ENSURING A VALID SOURCE AND DESTINATION FOR INTERNET TRAFFIC

by

TOBY EHRENKRANZ

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

June 2012

DISSERTATION APPROVAL PAGE

Student: Toby Ehrenkranz

Title: Ensuring a Valid Source and Destination for Internet Traffic

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science by:

Dr. Jun Li	Chair
Dr. Reza Rejaie	Member
Dr. Virginia Lo	Member
Dr. Li-Shan Chou	Outside Member
and	
Kimberly Andrews Espy	Vice President for Research & Innovation/ Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded June 2012

DISSERTATION ABSTRACT

Toby Ehrenkranz

Doctor of Philosophy

Department of Computer and Information Science

June 2012

Title: Ensuring a Valid Source and Destination for Internet Traffic

The Internet has become an indispensable resource for today's society. It is at the center of the today's business, entertainment, and social world. However, the core of our identities on the Internet, the IP addresses that are used to send and receive data throughout the Internet, are insecure. Attackers today are able to send data purporting to be from nearly any location (IP spoofing) and to reroute data destined for victims to the attackers themselves (IP prefix hijacking). Victims of these attacks may experience denial of service, misplaced blame, and theft of their traffic. These attacks are of the utmost importance since they affect the core layer of the Internet. Although the mechanisms of the attacks are different, they are essentially different sides of the same coin; spoofing attacks forge the identity of the sender, while hijacking attacks forge the identity of the receiver. They revolve around the same underlying lack of a secure identity on the Internet. This research reviews the existing state of the art IP spoofing and IP prefix hijacking research and proposes new defenses to close the missing gaps and provide a new level of security to our identities on the Internet.

This material is based upon work supported by the National Science Foundation under Grants No. CNS-0520326 and CNS-1118101. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

This dissertation includes both previously published/unpublished and co-authored material.

CURRICULUM VITAE

NAME OF AUTHOR: Toby Ehrenkranz

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:
University of Oregon, Eugene, OR

DEGREES AWARDED:
Doctor of Philosophy in Computer and Information Science, 2012, University of Oregon
Master of Science in Computer and Information Science, 2008, University of Oregon
Bachelor of Science in Mathematics and Computer and Science, 2002, University of Oregon

AREAS OF SPECIAL INTEREST:
Network Security, Routing Protocols, Anomaly Detection

PROFESSIONAL EXPERIENCE:

Graduate Research Fellow, University of Oregon, 2004-2010

GRANTS, AWARDS AND HONORS:

Erwin & Gertrude Juilfs Scholarship, 2010

Clarence & Lucille Dunbar Scholarship, 2008

CPATH I18N Workshop Travel Grant, Oct. 2008, May 2009

ACM SIGCOMM Travel Grant, Aug. 2007

USENIX Security Travel Grant, Aug. 2006

Baerncopf Scholarship, 2001 & 2002

PUBLICATIONS:

P. Elliott, T. Ehrenkranz, and J. Li, "Buddyguard: A buddy system for fast and reliable detection of IP prefix anomalies," University of Oregon, Tech. Rep. CIS-TR-2012-02, 2012.

T. Ehrenkranz, J. Li, and P. McDaniel, "Realizing a source authentic Internet," in *Proceedings of the Conference on Security and Privacy in Communications Networks*, September 2010.

T. Ehrenkranz and J. Li, "On the state of IP spoofing defense," *ACM Transactions on Internet Technology*, vol. 9, no. 2, May 2009.

J. Li, J. Mirkovic, T. Ehrenkranz, M. Wang, P. Reiher, and L. Zhang, "Learning the valid incoming direction of IP packets," *Computer Networks*, vol. 52, no. 2, pp. 399–417, February 2008.

J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, "BGP routing dynamics revisited," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 7–16, April 2007.

T. Ehrenkranz and J. Li, "An incrementally deployable protocol for learning the valid incoming direction of IP packets," University of Oregon, Tech. Rep. CIS-TR-2007-05, March 2007.

S. Stafford, J. Li, and T. Ehrenkranz, "Enhancing SWORD to detect zero-day-worm-infected hosts," *SIMULATION: Transactions of the Society for Modeling and Simulation International*, vol. 83, no. 2, pp. 199–212, February 2007.

S. Stafford, J. Li, and T. Ehrenkranz, "On the performance of SWORD in detecting zero-day-worm-infected hosts," in *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, vol. 38, no. 3, Calgary, Canada, July 2006, pp. 559–566.

