first expanded into internal keys K and N, where K is 1024 words (a *words* being 32-bit long) and N is

512-bit long. How *Key* establishes K and N can be dealt with utilizing any kind of *pseudo-random generator*(PRG), therefore we will not elaborate here. We consider the hash function **NH**, which can be used on each message block $Msg_1, ..., Msg_t$ of Msg. The hash function is named by $K = K_1 \cdots K_{1024}$, where K_i is 32-bit. So we let $NH_K(Msg)$ be the 64-bit string, if $l = 4$

$$NH_K(Msg) = [(Msg_1{+}32K_1){\times}64(Msg_2{+}32K_2)]{+}64[(Msg_3{+}32K_3){\times}64(Msg_4{+}32K_4)]$$

Where +32 is actually computer addition on 32-bit strings to provide the 32-bit add, +64 is computer addition on 64-bit strings to provide the 64-bit add, in addition to ×64 is computer multiplication on unsigned 32-bit strings to provide the 64-bit solution. This is described in *Figure 3.2*.

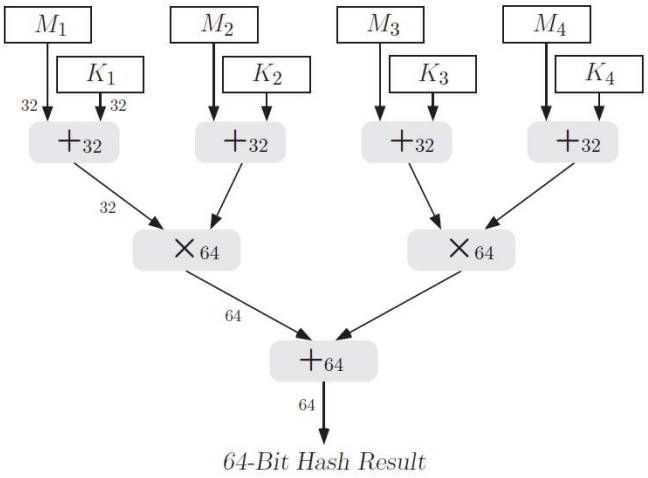


Figure 3.2: The NH hash function to get a 64-bit digest [KBC97]

The particular pseudo random function employed on $NM||Nonce$ is known as a parameter.

In *Figure 3.1*, when a packet is sent from host AS_{100} to host AS_{300}, the border router BR_1 of AS_{100} computes MAC_2 *(src, dst, len, IP ID,* AS_{100}*,* AS_{200}*). Figure 3.3* shows a possible SDS header format used in our implementation. Our design uses a 64-bit MAC for each AS hop. A border router stamps/replaces a SDS header for a packet with a valid source address, and discards the spoofed packet otherwise.

In *Figure 3.1*, when AS_{200} receives a packet from an external interface, it verifies the SDS header using the key it shares with AS_{100}. The verifying AS (AS_{200}) obtains the shared key, and recomputes the MAC using the shared key and the packet fields

Subkey generation:
Require: pseudo-random function generator (PRG)
Map Key to $K = K_1 \cdots K_{1024}$, with each K_i a 32-bit word, and to N, where $|N| = 512$.
Hashing the message:
Require: Msg to $HM = NHS_{Key}(Msg)$
Let Len be $|Msg|$ mod 4096, encoded as a 2-byte string. Append to Msg the minimum number of 0 bits to make $|Msg|$ divisible by 8.
Let $Msg = Msg_1||Msg_2|| \cdots ||Msg_t$ where each Msg_i is 1024 words except for Msg_t, which has between 2 and 1024 words.
Let $HM = NH_K(Msg_1)||NH_K(Msg_2)|| \cdots ||NH_K(Msg_t)||Len$
Computing the authentication tag:
The tag is $Tag = SHA1_N - HMAC(HM||Nonce)$

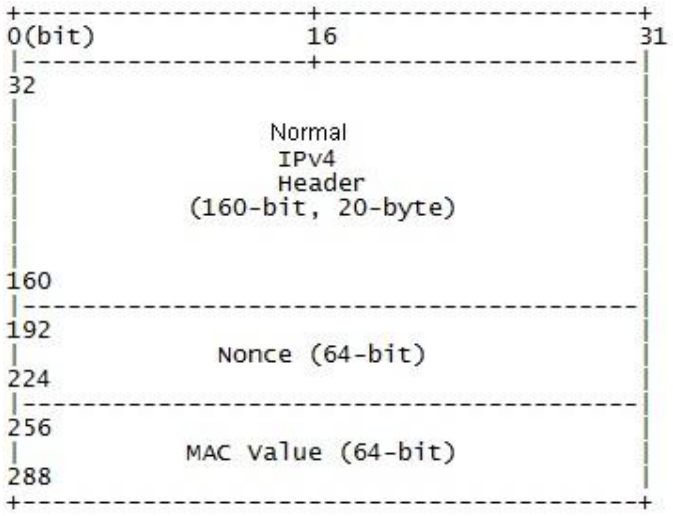


Figure 3.3: Possible header format of a SDS in IPv4

as used by the previous AS. If the source address is not spoofed, the recomputed MAC value will perfectly match the one in the SDS header. The router replaces the valid MAC value with new one which can only be recomputed by the next AS. A router replaces the MAC value after verification to prevent offline cryptanalysis. The secret key is shared between the current AS and the next AS. If the MAC verification fails, the AS border router discards the packet. If the internal interface is a host-to-router interface, e.g, the interface between host and a router BR_1 in *Figure 3.1*, the router discards the packet, as a host cannot generate a valid SDS header. This can only happen at a router-to-router interface.

After initial MAC generation and verification, there is no new MAC generated until communication has finished. For example in *Figure 3.1*, BR_1 does not need

to recompute the MAC (MAC_2), it just stamps the MAC (MAC_2) to the packet. And for verification, when BR_2 receives the packet from BR_1, it does not need to recompute the MAC value (MAC_2), instead of comparing the MAC value with previous one, that's because the MAC value will never change in the same path. If the MAC value is the same as the previous packet MAC value, it is a valid packet. Otherwise, it is a spoofed packet and the router will discard it.

Until now we have said nothing about key exchange. In the following subsection, the secret key exchange and update will be discussed. It will be divided into two part, key exchange among ASes and key exchange between IRV and Border Routers.

3.2.2 Key exchange among different ASes

Exchanging keys among routers in different ASes is required in our system. It will be described in two different circumstances. The first one is how to exchange keys between different ASes or ISPs. The second one is how to exchange the key between IRV and border routers. It is also important in our system to make sure that key exchange is secure.

Some IP spoofing defence systems exchange secret keys between different ASes or ISPs on the BGP, such as Passport [LLYW08]. This is obviously insecure due to various BGP security issues such as prefix hijacking.

Current research indicates that BGP seems to have three serious weaknesses [Mur06]. The very first weakness is there is no system to confirm the integrity, freshness and source authenticity in BGP message. Second, BGP does not provide any system to validate the authenticity of an address prefix and even an AS origin of the prefix inside the routing system. Last, the BGP protocol does not offer any way to ensure that the attributes of a BGP UPDATE message are correct. The possible lack of security methods in BGP results in it vulnerable to several kinds of control plane attacks [IGM09]. There are many recommendations to guarantee BGP with the addition of certificate keys to BGP and announcements to verify them. To secure BGP, two basic ways of key distribution have already been proposed and into account by the Internet Engineering Task Force (IETF). The first one is the central method which is included in S-BGP [KLS00] and in So-BGP [Whi09], where the organization of IP registries (ARIN, APNIC, RIPE) are responsible for distributing the keys. The second one is the IRV method which implements a distributed server (IRV) in every AS to organize the key distribution. We take advantage of the second

method just as used in SPM [BBL05] in order to distribute and manage the secret keys. Please note, the keys distributed here are certainly not the keys we employed in our SDS. Those distributed keys are being used for BGP authentication and verification, to find out more information see [GAG+03].

In our system, we use one central server in each AS to manage the secret keys. The IRV is used to validate the BGP data and acquire additional routing information relevant to an AS. IPSec or TLS can be used to ensure the integrity and authenticity for BGP messages. We modified this method to exchange our secret keys between ASes or ISPs. We assume that a secure communication channel has already been established before our secret key exchange. In our SDS, IRV is not only used to exchange BGP authentication keys, but also used to select and exchange the secret keys for different ASes border routers. We believe the IRV provides better security in partial deployments, which is important for the success of the SDS. Consider the implementation of an IRV server. It must be simple, robust, and built on widely deployed technology. Obviously, the HTTP server easily fills these requirements. Our system implements IRV as a web-based server. Many well known security solutions provide security to web-based services such as SSL/TLS and IPSec. Hence, ASes are free to implement IRV security as is appropriate for their environment which makes our assumption possible.

IRV Architecture

In [GAG+03], Goodell et al provide the IRV service, which acts as a companion to BGP. IRV is commonly employed to identify and detect routing configuration issues. Being a new protocol, IRV defends against fake, subverted, as well as grossly misconfigured ASes. It was eventually developed as a independent protocols due to the problems in adjusting broadly implemented protocols such as BGP [GAG+03]. IRV enables quick and minimal deployment. In [GAG+03], the writers argued that even small teams of ASes will discover immediate benefit from deployment. Additionally, they discussed the employment of IRV by real world samples, and take into consideration its use as an alternative for, and together with, other routing services and protocol extensions.

There are several explanations why existing BGP security solutions have never been broadly implemented. Such are minimal capability to be incrementally implemented, high computational costs and then the infeasibility of modifying the huge

installed based of BGP implementations. Realizing these limitations, the IRV architecture was recommended. IRV is required to verify BGP information and get extra routing information related to an AS when used in conjunction with BGP. According to [GAG+03], IRV provides the following aims:

- It allows ASes to acquire and validate both static and dynamic interdomain routing information.
- IRV provides incrementally deployability, meaning the IRV system provides substantial benefits even with limited adoption.
- Allow ASes to securely differentiate between requesters of routing information, in order that responses be tailored to the recipient.
- The IRV system be able to operate independently of the reception of the BGP message, and ASes to be free to validate and acquire routing information whenever desired.
- Allow ASes to receive routing-relevant information from remote entities.

IRV offers attributes of S-BGP along with the Internet Routing Registry. Similar to S-BGP, IRV helps autonomous systems to make sure that they reported certain routes. Compared with S-BGP, validation information is absolutely not transported in BGP UPDATE messages. Alternatively, designers propose the idea of an IRV. Each individual participating AS appoints an IRV in charge of responding to queries from other ASes. Users of the system query the IRV confirmed the obtained BGP information or to get additional route-relevant information. IPsec or perhaps TLS may be used to guarantee the integrity, authenticity, and timeliness of the queries and responses. From *Figure 3.1* when the IRV in AS_{200} needs to validate an BGP UPDATE message related to AS_{100} that AS_{200} received from one of its neighbors, it queries the IRV located in AS_{100}.

As Goodell et al pointed out in [GAG+03], the effect of the IRV structure is actually a distributed query system. Collaborating ASes set up an IRV in order to speak authoritatively via query interfaces about the local network status and configuration. It will be through a query interface which the IRV provide access to dynamic and static data. Static data consist of community information, routing policy and data relating to peering. Dynamic data normally include the latest

routing tables, previousy obtained BGP route announcements together with a description of advertised routes.