S. Stafford, J. Li, T. Ehrenkranz, and P. Knickerbocker, "GLOWS: A high fidelity worm simulator," University of Oregon, Tech. Rep. CIS-TR-2006-11, 2006.

J. Li, S. Stafford, and T. Ehrenkranz, "SWORD: Self-propagating worm observation and rapid detection," University of Oregon, Tech. Rep. CIS-TR-2006-03, 2006.

J. Li, T. Ehrenkranz, G. Kuenning, and P. Reiher, "Simulation and analysis on the resiliency and efficiency of malnets," in *Symposium on Measurement, Modeling, and Simulation of Malware*, Monterey, CA, June 2005, pp. 262–269.

ACKNOWLEDGEMENTS

I would like to thank my committee for their feedback and insight during my research. Special thanks go to my advisor, Dr. Jun Li—he has been invaluable in our brainstorming and research sessions, and for his technical knowledge and experience. He also always knows when I need encouragement, and when I need some tough criticism. Thank you to Paul Elliott, who will be receiving the Buddyguard torch and advancing the research. Also, thanks to the rest of the network security research group for hearing my ideas and providing feedback over the years, especially Shad Stafford. I would like to thank my wife, Peishan Jiang, for supporting me through my long graduate school journey. I know it was not easy for her, but without her support my journey would have stopped short. And finally, I want to thank my parents, Dave and Penny Ehrenkranz, and my grandmother, Susan Lockwood, for their emotional and financial support during my years as a graduate student supporting a growing family.

For my daughter, Elizabeth Azalea Ehrenkranz.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

Figure Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Attackers on the Internet today can send and receive packets using the addresses of victim networks. Nearly any host on the Internet can be a victim of such "spoofing" and "hijacking" attacks. Victims of these attacks may experience denial of service, mis-placed blame, and theft of their traffic. These attacks are of the utmost importance since they affect the core layer of the Internet. Although the mechanisms of the attacks are different, they are essentially different sides of the same coin; spoofing attacks forge the identity of the sender, while hijacking attacks forge the identity of the receiver. They revolve around the same underlying lack of a secure identity on the Internet. This research enables better defenses against both of these attacks.

1.1. Contributions

Securing identities at the IP layer, including IP spoofing and IP prefix hijacking, has been an on-going research topic for many years. Numerous defenses have been proposed. They often approach the problem from very different angles, and similarly propose solutions using very different mechanisms. Furthermore, the source address side and the destination address side are nearly always considered completely separately, even though they both affect the same identity. This research pulls the two sides together. We make the following contributions:

– Requirements of good IP spoofing defense system: All defenses try to identify and block spoofing packets, but there are many more features necessary for a good solution.

– Analysis of existing IP spoofing defense systems: All of the existing defenses have their pros and cons. Some are better in certain areas than others. Before making a new solution, we must understand what the existing solutions are missing.

– New solution for IP spoofing: With the existing defenses failing to meet all the requirements of a good defense, a new solution is required.

– Requirements of good IP prefix hijacking defense system: A solid understanding of what we should look for in a hijacking defense system is paramount. There are multiple criterion to consider when measuring a prefix hijacking defense system.

– Analysis of existing IP prefix hijacking defense systems: Previous hijacking defense mechanisms leave gaps in their defenses. In order to improve upon them we must understand their limitations.

– New solution for IP prefix hijacking: A new defense mechanism must be created to seal the cracks that allow attackers to sneak through.

– Analysis of issues relating to IP spoofing and IP prefix hijacking defense: With both sides of the IP layer identity better secured, we should know how the defenses relate to each other and what related issues there are.

1.2. IP Spoofing

Even in today's botnet infested Internet, an attacker prefers to use IP spoofing whenever possible. While the attacker may simply spoof a victim's address to hide the real attack source, it is very likely the victim address is the focus of a targeted attack.

For example, only through IP spoofing can an attacker perform DNS amplification (subverting DNS servers to perform a bandwidth-based DDoS attack [4, 5]), reset a victim's TCP connections (sending spoofed TCP reset packets with in-window sequence numbers [6]), poison a DNS cache (transparently redirecting victims to the attacker's server [7]), or circumvent spam filters (getting around mail blocks an ISP may place on a botnet's zombie machines [8, 9]). These attacks can cause the victim to have degraded performance, a complete denial of service, or even be transparently misdirected to the attacker's website. Although the underlying threat of IP spoofing is not new, the problem continues to worsen: attackers persist in finding new ways of crafting attacks using spoofed IP packets, and attackers can spoof from a greater portion of the Internet than before [9].

In today's Internet, attackers can forge the source address of IP packets to both maintain their anonymity and redirect the blame for attacks. When attackers inject packets with spoofed source addresses into the Internet, routers forward those packets to their destination just like any other packet—often without checking the validity of the packets' source addresses. These spoofing packets consume network bandwidth en route to their destinations, and are often part of some malicious activity, such as a DDoS attack. Unfortunately, routers on the Internet today cannot effectively filter out spoofing packets. They either do not know what distinguishes legitimate packets from spoofing packets, or do not leverage such knowledge.

1.2.1. The Missing Gap

For many years the research community has been working hard to combat IP spoofing, starting with early works such as ingress/egress filtering, and continuing through the present with modern solutions such as Spoofing Prevention Method [10] and Distributed Packet Filtering [11]. Many proposed solutions have shown great promise, but the problem has remained unsolved.

In order to understand the spoofing abilities of attackers today, we first examine results from the Spoofer Project [1, 12]. This project attempts to measure the ability of hosts throughout the Internet to send spoofing packets. We will show that, even with the relatively high deployment of ingress/egress filtering, IP source address spoofing remains a severe problem on the Internet. Although many may feel the network community solved the spoofing problem through the widespread adoption of ingress and egress filtering, attackers can still spoof a significant portion of existent IP addresses, often *any* IP address.

1.2.2. Necessary Features and Desired Properties

We then inspect and compare various IP spoofing defense solutions. We provide a comprehensive study of the state of the art, and meanwhile analyze what obstacles stand in the way of deploying those modern solutions and what areas require further research. We compare spoofing defense mechanisms in terms of three required features:

1. Identifying spoofing packets: Fundamental to any IP spoofing defense is the ability to detect spoofing packets. Spoofing packets cannot be defended against if they cannot be identified.

2. Mitigating spoofing attacks: An IP spoofing defense must be able to mitigate an IP spoofing attack, otherwise it is not really a defense. Different defenses may have varying degrees of success in mitigating an attack.

3. Pinpointing an attacker's real location: Falsifying an identity is central to IP spoofing. A good IP spoofing defense must be able to locate the actual source of the IP spoofing attack.

Note that identifying spoofing packets and mitigating a spoofing attack are not equal. For example, with a bandwidth-based denial-of-service attack, even if we are able to identify spoofing packets, we cannot mitigate an attack they cause if the identification is done at or close to the victim. In such a case, the spoofing packets must be filtered out *before* reaching the target network. Furthermore, identifying and mitigating an attack does not mean we can identify the actual attacker. Without being able to locate an attacker, there is no deterrent for attackers—their attacks may be stopped, but as long as they can continue to attack in anonymity there is no risk to themselves or their resources. Not all spoofing defense mechanisms implement all three features, and those that do may have implementations of varying effectiveness.

Spoofing defense mechanisms not only require the above capabilities, but should also maintain a set of desired properties:

1. Resilient against circumvention: An IP spoofing defense cannot rely on traffic characteristics that attackers can easily manipulate and spoof the correct values. If an attacker can discover the correct values for the characteristics they can avoid detection and succeed in their attacks.

2. Easily deployable: A spoofing defense should also be easy to deploy, and provide tangible benefit to early adopters. A system that is difficult to deploy, or that requires full deployment to be effective, stands little chance of ever seeing real world use.

3. Routing protocol independent: While the inter-AS level routing has standardized on BGP, there are a number of intra-AS routing protocols in use. Furthermore, nobody can know for sure what the future holds; BGP may change at some point, or inter AS routing may even move away from BGP at some point. A spoofing defense should be protocol agnostic as possible to ensure deployability across all current and future intra-AS and inter-AS networks.

4. Low overhead: An ideal defense mechanism must incur low overhead so as not to affect network performance, and to be scalable as network sizes increase.

In Chapter II, we study the existing IP spoofing defense systems in detail, and analyze them according to the above capabilities and properties. Then in Chapter III, after seeing that none of the existing systems satisfactorily meet all of the requirements, we describe a new IP spoofing defense solution that does have the necessary and desired properties.