The deployed IRV server is a specialized machine or couple of machines in the AS. The IRV must offer an interface by which outside entities are able to query routing data. Taking into account the threats against interdomain routing, the IRV must authenticate queries to avoid unauthorized access to sensitive information, such as enforce access control over routing information. Responses must be authenticated to avoid forgery. Confidentiality will also be preserved.

Queries and responses could be authenticated by using digital signatures. This method needs distributing some public keys, and might take significant computational resources [KLMS00]. Note, that these kinds of costs might be decreased by caching and later reusing regularly employed requests and responses [KLMS00]. Caches must be very carefully designed, because incorrect implementations could possibly expose vulnerabilities the system to replay attacks. The timeliness bounds must be set up and enforced on cached requests and responses.

The computational costs related to digital signatures can certainly put a significant burden on the currently resource-limited routers [GAG+03]. Differing from the BGP or the IRR, the validating entity in the IRV is in one on one communication with the relevant AS. Consequently, the current security protocols, including IPsec or TLS may be used to determine long-term security links. It is possible that many requests and responses are exchanged, therefore these associations are expected to be maintained over long periods.

Based on Kent et al's work [KLMS00], to view BGP UPDATE messages obtained from neighboring BGP speakers, an IRV server can easily set up I-BGP sessions with all the current border routers of the AS. All of the BGP messages obtained from peer ASes are also provided to the IRV. *Figure 3.4* demonstrates this. Dark lines are I-BGP sessions propagating routing information obtained from routers D, E, F, and G. The IRV is actually allowed to establish, through the identity of its I-BGP peer, the communicating AS and foreign BGP speaker which spread the message to the local AS. When a router (in this case, A) has BGP sessions with more than one routers (in this case, D and E), it may be required to configure the E-BGP listener regionally to employ private community fields or additional path features to point to the IRV the identity of the outside speaker.

As said in [KLMS00], malicious/compromised routers are able to manipulate AS-local IRV services simply by randomly omitting, slowing down, or perhaps

modifying I-BGP messages. However, protection of an AS from its own routers is clearly beyond the domain of the IRV. We consider that ASes will implement extra procedures to identify and disable defective or compromised routers.

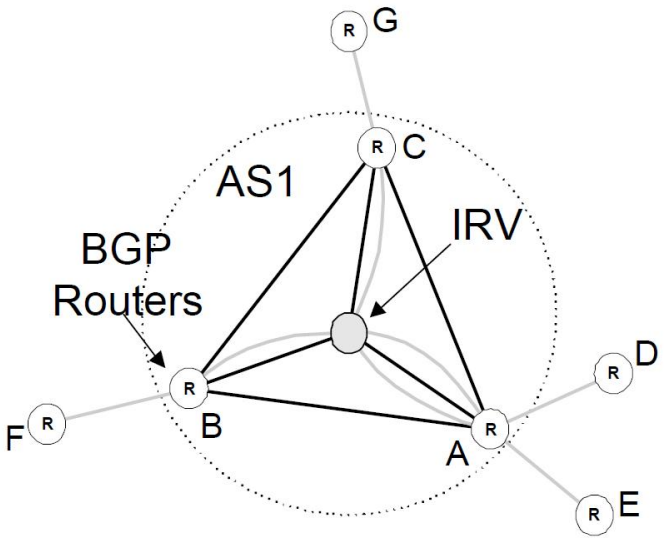


Figure 3.4: Dynamic Data acquisition by the IRV [KLMS00]

Keeping current information about BGP UPDATE messages delivered to outside ASes is very complicated. Particularly as there is no suitable method to intercept outbound E-BGP sessions which merely traverse a primary physical link between the E-BGP speaker and listener [GAG+03]. For every single outbound E-BGP session, the IRV-BGP designers proposed that setting up the communicating border router to determine another outbound E-BGP session using the IRV, configured to distribute a similar data as the corresponding session with the remote AS. The gray lines in *Figure 3.4* indicate this. These kinds of border routers, which perform as BGP speakers in multiple BGP sessions, set up multiple BGP sessions together with the IRV.

3.2.3 Key exchange between IRV and border router in AS

Now we are going to discuss how secret keys exchange between the IRV and border routers in our system. As we know the Diffie-Hellman key exchange [DH76] does not provide authentication, but that is not the problem in our system because IRV-BGP offer appropriate authentication mechanisms for border routers. Furthermore, it is vulnerable to "man in the middle attacks". However, the EKE system

Table 3.1: Notation

BR	System principal	Border Router
IRV	System principal	Inter-Domain Routing Validor
P	The password-liked secret key	often used as a key
R_{BR}, R_{IRV}	random number	generates by BR or IRV
α,β	System Parameters	for key exchange

[BM92] can solve both of these problems. The following protocol which provides both Diffie-Hellman key exchange and authentication comes from the EKE system. We modified the EKE system as part of our system to exchange secret keys between IRV and border routers.

In the following keys exchange we will use the notations defined in Table 3.1.

1. A BR picks a random number R_{BR} and calculates $P(\alpha^{R_{BR}(\text{mod }\beta)})$ to the IRV.

   The BR sends,

   $$BR, P(\alpha^{R_{BR}}(\text{mod}\beta)) \qquad (PDH.1)$$

   to the IRV; note that border router name (such as router number) is sent in the clear.

2. The IRV picks a random number R_{IRV} and calculates $\alpha^{R_{IRV}}(\text{mod } \beta)$. The IRV also uses the shared key P to decrypt $P(\alpha^{R_{BR}}(\text{mod } \beta))$, and calculates

   $$(\alpha^{R_{BR}R_{IRV}}(\text{mod } \beta))$$

   The Session key K is derived from this value. Finally, a random challenge $challenge_{IRV}$ is generated.

   The IRV transmits

   $$P(\alpha^{R_{BR}}(\text{mod}\beta)), K(challenge_{IRV}) \qquad (PDH.2)$$

3. The BR uses P to decrypt $P(\alpha^{R_{IRV}} (\text{mod } \beta))$. From this, K is calculated; it in turn is used to decrypt $K(challenge_{IRV})$. The BR then generates its own random challenge $K(challenge_{BR})$.

   The BR sends

$$K(challenge_{BR}, challenge_{IRV}) \qquad (PDH.3)$$

4. The IRV decrypts $K(challenge_{BR}, challenge_{IRV})$, and verifies that $challenge_{IRV}$ was echoed correctly.

   IRV sends

$$K(challenge_{BR}) \qquad (PDH.4)$$

5. The BR decrypts $K(challenge_{BR})$ to obtain $challenge_{BR}$ and verifies that it matches the original.

It is possible to omit encryption of one of the exponentials. For example, in the protocol shown above, the message $(PDH.1)$ could be replaced by

$$BR,\ \alpha^{R_{BR}} (\text{mod } \beta)$$

An attacker will not be able to decode the response from the IRV to get the key K. On the other hand, the enemy cannot respond to the challenge in the message $(PDH.2)$ without knowing the true value of R_{BR}. Thus far, we have said nothing about how to choose α and β. We will discuss this in following subsection.

Choosing α and β

At this point, we have discussed very little about how to precisely choose α as well as β for the key exchange. There are a variety of options, offering a range of tradeoffs between cost and security.

Even though there are a number of potential options for the modulus, reasonably large prime values of β will be more secure [Odl84]. Moreover, it is suitable that α be considered a primitive root of the field GF(β). In case we select β such that

$$\beta = 2p + 1$$

for some prime p, there are $(\beta - 1)/2 = p$ such values, therefore, they are really easy to find.

The security assumption is based on the hardness of solving the discrete logarithm problem. Our basic problem, when determining the way the BR and the IRV find out which value of α and β to use, would be to prevent leaking information. As known, we definitely cannot transmit P(β), testing a random value for primarily is very simple. We try to make α and β fixed and public. Consequently, there is no threat of information leakage or partition attacks. To keep security, β must be very large, which often makes the exponentiation operations expensive.

Some compromise within the length of the modulus may be possible. LaMacchia and Odlyzko [LO91] recommend 1000-bit values. They are assuming that the discrete logarithm problems are available to the attacker. In our system, the password **P** is required to encrypt these values, it is not possible to help essay a discrete logarithm computation except for all possible guesses of **P**. It is easy to select a size for β sufficient to make guessing attacks for too expensive.

An additional consideration inclines one towards a larger modulus. The border router's password might be compromised, recorded power of α will be accessible to the adversary. When computing discrete logarithms is solved, previous conversations will be authorized to be read. In case a large modulus value is used, all such conversation will remain secure.

Size requirements for β are produced from a desire to avoid calculations of discrete logarithms in the field GF(β). The existing best algorithms for this kind of calculations all need massive amounts of precalculation. When a different β is used each time, an attacker cannot create tables in advance; therefore, a much smaller modulus can be employed. For that reason, we recommend the IRV generate random values of β and α, and transmit them in cleartext during the initial exchange. There is little security risk associated with an attacker learning these values; the only problem would be with cut-and-paste attacks. And in many cases this risk is minimal when BR performs certain checks to defend against easily-solvable choices: that β should indeed be prime, which is large enough, that $\beta - 1$ have at least one large prime factor (to guard against Pohlig and Hellman's algorithm [PH78]), and that α is a primitive root of GF(β). The second two conditions are related. We must know the factorization of $\beta - 1$ to be able to verify α. Requiring that β be of

the form kp+1, where p is prime and k a very small integer, handles both issues. P is system-wide parameter.

Until now we have said nothing about choosing α. But if a suitable value of β is chosen, finding primitive roots is very easy. We can examine the integers starting 2, the density of primitive roots guarantees that one will be found quite quickly.

3.3 The SDS Structure

We now proceed to describe our main SDS setup steps. *Figure 3.5* illustrates how SDS works.

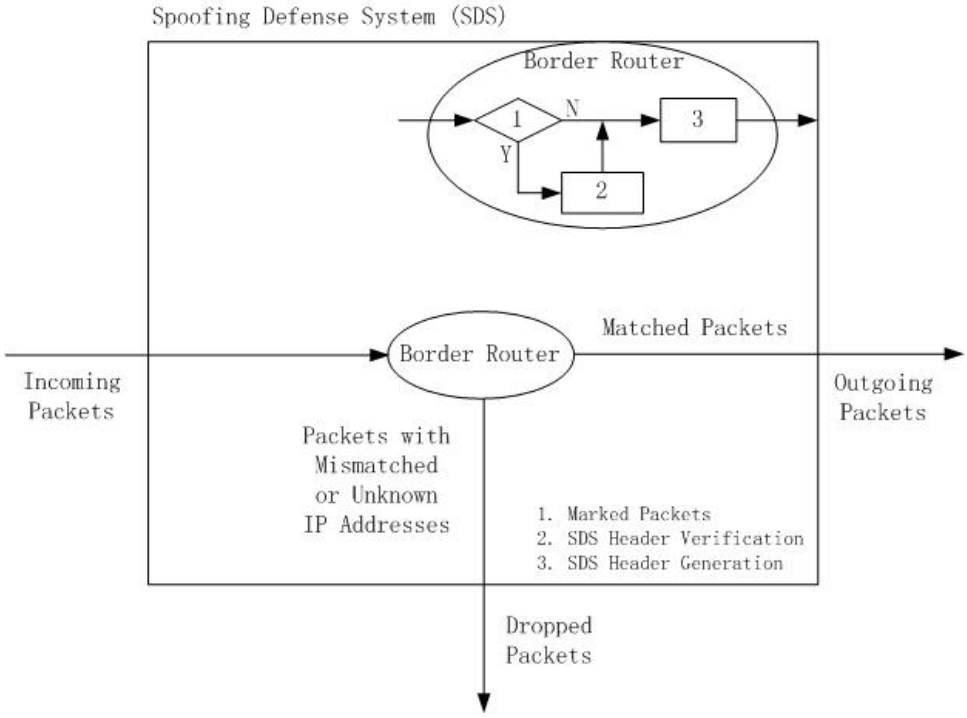


Figure 3.5: The system structure

Step 1 **System Setup**

In an AS each Border Router (BR) shares a password like key P with the IRV, this key P relates to the BR's ID and IP address. Both sides (IRV and BR) knowing P, can retrieve the power of α, and calculate the session key K. ASN_i and ASN_{i+1} share the key K via a secure communication channel, such as IPSec, which is used for BGP message authentication to IRV.