1.3. IP Prefix Hijacking

Due to operational malpractice or security attacks, an *IP prefix* (i.e., a block of IP addresses) can undergo many types of routing anomalies. A prefix may suddenly become unreachable, reachable only through a path with poor routing performance, or it may experience pathological routing dynamics (e.g., oscillation between different paths). Whether a prefix is used by major online businesses (such as Google or YouTube) or ordinary end users (e.g., Alice and Bob), these prefix anomalies can cause loss of revenue, identity theft, or many other devastating consequences.

One of the most infamous of such anomalies is *prefix hijacking*, in which an attacker hijacks traffic meant to reach the legitimate user of a prefix. Real world cases of prefix hijacking have occurred repeatedly [13, 2], including the well-known

Pakistan Telecom hijack of YouTube in 2008 [3]. Furthermore, executing this kind of attack is not difficult. Theoretical studies show that a prefix can be hijacked by a tier 1 autonomous system (AS) with around a 50–80% probability, a tier 2 AS with around a 30–70% probability, and a tier 3 AS with around a 5–30% probability [14].

Another common cause of prefix anomaly is *misconfiguration*, where an ISP advertises illegitimate routes or prefixes. In past history, such incidents have caused anomalies on a monumental scale [15, 16]. A recent instance occurred on April 8, 2010, when an AS operated by China Telecom falsely originated nearly 37,000 prefixes that it did not own, causing traffic to those prefixes to be mis-routed for about 15 minutes [17].

These anomalies are a real threat and *every* prefix on the Internet, whether commercial or private, is vulnerable. Even more disturbing, users or operators of a prefix cannot easily detect such incidents. The legitimate user of a prefix may not expect any incoming traffic at all while its traffic is being mis-routed; or in a more complex form of IP prefix hijacking called *prefix interception*, an attacker not only hijacks traffic, but also forwards that traffic to the victim—leaving the victim entirely unaware.

Unfortunately, it is unlikely that these prefix anomalies will be resolved in the near future. While there exist proposals to secure Border Gateway Protocol (BGP), the *de facto* inter-domain routing protocol, these measures require too much overhead to be quickly deployed. It is therefore critical to monitor prefixes and be able to detect prefix anomalies as they occur.

But in the domain of monitoring prefixes and detecting anomalies, the outlook is still bleak. The current state-of-the-art monitoring schemes are narrowly focused on prefix hijacking alone, leaving other forms of prefix anomalies undetected.

Furthermore, none of the current systems fully address the range of countermeasures that an intelligent attacker could employ to avoid detection. As such, we need more flexible and resilient solutions than what are currently available.

Chapter IV will go into further detail about existing prefix hijacking defenses, and why they are inadequate or easily circumvented. Our new approach that is effective and resilient against circumvention by attackers is then described in Chapter V.

1.4. Secured Identities at the IP Layer

It is important to remember that the two sides of the IP layer identity problem are not entirely independent. Without solving both sides of the identity problem, neither side is truly secured. If you can be sure your packets are reaching their intended destination, but you are not sure packets that you receive come from who they purported to be from, you cannot entirely trust your communication over the Internet. Similarly, if you can be sure the source of the packets you receive is legitimate, but you cannot be sure packets that you send will reach their intended destination, you also have poor communication quality and security.

With our new solutions for IP spoofing prevention and IP prefix hijacking detection, we can get much closer to having truly secured identities at the IP layer of the Internet. Attackers would not be able to hide behind a false identity. Legitimate network operators can have peace of mind knowing that there are no attackers stealing their traffic, nor any attackers masquerading as them to redirect blame for an attack. Network users can be confident that when they send information to someone it will not be intercepted by a hijacking 3rd party.

1.5. Dissertation Organization

The remainder of this dissertation is organized as follows. In Chapter II we will review the existing spoofing defense mechanisms, analyzing them according to the necessary features and capabilities: identifying spoofing packets, mitigating spoofing attacks, pinpointing an attacker's real location, being resilient against circumvention, having easy deployability, being routing protocol independent, and having low overhead. Once we have a solid understanding of the state of the art spoofing defenses, we will then, in Chapter III, describe and evaluate our new spoofing defense system that meets all of the requirements listed above. After covering the source address side of the identity problem, we will move on to the destination address side and discuss IP prefix hijacking. In Chapter IV we discuss the state of the art in prefix hijacking defenses, outlining what is needed and how the currently proposed solutions fall short—especially in terms of resiliency against circumventing detection. With the needs of an IP prefix hijacking detection system clear, we will describe and evaluate our new, highly resilient, hijacking detection method in Chapter V. In Chapter VI we provide further discussion and insight into issues relating to IP spoofing and prefix hijacking. We conclude and review all of our contributions that we presented in Chapter VII.

1.6. Co-Authorship Acknowledgments

Portions of the text in this dissertation are based on published and unpublished co-authored material. Chapter II includes text from [18]. Chapter III includes text from [19]. Chapter V includes text from [20].

CHAPTER II

ON THE STATE OF IP SPOOFING DEFENSE

The analysis of existing IP spoofing defenses provided in this chapter was published in volume 9 number 2 of the journal *ACM Transactions on Internet Technology* in May, 2009 [18]. My professor, Jun Li, and I were the principle investigators for this work.

This chapter will provide a survey of the state of IP spoofing defense, and is organized as follows. In Section 2.1 we discuss why IP spoofing is still a serious problem today. Then in Section 2.2 we categorize the spoofing defense mechanisms from a high level. Section 2.3, 2.4, and 2.5 describe the defense mechanisms in detail, focusing on host-based, router-based, and combination mechanisms, respectively. In Section 2.6 we analyze the pros and cons of all these defense mechanisms, addressing their capabilities and characteristics as well as overhead. We conclude the survey with a discussion on the future of spoofing defense in Section 2.7.

2.1. Spoofing Today

With the prevalence of ingress/egress filtering, one may conclude that attackers are not able to spoof many addresses. Also, some readers may feel that IP spoofing is no longer a problem. With the increase in botnets, it may seem that attackers no longer need spoofing. To see that these conclusions are not valid, we have only to look at the results from the Spoofer Project [1, 12], as well as how attackers use botnets. In fact, IP spoofing continues to be a prospective tool used by malicious users for attack and misdirection.

2.1.1. Spoofer Project

The Spoofer Project measures the ability of hosts throughout the Internet to send spoofing IP packets and the granularity of any ingress/egress filtering that packets from those hosts encounter.

Volunteers throughout the Internet participate in the Spoofer Project by downloading and running a testing program on their end-hosts. This program sends IP packets with forged source addresses towards a Spoofer Project server, where the forgery can use different allocated addresses. When picking allocated addresses to impersonate, the program tries to iterate through neighboring netblocks of the volunteer host—all the way from a neighboring /32 to a /9 netblock. For example, consider a host with IP address 208.77.188.166 that runs the testing program. The program will send a packet with spoofed source address 208.77.188.167 (the "neighbor" /32 block in 208.77.188.166/31). Then it will send packets with source addresses 208.77.188.164, 208.77.188.160, 208.77.188.175, etc. to test the "neighbor" blocks in 208.77.188.166/30, /29, /28, etc. Iterating through neighboring netblocks gives the filtering granularity, or how large of a netblock a volunteer end-host can successfully spoof.

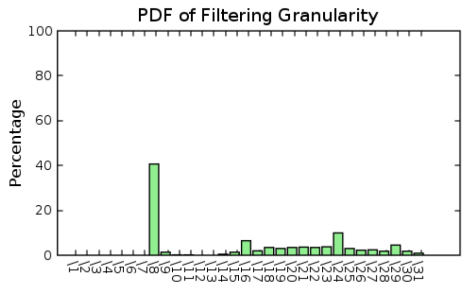
The Spoofer Project keeps track of both how many hosts can successfully spoof at least a single address, and the filtering granularity for such hosts. Assuming the volunteer end-hosts make up a representative sample of all Internet hosts, this project can then estimate the amount of spoofing possible on the Internet.

The results from the Spoofer Project indicate that hosts able to perform spoofing make up a significant percentage of Internet addresses. As of February 7, 2008, the Spoofer Project estimates hosts at 20.3% of all Internet addresses can perform spoofing, or approximately $464,000,000$ out of $2,290,000,000$ addresses. Note that as 20.3% can perform spoofing, it does not mean that attackers can spoof 20.3% of all Internet addresses. In fact, from some locations, attackers can spoof 100% of all source IP addresses.