Step 2 **MAC Generation**

The border router of ASN_i computes *MAC (src, dst, len, IP ID,* ASN_i*,* ASN_{i+1}*)* using the secret key shared between ASN_i and ASN_{i+1}. Then it stamps the MAC into the IP opinion field as part of IP header in the packet which will be sent to the ASN_{i+1}. Note: The MAC value is computed only once, the following packets simply add the same MAC value into the IP option field till communication finishes.

Step 3 **MAC Verification**

The border router of ASN_{i+1} verifies the MAC in the IP opinion field of the packet received from ASN_i. The border router will replace the MAC value in the IP option field. Then the new packet will be sent to the next AS on the path. Note: The first time verification needs recomputation of the MAC value.

---

if MAC value is matched **then**
  repeat Step 2
  until communication finishes
else
  discard the incoming packet
end if

Chapter 4

Implementation and Experiment

This chapter presents an overview benefits of our SDS. Then we display an analysis of SDS benefits. The goals are presented in threefold: firstly, the implementation of our system in a real world scenario is demonstrated. Secondly, the experiment is described in details. Finally, we detail a prototype implementation of our SDS.

4.1 Analysis of the benefits of our SDS

This section proposes to show the benefits of our SDS. Basically, it possesses the following benefits:

1. The SDS header is unforgeable;
2. The SDS easily cooperates with other defence mechanisms;
3. The SDS header can be efficiently generated and verified at packet forwarding time;
4. The SDS has secure and automatic key distribution;
5. The SDS is easy to implement;
6. The SDS has low processing and no bandwidth overhead;
7. There are no inherent security flaws in our SDS;
8. Internal topologies of the ISPs are not revealed to our SDS;
9. The SDS can limit the attacker's ability or locate an attacker.

We use an efficient symmetric key MAC (UMAC) [Kro06] in our SDS. The UMAC with 128-bit keys is computationally infeasible to break [Kro06]. In spoofing

Table 4.1: Cooperative consideration

System	Can work with	Cannot work with
SPM	*	
Passport		*
DPF	*	
SAVE	*	
IDPF	*	
BASE	*	
Pi and StackPi	*	

attacks, the attacker may try to guess a valid SDS header by sending packets. Because our SDS header is a 64-bit MAC value, the attacker has to send at least 2^{63} packets to guess the correct one. The time that the attacker takes exceeds the period of UMAC renewal.

It is obviously easy to use cooperatively with other defence mechanisms such as SPM [BBL05], because we use the IP option field when most of other methods are using the IP ID field. *Figure 3.3* shows the possible header of a SDS. Table 4.1 shows some defence systems which could work together with our system. We only discuss router based systems here, because our system is a router-based one. There is no problem cooperating with other IP spoofing defence systems.

A current domain needs to share a secret key with the previous domain in order to validate the SDS header sent from the previous domain. The key distribution process must be automated in order to be feasible. In our design, keys are exchanged within the IRV. Each border router will obtain this shared key by using the Diffie-Hellman key exchange protocol as we mentioned in Chapter 3. We assume that each IRV server domain has set up a secure communication channel with the next domain.

Key distribution is required in IRV-BGP as described in an earlier section. The IRV system is an *independent* system meaning it is possible to configure the routing system to avoid malware hosts injecting routing packets. In particular, two adjacent routers authenticate each other ahead of their established communication. When this type of communication is established, routers are able to forward and process routing packets along with the highest priority. Frequent data traffic are unable to congest the routing channel.

For better security, domains need to periodically change the secret keys. This requires periodic update of secret key pairs, which can happen at a large time scale. The SDS accompanies a BGP UPDATE message to renew its secret key pairs. Rapid

rekeying prevents an attacker from replaying an SDS packet.

In our design, each transit domain can independently verify an SDS packet. This allows routers to drop packets with invalid SDS header as early as possible and prevents these packets from congesting downstream links. However, it has the side effect of binding a SDS to an AS path. The problem comes from the fact that in the process of routing convergence, the actual AS path a packet traverses may differ from the one in its SDS header that is obtained from the source domain's BGP table, and then the packet may be dropped by a transit domain. However, this limitation may not be a serious problem in practice, because even without the SDS there is no guarantee that a packet can reach its destination in case of a routing inconsistency. When there is no congestion, such packets can still arrive at their destinations.

The immediate application of a SDS is to filter unwanted traffic, because a SDS header shows the real source of a packet. Suppose Router BR_2 in *Figure 3.1* classifies packets from AS_{100} as attack traffic and block traffic from AS_{100}. Destination AS_{300} will never receive the spoofed packets. Note, attackers are not in AS_{100}, it means attackers can not launch neighbour IP spoofing attack. Reducing this kind of attack is the ISPs' responsibility.

Our model encourages incremental deployment and offers benefits for early adoption. In the event of a general deployment, the AS path in a SDS created by the source domain basically involves the domains that support SDS. The SDS enabled domains will verify the SDS header, and process legal packets as well as packets having invalid SDS headers dropped. Domains not having our SDS might ignore the SDS headers and forward the packets.

Early adopters of the SDS will be able to obtain the following benefits. First of all, a domain which deploys the SDS can easily prevent other domains from spoofing its source identifiers. Subsequently, it could identify almost any attack source from some other domain which also supports the SDS. Last, it could possibly avoid responsibility for attacks that declare it as the origin but are not in fact so. Widespread deployment is not a real possibility, however higher level deployment is certainly strongly recommended, e. g. in case AS level deploy SDS, the spoofing attacks will probably be restricted in the ISP level.

When our SDS received spoofing packets, an attacker can be located in the AS which deployed the SDS as this system is able to locate an attacker or attacker's zombies. Thus, an attacker will not risk mounting an attack. None of the host-based mechanisms such as IPSec can identify where an attacker is located. The details

will be given in Chapter 5.

4.2 Implementation of our SDS in a real system

Similarly to various network researchers, we experienced a serious problem when implementing applications in routers intended for testing. Router systems commonly are not open systems, in either the open-source or the open-API sense. During this section we show an eXtensible Open Router Platform (XORP) [HHK02] over which we are establish our system. Primary goals of XORP are extensibility, performance and robustness. As described in [HHK02], XORP is really a research tool as well as a dependable deployment system, therefore facilitating the conversion of new thoughts from the lab towards the actual world.

XORP is an open-source software router system, operating on commodity hardware, which is workable as a research design creation platform. According to [HHK02], the application structure of XORP was designed with extensibility as a primary purpose and should really allow experimental protocol deployment with minimum risk to current services implementing on that router. We are able to access the router software and then share a common system for testing our implementation on a system where real traffic circumstances exist. This is why we selected XORP as our system experimentation deployment environment. The next subsection highlights XORP.

4.2.1 XORP

XORP is definitely an open source routing system which is certainly developed for extensibility from the beginning. XORP provides a unified platform to configure and implement the IPv4 and IPv6 routing protocols. We select XORP mainly because it may be the only open source system to provide integrated multicast functionality. XORP's modular structure permits us to create our new protocol, new elements and functionality, and much more important XORP has support for customized hardware and software forwarding.

XORP could be separated into two subsystems [HHK02, She09]. The higher-level ("user-level") subsystem contains the routing protocols, information bases and support processes. The lower-level subsystem operates inside an OPERATING-SYSTEM kernel, handles the forwarding path, and offers APIs for the higher level to reach.

The SDS operates at a lower level which applies the Click modular router [KMC+00, MKJK99, Koh06, Koh00], a modular extensible toolkit designed for packet processing on normal PCs. For our experimentation, we do not require usage of the forwarding path, a normal OPERATING-SYSTEM just like Linux kernel as lower level might be enough.

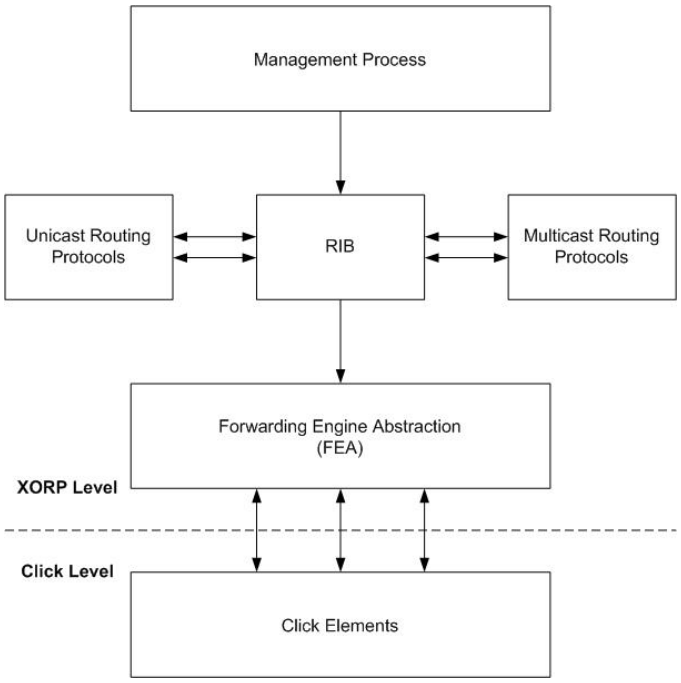


Figure 4.1: XORP high-level processes[1]

Figure 4.1 shows how a XORP router's user-level processes and Click forwarding path fit together. Based on [HHK02], there are four core processes: the *router manager*, the *finder*, the *routing information base (RIB)*, and the *forwarding engine abstraction (FEA)*.

The *router manager* process manages the router. It keeps router configuration information, including routing protocols when requested by the configuration, and restarts failed processes as needed.

The *finder* process which is a part of XORP management processes establishes mappings between abstracted application requests, including the number of interfaces. When an application would like to make an IPC call, it consults the finder to determine ways to achieve it. Additionally, the finder might advise applications to update contact information which makes application requests completed at run-time.

[1]The XOPR High-level processes is summarized from [HHK02, She09].

The *RIB* process obtains routes through the routing processes. Routes need to be spread into the forwarding path as RIB arbitrates, or redistributed to other routing processes.

The forwarding path will be handled through the *FEA* process. The FEA abstracts the main points of the way the forwarding path of the router is executed. The routing processes are usually uncertain as to whether or not the forwarding plane is Click based, a conventional UNIX kernel, or maybe other techniques. The FEA deals with the networking interfaces together with forwarding table in the router, and offers information to routing processes regarding the interface attributes as well as events occurring on interfaces. During the next subsection we will present the Click modular router in details.

4.2.2 Click

Click is known as a new software architecture. It is intended for building manageable and configurable routers [KMC+00, Koh06, Koh00, KMP00]. A Click router is usually constructed from packet processing modules named elements which execute simple router functions including queueing, scheduling, packet classification, and interfacing with network devices. Entire configurations are created by linking elements into a graph. For example pull processing, which models packet flow driven by transmitting interfaces, and flow-based router context, which assists an element discover other useful elements.

The Click configurations are modular. Basically the IP router has 16 elements on the forwarding path and is easy to extend by adding additional elements. Based on [KMC+00], on commodity PC hardware running Linux, the Click IP router can forward 64-byte packets at 73,000 packets per second, just 10% slower than Linux alone.

The components constitute Click routers called *elements*, which are plugged together into *configurations* [Koh06]. Elements can create packets, modify them, classify them into different paths, and so forth. There is an example, elements include *"from-Device(eth1)"*, which reads and emits packets from network device **eth1**, and *"Discard"*, which drops any packets it receives. Here is a simple router configuration file using those elements:

```
FromDevice(eth1) -> Discard;
```

In the running router, every element is known as a C++ object and connections are pointers to elements [MKJK99]. The overhead of passing a packet together with a connection is usually a single virtual function call. The most significant attributes of the element are:

1. *Element class.* Similar to objects in an object-oriented program, each individual element contains a class which establishes its behavior.

2. *Input and output ports.* Ports are classified as the endpoints connected with links between elements. An element provide any number of input or output ports, which often can possess different definitions.

3. *Configuration string.* Various element classes support extra arguments, accustomed to initialize element state and element behavior. The configuration string contains these arguments.

Figure 4.2 indicates the way we diagram these kinds of attributes for the singular element, Tee(2). 'Tee' is the element class; a Tee duplicates each packet it gets from its single input port, delivering one copy to each output port. (The packet data is not really copied: Click packets are actually copy-on-write.) Configuration strings are generally encapsulated in parentheses: the '2' in 'Tee(2)' is a configuration string which Tee interprets to be a request for two outputs.

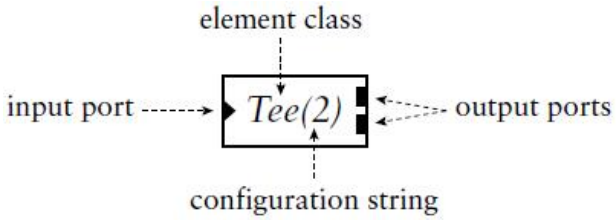


Figure 4.2: A sample element. Triangular ports are inputs and rectangular ports are outputs. [KMC+00]

According to [KMC+00], every single action executed by using a Click router software will be encapsulated in an element, from device reading and writing to queueing, routing table queries, and counting packets. The user decides exactly what a Click router does by selecting the elements being utilized as well as links among them.

A Click driver can easily operate in the Linux kernel, and/or at user level on any kind of Unix-like OPERATING-SYSTEM. Many element source files could

be compiled for possible drivers. The Click kernel configurations can certainly operate at or even at the boundaries of standard PC buses [KMC+00, KMC02, CM01, HPK01]. As an illustration, an optimized Click IP router can send 740, 000 minimum-size packets a second over Gigabit Ethernet on a 1.6 GHz Athlon MP with 64-bit/66 MHz PCI [KMC02].

Figure 4.3 illustrates the most basic explanations. A Click setting which numbers all of the packets arriving on device *eth0*, and then throws those packets away. The *"PROMISC true"* argument implies that the *eth0* device needs to be set in promiscuous mode. This particular setting works similarly nicely at a user level or simply inside a patched Linux. In a patched kernel, it sorts packets straightaway in the appropriate device queues, preventing any packet copies and scheduling holdup. With certainty network cards, a *PollDevice* element can be utilized in place of *FromDevice*. As pointed in [MR97], this increases operation at high input rates using polling, which removes all interrupt as well as programmed-I/O overhead together with any received livelock.

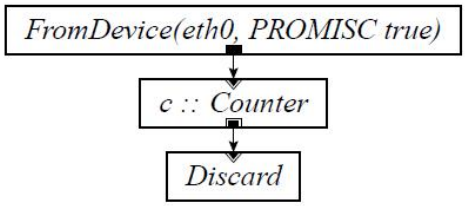


Figure 4.3: Counting packets from the *eth0* device. [KMC+00]

4.3 Experiment

This section details a prototype implementation of our SDS. We apply much of the features of our SDS employing XORP [HHK02] together with Click modular router [KMC+00, MKJK99, Koh06, Koh00]. *Figure 4.4* indicates the structure of the implementation. We customize XORP to be able to piggyback the Diffie-Hellman key exchange protocol inside BGP, and adjust elements in Click to assist SDS header stamping and verification. We additionally improve XORP to communicate with Click with the */click* file system (See more information in Appendix C ip-static.click).

The matching shared secret key is actually created and delivered to Click with the set key interface in *Figure 4.4*. We customize *RibIpcHandler* to be able to

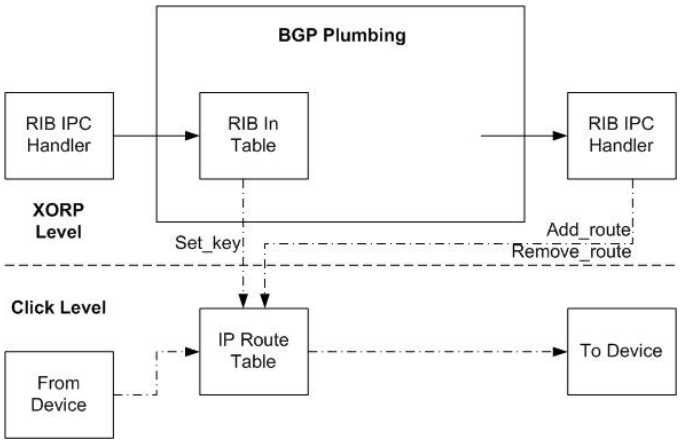


Figure 4.4: How our SDS is implemented using Click and XORP.

update the Click routing table together with the SDS associated information using the add route and remove route interfaces in *Figure 4.4*. The SDS associated information consists of AS paths and prefixes to origin AS mapping. In our recent implementation, a new Diffie-Hellman value is created at XORP's startup time. This element is required to be expanded with supporting regular re-keying, and additional key distribution using BGP.

We adjust the IPRouteTable element in Click so that it obtains shared secret keys coming from XORP and requests generate *sds*() or validate *sds*() in its *push*() approach to stamp or verify SDS headers. We apply Click priority scheduler to manage normal and demoted traffic. We use priority queuing rather then weighted fair queuing in order to highlight major benefits of implementing our SDS. The ARP queue implies that link-local ARP packets have highest forwarding priority.

We utilize three PCs in our research laboratory in order to measure the computational overhead of SDS header stamping and verification. One PC is employed as a router, linking a packet generator PC along with a sink PC. The router has a Pentium-D 3.4GHz CPU, 2GB memory, plus two Broadcom NetXtreme 57xx Gigabit Ethernet Controllers. The packet generator as well as sink also are built with Pentium-D 3.4GHz CPU, 2GB memory plus Broadcom NetXtreme 57xx Gigabit Ethernet Controller. We evaluate the throughput SDS stamping and verification.

In order to determine the throughput of the SDS header stamping, we allow a packet generator to send out minimum sized packets (40 bytes TCP/IP headers) at several input rates. Our experiments suppose legacy hosts, and also the router PC inserts SDS headers inside the packets. The sink PC calculates the output rate.

Ten million packets are provided for every experiment. The following basic code, UDPGen.click is employed for the purpose of creating examining packets together with UDPcount.click is operating on sink PC to be able to count the number of UDP packets obtained on port 1234 in this experiment.

```
// UDPGen($device, $rate, $limit, $seth, $sip, $sport, $deth, $dip,
// $dport);
//
//  $device     name of device to generate traffic on
//  $rate       rate to generate traffic (packets/s)
//  $limit      total number of packets to send
//  $size       bytes per packet
//  $seth       source eth addr
//  $sip        source ip addr
//  $sport      source port
//  $deth       destination eth addr
//  $dip        destination ip addr
//  $dport      destination port
 u :: UDPGen(eth1, 65000, 10000000, 40,
        00:0E:0C:BB:04, 10.10.10.1, 1234,
        00:0E:0C:BB:05, 10.10.20.1, 1234);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// UDPcount
 AddressInfo(the_interface 10.10.20.1 00:0E:0C:BB:05);
 classifier :: Classifier(12/0800 /* IP packets */,
                          12/0806 20/0001 /* ARP requests */,
                          - /* everything else */);
 ip_classifier :: IPClassifier(dst udp port 1234 /* relevant UDP
                               packets */, - /* everything else */);
 in_device :: PollDevice(the_interface);
 out :: Queue(200) -> ToDevice(the_interface);
 to_host :: ToHost;
 ctr :: Counter /* or AverageCounter */;
 in_device -> classifier
           -> CheckIPHeader(14, CHECKSUM false)
```

```
            -> ip_classifier
            -> ctr
            -> Discard;
 classifier[1] -> ARPResponder(the_interface) -> out;
 classifier[2] -> to_host;
 ip_classifier[1] -> to_host;
```

In the same way, in order to estimate the throughput for SDS header verification, we allow a packet generator send out minimal sized packets together with pre-specified SDS headers at several rates through 2 AS hops, make use of the router PC to be able to verify SDS headers, and determine the result rates at the sink PC.

SDS-enabled routers additionally exchange Diffie-Hellman keys over the routing plane. The cryptographic functions consist of producing Diffie-Hellman values as well as calculating shared secret keys from Diffie-Hellman values. Either Diffie-Hellman values or shared secret keys usually are created using modular exponentiation.

Re-keying needs to occur infrequently such as once a week. When an AS itself re-keys, it must obtain a shared secret key with other AS. There are actually fewer than 30K ASes observed in BGP routing tables depending on data from RouteViews [Rou09]. It will take fewer than three minutes to generate almost all shared keys on the router PC. In cases where we consider all ASes re-keys at random within a period of a week, and then typically, an AS may possibly obtain fresh Diffie-Hellman values each minute. Based on Liu's paper [LLYW08] we possibly spend 17 ms to calculate the shared secret keys.

SDS keeps per-AS key details. A router maintains a shared secret key for each AS for SDS header stamping and verification. A MAC calculation generally demands the initialization of a key context (refer to Chapter 3). It is desired that a router initializes while establishing the key context for the purpose of fast processing. The length of a key context varies according to the particular MAC. Our system implementation runs on the UMAC key context which occupies 128 bits per key. Determined by our experiment, there is less than 12MB memory to keep the distributed keys together with their key contexts.

Whenever an AS re-keys, an alternative AS may require two different keys for the purpose of SDS stamping and verification: one key created using the previous Diffie-Hellman value, and then the other using a completely new value. This involves extra memory. According to Liu's experiment [LLYW08], the average re-keying rate

Table 4.2: Equivalent MAC security level of well-known public key signature schemes. [LV00]

	Security	Sig. Size	Signing	Verification
RSA-512	60-bit	64 bytes	512 μs	40 μs
RSA-1024	72-bit	128 byte	2214 μs	102 μs
DSA-512	65-bit	40 byte	368 μs	443 μs
ECDSA-160	78-bit	40 byte	300 μs	1400 μs

at all ASes is usually only 3 key pairs per minute. In the event that we suppose it will take no more than an hour for BGP to converge, consuming 20KB memory overhead.