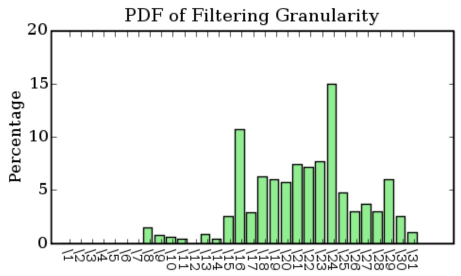
Looking at the filtering granularity helps showcase how many source IP addresses attackers can spoof. A finer granularity means the range of addresses an attacker can spoof is smaller. A coarser granularity means the range of addresses an attacker can spoof is larger. If we compare the original results [12] to the current online results [1], we see that around 40% of the filtering was originally occurring at /8 granularity (Figure 2.1a), but currently less than 2% of the filtering occurs at /8 granularity (Figure 2.1b). This would indicate a drastic increase in filtering efficacy: hosts that could spoof any address within a /8 netblock are now limited to spoofing a much smaller netblock. Unfortunately, however, this difference in granularity is actually because of a change in the reporting methodology.[1] The original results counted hosts that could spoof *any* address as encountering a /8 filtering granularity. The current filtering granularity results simply ignore such hosts. Most of the hosts previously reported as having a /8 filtering granularity could in fact spoof any valid Internet address, and now do not show up in the results at all.

If we consider the sampling used in the Spoofer project, the situation may be even worse. Only 7,980 unique hosts participated in the measurement. Figure 2.2 shows the locations of those hosts which contributed to the Spoofer Project. It is not clear that this sampling is representative of the Internet as a whole. The United States and Europe appear well represented, but since few hosts from other areas of the world contributed measurements, the results may be strongly biased towards US

[1]Discussion relating to this change can be found in the Spoofer Project's mailing list: `https://lists.csail.mit.edu/pipermail/spoofer/2006-August/000009.html`



(a) Results before the change in reporting methodology (August 2006) [12].



(b) More recent results from the Spoofer Project website (February 2008) [1].

FIGURE 2.1. Filtering granularity results from the Spoofer Project. Older results had a large spike at /8 granularity because hosts which did not encounter any filtering were counted as hosts with /8 filtering granularity.

and European networks. If the networks in less-developed areas are less secure, the actual amount of spoofable addresses would be much greater.

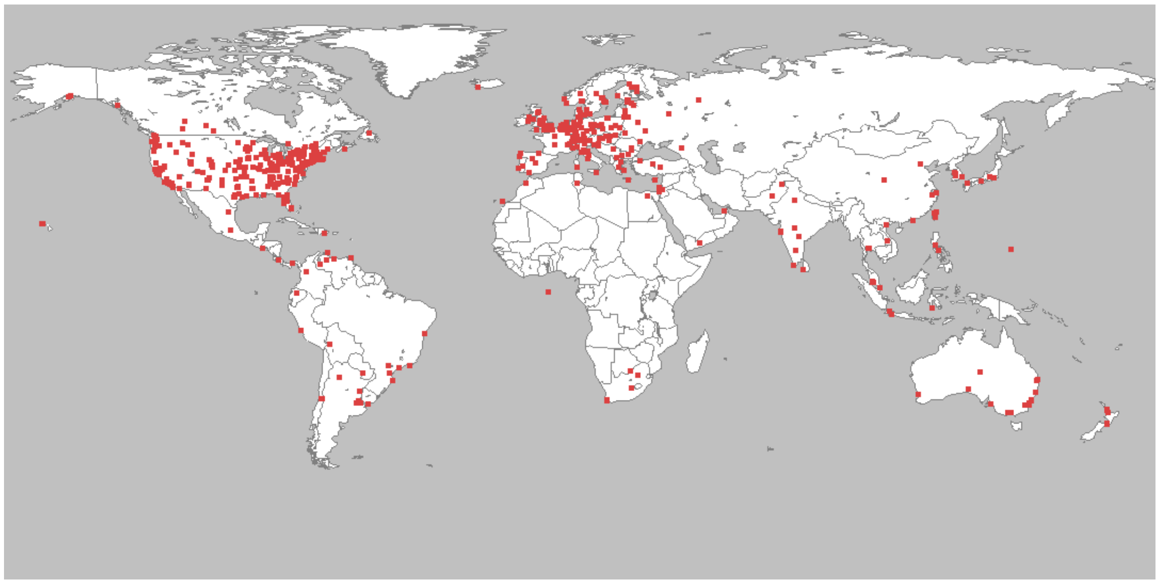


FIGURE 2.2. Locations of hosts that participated in the Spoofer Project [1].

2.1.2. Botnets

With the growing popularity of botnets, some believe attackers no longer need to use IP spoofing. When attackers can control hundreds of thousands of zombie nodes, why should we care about spoofing?