SDS's header, as well as processing overheads, is actually inherent to cryptography-based security mechanisms. For the purpose of evaluation, we listed the header and processing overhead of well-known public key signatures that offer a similar degree of security as one 64-bit MAC in Table 4.2. The particular tests are accomplished with the OpenSSL speed analyze on the router PC, additionally, the security degrees are determined based on Lenstra and Verheul's experiment [LV00].

Chapter 5

Results and Evaluation

This chapter investigates our SDS security and experiment results, and how this SDS has been used in the real systems. It is divided into four parts. The system security is considered under different kinds of attacks in Section 5.1. In order to indicate the performance and feasibility of our system, some real deployment experiment results are analyzed in Section 5.2. Section 5.3 shows how to use modeling to study the adoptability and deployment benefits of our system. In the last section, the SDS characteristics will be analyzed and compared with various spoofing defence mechanisms.

5.1 Security

In this section, we discuss different attackers based on their locations which might attack our system. According to [CDPZ06], attackers can be classified as Host Attacker, Monitor Attacker and Router Attacker.

Host Attacker: In order to spoof source addresses, a host attacker might possibly make an attempt to break the MAC that is certainly the center of the SDS. Nevertheless our design implements a standard MAC scheme with 128-bit keys, that is definitely computationally infeasible to break. The attacker will probably preferably instead seek to pretend to be a valid SDS header by sending packets. Considering an SDS header carries a 64-bit destination MAC value, an attacker would certainly expect to send no less than 2^{63} packets to help guess just one correctly, however the time required to send those packets is greater than the time period for which the SDS header is valid. The long-term key distributed via BGP may have advanced during the meantime. This kind of attack would also indicate a clear anomaly by using a great number of invalid SDS headers.

The attacker may possibly attempt to guess the 64-bit MAC value by sending

TTL terminated packets, together with the echoed IP header and SDS header to look at whether or not a guess is demoted. However this again is actually infeasible since ICMP messages are usually rate restricted.

Monitor Attacker: The eavesdropper might discover valid SDS headers but cannot easily send them to a different path just because an SDS header is bound to one particular AS path, it means, its MACs will probably be identified as invalid when sent using another AS path. Therefore, packets transferred to some other paths will be demoted on the paths to destinations, and then can not compete for bandwidth with packets having correct MACs.

Router Attacker: Packets copied by routers within the forwarding path of a source may possibly arrive at a destination without having been discarded. Nevertheless, in this instance, routing is compromised, and the SDS's security is bound to routing security. An AS spoofed by a router on its forwarding path must choose a different path.

When an attacker compromises an AS, it might utilize the AS's keys at other, unprotected places in the network to forge a SDS header which spoofs the AS's addresses. However this approach only implicates the compromised AS, though possibly not in other parts of the network. Moreover, a compromised router is able to spoof packets from many other addresses inside its AS, nevertheless this once again implicates the AS and might negatively threaten its traffic.

We realize that even though two ASes share a secret key, they can definitely not make use of this key to spoof each other's addresses at other ASes, since different ASes employ individual keys shared with these two ASes respectively to verify their addresses.

The security of the SDS is bound to routing security. We depend upon routing to distribute the correct public Diffie-Hellman values over ASes. Through the Diffie-Hellman structure, we do not rely on routers to maintain the public values secret from attackers, mainly because it is computationally infeasible to obtain the keys given only the public values. However, in the event that an attacker can successfully hijack a prefix announcement as well as restore the Diffie-Hellman value, it can both acquire packets for the specific prefix, and also deliver packets almost as if they were originated from that prefix.

5.2 Performance and Feasibility of SDS

A comprehensive experimental study is required to examine the performance and feasibility of our design. As a first step, we provide a preliminary analysis to assess the feasibility of the design. Our design has two primary sources of cost:

1. The computational cost for the SDS header generation and validation.

2. Communication and computational cost for keys exchange.

First, we estimate the computational cost for SDS header generation and validation. With our design, the source domain needs to compute a MAC to generate an SDS header. A transit domain or the destination domain needs to compute one MAC to validate the first SDS packet header and compute another MAC to replace the old valid one. MAC functions such as UMAC can be computed efficiently on modern processors. In a router we expect that the special hardware can perform MAC generation at a much higher speed.

Second, we analyse the communication and computational cost for key exchange. With BGP, two IRV servers need to set up a secure communication channel first. When this channel has been established, the secret key used in the SDS header will be exchanged through this channel, which only happens at system set up stage and system update stage. Therefore, both the communication and computation cost is affordable. Note, in this part, we do not count computational cost of BGP authentication, and the cost of IRV servers' secure communication channel establishment between each other.

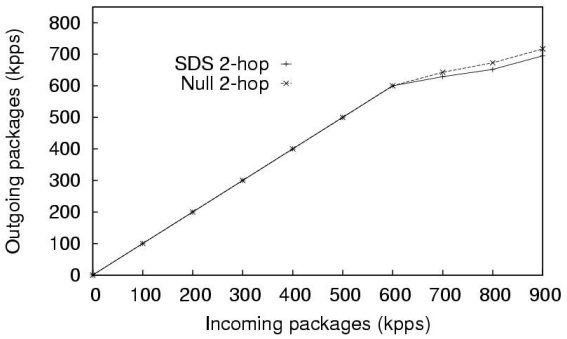


Figure 5.1: The SDS header stamping in 2 hops

We implement some of the features of SDS using Click [KMC+00]. We modify the *IPRouteTable* element in Click, so it receives shared secret key from a key

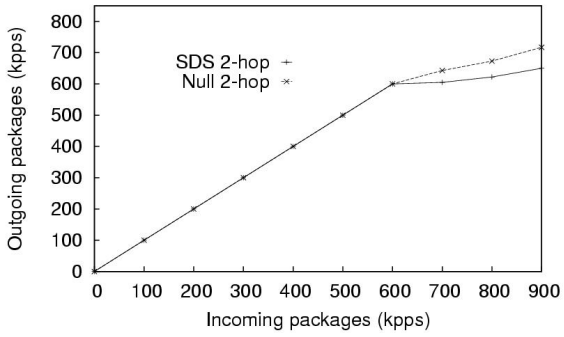


Figure 5.2: The SDS header verifying in 2 hops

generator function which is the C program to generate keys. Note, the purpose of our experiment is testing the SDS header processing overhead. So we do not consider the BGP implementation and Diffie-Hellman key exchange cost in our experiment. The element calls $generate-sds()$ or $verify-sds()$ to stamp or verify SDS headers. We also use UMAC because of its speed as we described in Chapter 3, Subsection 3.2.1. We used three PCs in our lab to measure the computational overhead of SDS header stamping and verification. As we mentioned in Chapter 4, one PC is used as a router, connecting a packet generator PC and a sink PC. To test the SDS header stamping, we let the packet generator send 40-byte UDP packet at various rates. The sink PC measures the output rate. The experiment is continuing, in this section we only show 2 hops MAC computations in *Figure 5.1* and *Figure 5.2*. We predict that the SDS header stamping will perform the same in N hops.

Figure 5.1 together with *Figure 5.2* demonstrate the SDS header stamping and verification throughput, along with Click null forwarding throughput for packets of the similar size (40-byte). The SDS header verification throughput matches properly with Click's null forwarding throughput, because it only involves one MAC computation. The verification throughput changes from 549 kpps to 636 kpps for SDS headers of two AS hops. The slight decrease is mainly a result of the increase in packet length or router's hardware limitations, definitely not the MAC computation.

5.3 Modeling

In this section we intend to take advantage of modeling to examine the adoptability benefits of SDS. We are interested in this study because SDS provides much more motivation for adoption than with ingress/egress filtering. The AS which deploys

ingress/egress filtering can still get its addresses spoofed by other regions of the network.

To look at whether or not our SDS offers better security benefit, we use the framework introduced in [CDPZ06] to evaluate its adoptability together with ingress /egress filtering.

The adoptability model simulates the adopting collection of every AS as a result of iterations. With each one iteration, an AS α which has not implemented a spoof defence mechanism investigates its security advantage before and after adopting. In cases when the benefit is higher than its cost threshold, α adopts the mechanism. The iterations cease when no more ASes adopt the mechanism. The important threshold is the primary cost threshold that triggers full deployment. It calculates the security advantage a spoof defence mechanism provides to an adopter.

For an AS α which takes into consideration a spoofing prevention mechanism, we express the security benefit as the average probability that the attacker cannot spoof α's addresses in the network. To be able to calculate this particular probability, an AS α iterates via every single different AS β, together with investigations into cases where a malware adversary, at an AS M, can possibly spoof its addresses at β. We determine the average security S within the router through the security indicator $E(M, \beta) \in \{0, 1\}$ which is set to 1 when M cannot spoof α, and 0 otherwise. The probability that an adversary cannot spoof α at β is calculated as follow,

$$S = \sum_{M} E(M, \beta) P(M) \tag{5.1}$$

where $P(M)$ is the probability that M is malicious. Using the formula (5.1), S might take a value in {0,1} where 1 implies that spoofing cannot happen, and 0 otherwise. The security benefit of α, denoted by F is the weighted average of (5.1) within all βs. That is,

$$F = \frac{\sum_{\beta} S \cdot \omega_{\beta}}{\sum_{\beta} \omega_{\beta}} \tag{5.2}$$

The weight ω_{β} models the case that an AS α transmits various degrees of traffic to different AS βs. Intuitively, the more traffic α sends to β, the more important

it is that α's addresses are certainly not spoofed at β. The weight $P(M)$ models several security degrees of different ASes.

An AS α computes the security benefit F' right after its adoption, as well as current security benefit F prior to its adoption, and also purposes the variation $\Delta F = F' - F$ as its motivation for usage. At each iteration, α compares ΔF along with a cost metric c. When $\Delta F > c$, it adopts the spoof prevention mechanism.

We compute and compare the security benefit F for ingress/egress filtering as well as our system. In ingress/egress filtering, an AS α which deploys ingress filtering is able to filter spoofed traffic coming from hosts in its network or from its client ASes together with incoming traffic that spoofs unique addresses [Fer00]. Before an AS α deploys ingress/egress filtering, every malicious node can spoof its address space at an AS β. Soon after α adopts ingress/egress filtering, an adversary can still spoof α at β, except in cases where the attacker's traffic to β is sent via α, or $\beta = \alpha$. As an illustration, in *Figure 5.3*, in case M' is the malicious node, once α deploys ingress/egress filtering, M' cannot spoof α at β.

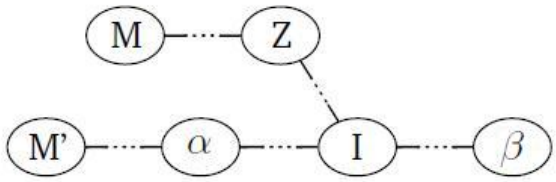


Figure 5.3: A sample topology

If an AS α does not deploy any SDS, any malicious node can easily spoof α's addresses located at an AS β. When an AS α deploys our SDS, its security benefit is determined by the attacker model. In the case a malicious node is known as a **Host attacker**, the malicious node cannot spoof α at β when there is more than one upgraded AS on the path from the malicious node to β. For this reason that the malicious node cannot spoof a valid SDS header.

If the malicious node is known as a **Monitor attacker**, it is able to sniff α's traffic sent by itself, and collude with some other compromised Host attackers to replay sniffed traffic. This replayed traffic by the compromised Host attacker is going to be discarded when the route from the Host attacker to β along with the route from α to β differ by at least one link whose end node is usually an upgraded AS [CDPZ06]. As an example, in *Figure 5.3*, in cases where M is a colluding compromised Host attacker, as long as the path from M to I (including I) includes one AS which

deploys our SDS, M cannot spoof α at β without having to be discarded, no matter the monitor attacker's location. Due to the fact that the MAC value in any SDS header covers the previous incoming AS number.

The security benefit against **router attacker** threat is comparable to that against **monitor attacker**, with the exception that a router attacker can certainly replay sniffed packets at its own location. Under the router attacker threat, a router attacker on an AS α's forwarding path can constantly spoof α's addresses, before and/or after α adopts ingress/egress filtering [Fer00].

We observe that while assessing the security benefit for an SDS, assuming that β's spoofed traffic can be discarded at β, α considers its traffic not spoofable by M at β, because M's spoofed traffic is actually distinguishable from its authentic traffic.

5.4 Analysis

Previously we studied how the other spoofing defence mechanisms work, in this section we are going to analyze how different spoofing defence mechanisms identify spoofing packets, and what deployability issues they may have as well as our system. Several mechanisms can probably identify most of spoofing packets, but may not succeed in the real-world.

Most of spoofing defence mechanisms are able to detect spoofing packets, however they often cannot identify all spoofing packets. Table 5.1 illustrates how those defence mechanisms identify spoofing packets, as well as their limitations. Note, the efficacy we mentioned here is under the conditions stated in next paragraph. They may not reach such efficiency in real world deployment.

Even if spoofing defence mechanisms detect and discard a spoofing packet, additional spoofing packets may keep coming. There is no risk for an attacker unless it is located. As we mentioned in Chapter 4, our SDS can locate an attacker or an attacker's zombies. Table 5.2 shows how various spoofing defence mechanisms can locate an attacker. Compared with router-based mechanisms, no host-based mechanisms can locate the attacker. When using a host-based mechanism, there is no information contained in spoofing packet to expose an attacker's location. Our SDS can identify where an attacker is located. If the attacker is in a SDS-protected AS, the SDS can identify the attacker's location, the attacker will be located by the SDS enabled router in his respective AS.

Table 5.1: Analysis different spoofing defence mechanisms

Mechanism	Efficacy	Condition
BASE	over 90% identify	level 1 ASes deployment
DPF	over 90% identify	over 70% random deployment
Hop-count filtering	over 90% identify	spoofing packet with incorrect hop-count over Internet
Ingress/egress filtering	100% identify	full deployment
IDPF	over 80% identify	over 60% random deployment
IPSec	100% identify	every client deploy IPSec
Passport	100% identify	full deployment
SAVE	100% identify	full deployment
SDS	**100% identify**	require at least 2 ASes to deploy SDS
SPM	100% identify	only if the destination deploys SPM
StackPi	100% identify	full router deployment

Table 5.2: Summary of locating attackers

Mechanisms	Cannot Locate	Can Locate
A packet marking method with Bloom Filter		√
BASE		√
Deterministic Packet Marking (DPM)		√
Distributed Packet Filtering (DPF)		√
Hop-Count filtering		√
Ingress/engress filtering		√
Inter-Domain Packet Filtering (IDPF)		√
IP puzzle	√	
IPSec	√	
OS fingerprinting	√	
Passport		√
Peer-to-Peer Based Anti-Spoofing Method		√
Pi		√
Probabilistic Packet Marking (PPM)		√
Reverse Path Forwarding (RPF)		√
Router-Based Filtering (RBF)		√
SAVE		√
Spoofing Defence System (SDS)		√
Spoofing Prevention Method (SPM)		√
Pi/StaticPi		√
SYN cookies	√	
TCP probing	√	

Chapter 6

Conclusion and Future Work

During this chapter we focus on several feasibility questions regarding deployment of our system in actual conditions. Then we consider further research in the near future.

6.1 Discussion

- **Discard vs Demotion**
  Note that when compared with demotion, our SDS discards packets having incorrect MACs in intermediate ASes. One other design option is to demote these packets. We select discard over demote as discarding offers two primary advantages. First of all, invalid packets consume no more of the network's resources. Second, legacy ASes are not required to make configuration adjustments. The disadvantages may be to introduce unwanted packet loss at the time of routing convergence. However this small-scale loss rate may not affect TCP's overall performance.

- **Packet Fragmentation**
  Packets fragmented involved within the network will not have the valid SDS header. Our SDS discards almost all fragments, including at the destination AS. We are not really worried about this issue, simply because fragmentation by the network is actually discouraged, and it has been recently disabled by IPv6.

- **Inter-domain multicast**
  Our SDS basically features source authentication just for unicast traffic, since the origin of a duplicated multicast packet would not match its source address.

Routers need to implement different authentication systems for example [PCST01] in order to authenticate multicast traffic.

- **Our SDS on High Speed Routers**
  Typically the performance of our system implementation might be limited for high speed routers. Nevertheless, by using optimized hardware implementation, throughput of tens of gigabits per second could be achieved. The bottleneck of SDS header processing is the AES-based MAC calculation. There are several industrial hardware AES implementations which are able to encrypt at a speed of 40Gbps [Hel10].

6.2 Conclusion

Currently, most solutions which have been proposed to defend against IP spoofing are based on filtering packets, the IP source address and the incoming interface. In this thesis, we presented an alternative solution to the IP spoofing problem, using an efficient symmetric key message authentication code (UMAC) as its authentication information to verify that an IP packet has not been spoofed; this is a router based packet marking solution. The IRV architecture is used with the Border Gateway Protocol 4 (BGP) [RLH06] to validate its data and routing information relevant to an AS.

The SDS is an efficient secure system which is easily used collaboratively with other defence mechanisms. The approach relies on IRV-BGP update message to exchange its key for marking outgoing packets, and it effectively filters spoofing packets. Obviously, our SDS is efficient, secure and easy to use in collaboration with the other defence mechanisms. Different ASes border routers obtain shared keys via IRV servers in the ASes by using the Diffie-Hellman key exchange protocol. The IRV will manage the secret keys and exchange keys between different ASes via a secure communication channel such as using IPSec to establish the security communication channel between the IRV servers. SDS is deployable because the computational cost for our SDS header generation, validation and key exchange is affordable.

We continue to work on improving our system, and carry out simulation experiments. We are planning to prove the deployment performance of SDS using a test bed in the real world in the future. Key exchange and BGP implementation will be

consider in later tests.

The work in Chapter 3 has appeared in ISI 2010 [WXS10].

Bibliography

[BA03] Belenky, A. and Ansari, N. IP traceback with deterministic packet marking. IEEE Communications Letters, vol. 7, 2003, pp. 162–164

[BB05] Beverly, R. and Bauer, S. The Spoofer Project: Inferring the Extent of Source Address Filtering on the Internet. USENIX SRUTI 2005, 2005, pp. 53–59

[BBL05] Bremler-Barr, A. and Levy, H. Spoofing Prevention Method. In: IEEE INFOCOM. IEEE, 2005, pp. 536–547

[BCK96] Bellare, M., Canetti, R. and Krawczyk, H. Keying hash function for message authentication. In: Advances in Cryptology - Crypto'96, Lecture Notes in Computer Science (LNCS), vol. 1109. Springer-Verlag, 1996, pp. 1–15

[BCKR07] Bates, T., Chandra, R., Katz, D. and Rekhter, Y. Multiprotocol Extensions for BGP-4. Request for comments rfc4760, IETF, January 2007

[BCR08] Bhati, G., Chakraverti, A. K. and Ram, D. Detecting and Preventing IP Spoofed Attack by Cryptography. In: 2nd National Conference on Challenges and Opportunities in Information Technology (COIT), March 29, 2008, pp. 60–63

[Bel04] Bellovin, S. M. A Look Back at 'Security Problems in the TCP/IP Protocol Suite'. In: 20th Annual Computer Security Applications Conference (ACSAC'04), 2004, pp. 229–249

[Ber10] Bernstein, D. J. SYN cookies. http://cr.yp.to/syncookies.html, 1996, viewed: 2010

[BHK+99] Black, J., Halevi, S., Krawczyk, H., Krovetz, T. and Rogaway, P. UMAC: Fast and Secure Message Authentication. In: Crypto'96, Lecture Notes in Computer Science (LNCS). Springer-Verlag, 1999, pp. 216–233

[BM92] Bellovin, S. M. and Merritt, M. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: The IEEE Symposium on Research in Security and Privacy. IEEE, May 1992, pp. 72–84

[BQS09] Banday, M. T., Qadri, J. A. and Shah, N. A. Study of Botnets and Their Threats to Internet Security. Sprouts: Working Papers on Information Systems, vol. 9, no. 24, 2009

[Bra89a] Braden(ed.), R. Requirements for Internet Hosts - Application and Support. Request for comments rfc 1123, IETF, October 1989

[Bra89b] Braden(ed.), R. Requirements for Internet Hosts - Communication Layers. Request for comments rfc 1122, IETF, October 1989

[BS04] Baker, F. and Savola, P. Ingress Filtering for Multihomed Networks. Request for comments rfc 3704, IETF, 2004

[CDPZ06] Chan, H., Dash, D., Perring, A. and Zhang, H. Modeling Adoptability of Secure BGP Protocols. In: The joint international conference on Measurement and modeling of computer. ACM, 2006, pp. 389–390

[Che00] Chen, E. Route Refresh Capability for BGP-4. Request for comments rfc2918, IETF, September 2000

[CKCL05] Chang, F., Kaiser, W., Chi, F. and Liu, A. Design and implementation of network puzzles. In: The Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom). IEEE, 2005, pp. 2372–2382

[CM01] Chen, B. and Morris, R. Flexible Control of Parallelism in a Multiprocessor PC Router. In: The USENIX 2001 Annual Technical Conference (USENIX '01). USENIX, June 2001, pp. 1–14

[Com00] Comer, D. Internetworking with TCP/IP Principles, Protocols, and Architectures. Prentice Hall, vol. 1, no. 4, 2000

[DH76] Diffie, W. and Hellman, M. E. New directions in cryptography. IEEE Transactions on Information Theory, vol. IT-11, November 1976, pp. 644–654

[DYC08] Duan, Z., Yuan, X. and Chandrashekar, J. Controlling IP spoofing through interdomain packet filters. IEEE Transactions on Dependable and Secure Computing, vol. 5, 2008, pp. 22–36

[EL09] Ehrenkranz, T. and Li, J. On the State of IP Spoofing Defense. ACM Transactions on Internet Technology, vol. 9, no. 2, 2009

[Fer00] Ferguson, P. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. Request for comments rfc2267, IETF, 2000

[GAG+03] Goodell, G., Aiello, W., Griffin, T., Ioannidis, J., McDaniel, P. and Rubin, A. Working Around BGP: An Incremental Approach to Improving Security and Accuracy of Interdomain Routing. In: Network and Distributed System Security Symposium. San Diego, CA, 2003

[HB96] Hawkinson, J. and Bates, T. Guideline for creation, selection and registration of an Autonomous System (AS). Request for comments rfc 1930, IETF, March 1996

[Hed88] Hedrick, C. Routing Information Protocol. Request for comments rfc 1058, IETF, June 1988

[Hel10] HelionTechnology. AES Cores. `http://www.heliontech.com/aes.htm`, 2008, viewed: Feb 2010

[HHK02] Handley, M., Hodson, O. and Kohler, E. XORP: An Open Platform for Network Research. ACM SIGCOMM Computer Communication Review, 2002, pp. 53–57

[HK97] Halevi, S. and Krawczyk, H. MMH: Software message authentication in the Gbit/second rates. In: The 4th Workshop on Fast Software Encryption, vol. 1267, 1997, pp. 172–189

[HPK01] Handley, M., Paxson, V. and Kreibich, C. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In: 10th USENIX Security Symposium (Security '01). USENIX, August 2001

[IAN09] IANA. Internet Assigned Numbers Authority. `http://www.iana.org/`, viewed: March 2009

[IGM09] Israr, J., Guennoun, M. and Mouftah, H. T. Mitigating IP Spoofing by Validating BGP Routes Updates. IJCSNS International Journal of Computer Science and Network Security, 2009, pp. 71–76

[IH05] Ianelli, N. and Hackworth, A. Botnets as a vehicle for online crime. Request for comments rfc 1700, IETF, December 2005

[KBC97] Krawczyk, H., Bellare, M. and Canetti, R. HMAC: Keyed-Hashing for Message Authentication. Request for comments rfc2104, IETF, 1997

[KLMS00] Kent, S., Lynn, C., Mikkelson, J. and Seo, K. Secure Border Gateway Protocol (S-BGP) – Real World Performance and Deployment Issues. In: Network and Distributed Systems Security 2000, February 2000

[KLS00] Kent, S., Lynn, C. and Seo, K. Secure Border Gateway Protocol (Secure-BGP). IEEE Journal on Selected Areas in Communications, vol. 18, no. 4, April 2000, pp. 528–592

[KMC+00] Kohler, E., Morris, R., Chen, B., Jannotti, J. and Kaashoek, M. F. The Click Modular Router. ACM Transactions on Computer Systems, vol. 18, no. 4, 2000, pp. 263–297

[KMC02] Kohler, E., Morris, R. and Chen, B. Programming language optimizations for modular router configurations. In: 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), vol. 10(02), October 2002, pp. 1–13

[KMP00] Kohler, E., Morris, R. and Poletto, M. Modular components for network address translation. Technical report, MIT LCS Click Projcet, 2000

[Koh00] Kohler, E. The Click Modular Router. Ph.D. thesis, MIT, 2000

[Koh06] Kohler, E. Click for Measurement. Technical report TR060010, UCLA Computer Science Department, 2006

[Kro06] Krovetz, T. UMAC: Message Authentication Code using Universal Hashing. Request for comments rfc4418, IETF, 2006

[KS05] Kent, S. and Seo, K. Security architecture for the Internet Protocol. Request for comments rfc4301, IETF, 2005

[LKHP07] Lee, H., Kwon, M., Hasker, G. and Perring, A. BASE: An incrementally deployable mechanism for viable IP Spoofing prevention. In: 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS'07). Singapore, 2007, pp. 20–31

[LLYW08] Liu, X., Li, A., Yang, X. and Wetherall, D. Passport: Secure and adoptable source authentication. In: USENIX Symposium on Networked Systems Design and Implementation. USENIX, 2008, pp. 365–378

[LME+08] Li, J., Mirkovic, J., Ehrenkranz, T., M. Wang, P. R. and Zhang, L. Learning the valid incoming direction of IP packets. Computer Networks, vol. 52, 2008, pp. 399–417

[LMMWZ02] Li, J., Mirkovic, J., Mengqiu Wang, P. R. and Zhang, L. SAVE: Source address validity enforcement protocol. In: the Annual Joint Conference of The IEEE Computer and Communications Societies (InfoCom). IEEE, 2002, pp. 1557–1566

[LO91] LaMacchia, B. A. and Odlyzko, A. M. Computation of discrete logarithms in prime fields. Designs, Codes, and Cryptography, vol. 1, 1991, pp. 46–62

[LRST00] Lau, F., Rubin, S. H., Smith, M. H. and Trajkovic, L. Distributed Denial of Service Attacks. In: IEEE International Conference on Systems, Man, and Cybernetics. IEEE, 2000, pp. 2275–2280

[LS07] Lv, G. F. and Sun, Z. G. Towards spoofing prevention based on hierarchical coordination model. In: 2007 IEEE Workshop on High Performance Switching and Routing (HPSR). IEEE, 2007, pp. 234–239

[LV00] Lenstra, A. K. and Verheul, E. R. Selecting Cryptographic Key Sizes. Lecture Notes in Computer Science (LNCS), vol. 1751, 2000, pp. 446–465

[MIT10] MIT. MIT ANA Spoofer project. `http://spoofer.csail.mit.edu/index.php`, viewed: 2010

[MJR06] Mirkovic, J., Jevtic, N. and Reiher, P. A Practical IP Spoofing Defense Through Router-Based Filtering. Cis-tr-2006-332, University of Delaware CIS Department, 2006

[MKJK99] Morris, R., Kohler, E., Jannotti, J. and Kaashoek, M. F. The Click Modular Router. In: 17th ACM Symposium on Operating Systems Principles (SOSP '99), vol. 34(5). ACM, 1999, pp. 217–231

[MP03] Meyer, D. and Patel, K. BGP-4 Protocol Analysis. Internet draft, IETF, http://www.ietf.org/internet-drafts/draft-ietf-idrbgp-analysis-04.txt, June 2003

[MR97] Mogul, J. C. and Ramakrishnan, K. K. Eliminating receive livelock in an interrupt-driven kernel. ACM Transactions on Computer Systems, vol. 15, no. 03, August 1997, pp. 217–252

[Mur06] Murphy, S. BGP Security Vulnerabilities Analysis. Request for comments rfc 4272, IETF, 2006

[Odl84] Odlyzko, A. M. Discrete logarithms in finite fields and their cryptographic significance. In: Eurocrypt'84, 1984, pp. 225–314

[PCST01] Perring, A., Canetti, R., Song, D. and Tygar, J. D. Efficient and Secure Source Authentication for Multicast. In: Network and Distributed System Security Symposium, 2001

[PH78] Pohlig, S. C. and Hellman, M. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. IEEE Transactions on Information Theory, vol. IT-24, 1978, pp. 106–110

[PL01] Park, K. and Lee, H. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets. Computer Communication Review, 2001, pp. 15–26

[Pos81] Postel, J. Internet Protocol. Request for comments rfc 0791, DARPA INTERNET PROGRAM, 1981

[Rek00] Rekhter, Y. Multiprotocol Extensions for BGP-4. Request for comments rfc 2858, IETF, June 2000

[RIP10] RIPE. YouTube Hijacking: A RIPE NCC RIS case study. `http://www.ripe.net/news/study-youtube-hijacking.html`, viewed: Feb 2010

[RL95] Rekhter, Y. and Li, T. A Border Gateway Protocol 4. Request for comments rfc1771, IETF, 1995

[RLH06] Rekhter, Y., Li, T. and Hares, S. A Border Gateway Protocol 4 (BGP-4). Request for comments rfc4271, IETF, 2006

[Ros82] Rosen, E. Exterior Gateway Protocol (EGP). Request for comments rfc 827, IETF, October 1982

[Rou09] RouteViewsProject. University of Oregon Route Views Project. `http://www.routeviews.org/`, viewed: May 2009

[SBEL02] Smith, B. R., Banchelli, G., Echeverry, M. M. and Lachmann, A. HTTP Server (powered by Apache): An Integrated Solution for IBM @server iSeries Servers. IBM Corp., 2002

[SBWL08] Shen, Y., Bi, J., Wu, J. and Liu, Q. A two-level source address spoofing prevention based on automatic signature and verification

mechanism. In: The Passive and Active Measurement Conference, 2008, pp. 392–397

[She09] Shepherd, M. C. XORP users manual - Version 1.6. `http://www.xorp.org`, 2009, viewed: September 2009

[SKS+97] Schuba, C., Krsul, I., Spafford, E., Sundaram, A. and Zamboni, D. Analysis of a denial of service attack on TCP. In: IEEE Symposium on Security and Privacy. IEEE, 1997

[SWKA00] Savage, S., Wetherall, D., Karlin, A. and Anderson, T. Practical network support for IP traceback. Computer Communication Review, vol. 30, September 2000, pp. 295–306

[Tal03] Taleck, G. Ambiguity resolution via passive OS fingerprinting. In: The Symposium on Recent Advances in Intrusion Detection, 2003, pp. 192–206

[Tea10] Team, C. E. R. CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks. http://www.cert.org/advisories/CA-1998-01.html, January 1998, viewed: 2010

[TL03] Templeton, S. J. and Levitt, K. E. Detecting Spoofed Packets. In: The DARPA Information Survivability Conference and Exposition (DISCEX'03), 2003

[TMI07] Takurou, H., Matsuura, K. and Imai, H. IP Traceback by Packet Marking Method with Bloom Filters. In: 41st Annual IEEE International Carnahan Conference. IEEE, 2007, pp. 255–263

[Whi09] White, R. Architecture and deployment considerations for secure origin BGP (SoBGP), 2004, viewed: December 2009. Draft-white-sobgparchitecture-00.txt

[WJS07] Wang, H., Jin, C. and Shin, K. G. Defense against spoofed IP traffic using hop-count filtering. IEEE/ACM Transactions on Networking, vol. 15, no. 1, 2007, pp. 40–53

[WXS10] Wang, L., Xia, T. and Seberry, J. Inter-Domain Routing Validator based Spoofing Defence System. In: IEEE International Conference on Intelligence and Security Informatics (ISI 2010). CFP10ITI-PRT, IEEE, Vancouver, BC, Canada, May 2010, pp. 153–155

[YPS03] Yaar, A., Perring, A. and Song, D. Pi: A path identification mechanism to defend against DDoS attacks. In: The IEEE Computer Society Symposium on Research in Security and Privacy. IEEE, 2003, pp. 93–107

[YPS06] Yaar, A., Perring, A. and Song, D. StackPi: New packet marking and filtering mechanisms for DDoS and IP Spoofing defense. IEEE Journal on Selected Areas in Communications, vol. 24, 2006, pp. 1853–1863

[Zal09a] Zalewski, M. Strong attractors and TCP/IP sequence number analysis. `http://lcamtuf.coredump.cx/oldtcp/`, 2001, viewed: 2009

[Zal09b] Zalewski, M. Strong attractors and TCP/IP sequence number analysis-One year later. `http://lcamtuf.coredump.cx/newtcp/`, 2002, viewed: 2009

[Zal09c] Zalewski, M. Passive OS fingerprinting tool. `http://lcamtuf.coredump.cx/p0f.shtml`, 2006, viewed:2009

Appendix A

Click and XORP Resources

More information on Click can be found at

`http://read.cs.ucla.edu/click`

More information on XORP can be found at

`http://www.xorp.org/`

Appendix B

UMAC C Source Code

You can download the whole UMAC C source code at `http://fastcrypto.org/umac/`. The following description is the original description of UMAC C code.

umac.c – C Implementation UMAC Message Authentication

Version 0.92 of draft-krovetz-umac-07.txt – 2006 February 21

For a full description of UMAC message authentication see the UMAC world-wide-web page at `http://www.cs.ucdavis.edu/~rogaway/umac`

Please report bugs and suggestions to the UMAC webpage.

Comments should be directed to Ted Krovetz (tdk@acm.org)

IMPORTANT NOTES

1 This version does not work properly on messages larger than 16MB

2 If you set the switch to use SSE2, then all data must be 16-byte aligned

3 When calling the function *umac*(), it is assumed that msg is in a writable buffer of length divisible by 32 bytes. The message itself does not have to fill the entire buffer, but bytes beyond msg may be zeroed.

4 Two free AES implementations are supported by this implementation of UMAC. Paulo Barreto's version is in the public domain and can be found at `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/` (search for "Barreto"). The only two files needed are rijndael-alg-fst.c and rijndael-alg-fst.h. Brian Gladman's version is distributed with the GNU Public lisence at `http://fp.gladman.plus.com/AES/index.htm`. It includes a fast IA-32 assembly version.

5 With FORCE_C_ONLY flags set to 0, incorrect results are sometimes produced under gcc with optimizations set -O3 or higher. Dunno why.

Appendix C

IP-static.click

```
// This click configuration file used for IP router
// configuration.
// eth2 10.10.10.1 00:0E:0C:BB:04
// eth3 10.10.20.1 00:0E:0C:BB:05

// Shared IP input path and routing table
 ip :: Strip(14)
    -> CheckIPHeader(INTERFACES 10.10.10.1/255.255.255.0
     10.10.20.1/255.255.255.0)
    -> rt :: RadixIPLookup(
     10.10.10.1/32 0,
     10.10.10.255/32 0,
     10.10.10.0/32 0,
     10.10.20.1/32 0,
     10.10.20.255/32 0,
     10.10.20.0/32 0,
     10.10.10.0/255.255.255.0 1,
     10.10.20.0/255.255.255.0 2,
     10.10.30.0/255.255.255.0 10.10.20.2 2,
     255.255.255.255/32 0.0.0.0 0,
     0.0.0.0/32 0);

// ARP responses are copied to each ARPQuerier and the host.
 arpt :: Tee(3);

// Input and output paths for eth2
```

```
c0 :: Classifier(12/0806 20/0001, 12/0806 20/0002,
12/0800, -);
FromDevice(eth2, BURST 100) -> c0; out0 :: Queue(200)
-> todevice0 :: ToDevice(eth2, BURST 100);
c0[0] -> ar0 :: ARPResponder(10.10.10.1
00:0E:0C:C1:B8:04) -> out0;
arpq0 :: ARPQuerier(10.10.10.1,
00:0E:0C:C1:B8:04) -> out0;
c0[1] -> arpt;
arpt[0] -> [1]arpq0;
c0[2] -> Paint(1) -> ip;
c0[3] -> Print("eth2non-IP") -> Discard;

// Input and output paths for eth3
c1 :: Classifier(12/0806 20/0001, 12/0806
20/0002, 12/0800, -);
FromDevice(eth3, BURST 100) -> c1;
out1 :: Queue(200) -> todevice1 :: ToDevice(eth3,
BURST 100);
c1[0] -> ar1 :: ARPResponder(10.10.20.1
00:0E:0C:C1:B8:05) -> out1;
arpq1 :: ARPQuerier(10.10.20.1,
00:0E:0C:C1:B8:05) -> out1;
c1[1] -> arpt;
arpt[1] -> [1]arpq1;
c1[2] -> Paint(2) -> ip;
c1[3] -> Print("eth3non-IP") -> Discard;

// Local delivery
toh :: ToHost;
arpt[2] -> toh;
rt[0] -> IPReassembler -> ping_ipc ::
IPClassifier(icmp type echo, -);
ping_ipc[0] -> ICMPPingResponder -> [0]rt;
ping_ipc[1] -> EtherEncap(0x0800, 1:1:1:1:1:1,
```

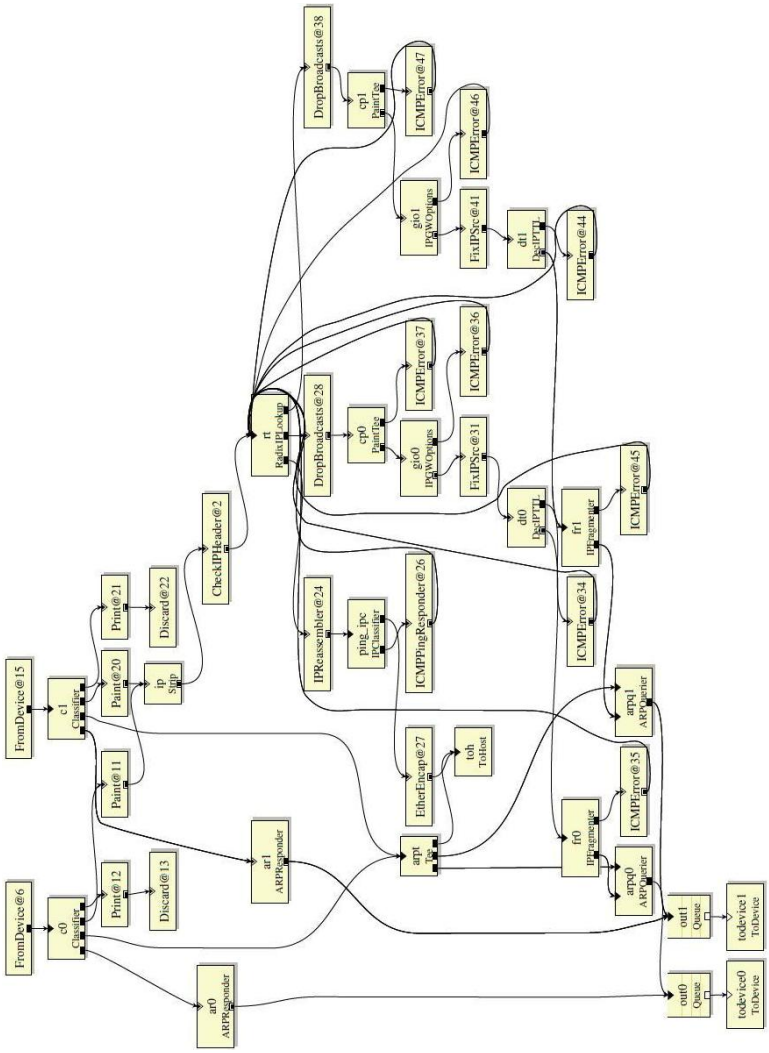
```
 2:2:2:2:2:2) -> toh;

// Forwarding path for eth2
 rt[1] -> DropBroadcasts
     -> cp0 :: PaintTee(1)
     -> gio0 :: IPGWOptions(10.10.10.1)
     -> FixIPSrc(10.10.10.1)
     -> dt0 :: DecIPTTL
     -> fr0 :: IPFragmenter(1500)
     -> [0]arpq0;
 dt0[1] -> ICMPError(10.10.10.1, timeexceeded) -> rt;
 fr0[1] -> ICMPError(10.10.10.1, unreachable, needfrag) -> rt;
 gio0[1] -> ICMPError(10.10.10.1, parameterproblem) -> rt;
 cp0[1] -> ICMPError(10.10.10.1, redirect, host) -> rt;

// Forwarding path for eth3
 rt[2] -> DropBroadcasts
     -> cp1 :: PaintTee(2)
     -> gio1 :: IPGWOptions(10.10.20.1)
     -> FixIPSrc(10.10.20.1)
     -> dt1 :: DecIPTTL
     -> fr1 :: IPFragmenter(1500)
     -> [0]arpq1;
 dt1[1] -> ICMPError(10.10.20.1, timeexceeded) -> rt;
 fr1[1] -> ICMPError(10.10.20.1, unreachable, needfrag) -> rt;
 gio1[1] -> ICMPError(10.10.20.1, parameterproblem) -> rt;
 cp1[1] -> ICMPError(10.10.20.1, redirect, host) -> rt;
```

Appendix D

SDS Click



Appendix E

Click Linux Module Installation Instructions

Note, this appendix is modified from original Click Installation Instructions. You can find and download the whole document at `http://read.cs.ucla.edu/click/download` which included in Click packages. Before you can compile or use the Click loadable kernel module, you need to install, patch, compile, and boot a compatible Linux kernel. These instructions assume you have at least passing familiarity with compiling and running Linux kernels.

1. Archive a complete, working Linux kernel and any corresponding modules. This is in case something goes wrong and you need to revert to an old kernel.

2. Get a vanilla Linux kernel source distribution from `www.kerenl.org` or one of its mirrors. (Patched kernels, such as Red Hat kernels, usually do not work.) Unpack this distribution into /usr/src/linux. (Save the old kernel source tree, if you had one.) We use Linux kernel 2.6.24.7 with Click patch in our experiment. Our OS is Ubuntu 8.04. Unpack linux-2.6.24.7.tar.gz to /usr/src/linux (Ubuntu way)

   $$sudo\ tar\ zxvf\ linux-2.6.24.7.tar.gz\ -C\ /usr/src\ (sudo\ mv\ linux-2.6.24.7\ linux)$$

3. Install the Click Linux kernel patch:

   $$sudo\ patch\ -p1\ -b\ <CLICK/etc/linux-2.6.24.7-patch>$$

   The patch fixes syntax errors in several header files (the C++ compiler doesn't accept them), adds several new functions, and changes the 'struct device'kernel data structure. Therefore, you will need to recompile any kernel modules that touch 'struct device'.

4. If you archived, working kernel has the same version number as the patched kernel (they're both 2.6.24.7, for example), then it is a good idea to change the patched kernel's 'EXTRAVERSION'. This way, the modules for the new kernel won't overwrite the old, working modules. Edit LINUXSRCDIR *sudo gedit* $Makefile$ and change the line 'EXTRAVERSION=' to 'EXTRAVERSION= -experiment', or whatever you would like.

5. Configure the new kernel. The Click patch does not add any configuration options, so you can start from your old configuration, or you can do the usual *sudo make* $xconfig$. Use a minimal set of options. Click is not currently safe for kernels with involuntary preemption. Make sure that the CONFIG_PREEMPT option is off. CONFIG_PREEMPT_VOLUNTARY is OK.

6. Compile and install the kernel:

$$sudo\ dpkg\ -i\ linux-image-2.6.24.7-...$$

7. Reboot your machine with the new kernel.

8. Now you are ready to compile and install the Click module. Rerun $./configure$ to tell the system about your new kernel:

$$rm\ -f\ config.cache;\ ./configure\ [OPTIONS]$$

9. Then build and install the click module, its companion proclikefs module, and the language tools:

$$sudo\ make\ install$$

   If you get errors while compiling the Linux kernel module (the 'linuxmodule' directory), and the errors look like they are in Linux header files, you may have turned on too many options when configuring your kernel. Return to Step 5, turn off the option that seems to be causing a problem.

10. This will create two module object files, 'linuxmodule/click.ko' and 'linuxmodule /proclikefs.ko', and place them in 'CLICKPREFIX/lib'. To install these modules and a router configuration, run

$$CLICKPREFIX/sbin/click-install\ ROUTERCONFIGFILE$$

    Alternatively you could use /sbin/insmod by hand.