In fact, when considering botnets, IP spoofing remains a problem for defenders and an asset for attackers. In some botnet-based attacks, such as a DNS DDoS amplification attack [21], IP spoofing is vital to the attack's success. In other attacks, IP spoofing may be used but not necessary. Even when not necessary, IP spoofing gives botnet owners another layer of anonymity, and protects their botnet. Botnet owners would prefer to keep the identity of their zombies anonymous as long as possible, in order to prevent defenders from identifying or perhaps even disconnecting zombies. Larger botnets can more effectively perform their tasks, such as stealing identities or producing spam [22, 23], equaling more money for the botnet owner.

Even if botnets did not use IP spoofing, the threat of IP spoofing would still exist. As network defenders and network security researchers, we should not only research how to defend against known attacks. We should learn how to mitigate any possible threat and deal with any underlying flaw of the network infrastructure.

2.2. Overview of IP Spoofing Defenses

IP spoofing remains a severe problem in the Internet today. Unfortunately, although many defense mechanisms have been proposed, none have eradicated the spoofing problem. In this section, we present an overview of these mechanisms.

Spoofing defense solutions can essentially be broken down into three categories:

1. End-host-based solutions. These solutions are implemented on end-hosts, and aim to allow an end-host to recognize spoofing packets. Although some such solutions can be deployed at routers as well, end-host-based solutions are designed with end-hosts in mind, and do not rely upon any special router functionality. In general, these solutions do not need to change networking infrastructure and are the easiest to deploy. On the other hand, they may act too late since the spoofing packets must reach end-hosts before they are detected. (As one seldom deploys an end-host-based solution at source end-hosts—which cannot even come close to preventing IP spoofing from occurring unless every source host is uncompromised and honest. In this paper we will only survey those deployed at destination end-hosts.)

2. Router-based solutions. These solutions are meant to be implemented by routers at the core of the Internet, the edge of the Internet, or both. These solutions generally face more hurdles to deployment, but can be the most effective since they can stop spoofing packets from even reaching end-hosts. Seldom can router-based mechanisms be ported to end-hosts. (Routers may deploy some reactive mechanisms such as tracing where a malicious packet is from. However, reactive mechanisms are triggered only after spoofing packets are detected, so in this paper we focus on router-based mechanisms with the main objective of preventing the delivery of spoofing packets.)

3. Solutions requiring the use of both routers and end-hosts. Routers and end-hosts must work together in order for these solutions to work.

An obvious difference between host-based and router-based mechanisms relates to the end-to-end argument [24]. Host-based mechanisms clearly adhere to end-to-end principles while router-based mechanisms do not. This allows for host-based mechanisms to be deployed much more easily. Host-based mechanisms can generally be deployed even on a single host, without requiring the cooperation of any other host or router. Host-based mechanisms, adhering to end-to-end principles, also avoid increasing the complexity of routers. This can be seen as an argument in favor of host-based mechanisms, but router-based mechanisms should not be discounted so easily.

In contrast to host-based solutions, router-based defense mechanisms are able to address the fundamental weakness which allows IP spoofing. Packets with a forged source address can successfully reach their destination because routers only use the destination address of packets to deliver them and do not verify the source address of packets. Preventive router-based solutions can either provide a way for routers to verify the source address of packets based on their incoming direction, or use some marking to identify the true source of a packet. This allows routers to detect and drop spoofing packets closer to an attacker, before the packets even reach end-hosts.

Next, in Section 2.3, 2.4, and 2.5 we will briefly describe the various spoofing defense mechanisms, and then in Section 2.6 we will analyze their capabilities, characteristics, as well as performance. Table 2.1 lists the specific spoofing defense mechanisms we describe and analyze.

2.3. Host-Based Defense Methods

We can categorize host-based defense mechanisms into active and passive types. Active mechanisms require the end-host to perform some sort of active probing, or other pro-active actions. Passive mechanisms on the other hand rely solely on

TABLE 2.1. Spoofing Defense Mechanisms

Host-based Solutions	**Router-based Solutions**	**Combination**
Active — *Cryptographic:* IPsec *Probing:* OS fingerprinting, IP ID field probing, TCP probing *Other:* SYN cookies, IP puzzles	**Basic —** Martian address filtering, ingress/egress filtering, reverse path forwarding	Pi, StackPi
	Distributed — SPM, Passport, DPF, SAVE, IDPF, BASE	
Passive — Hop-count filtering		

information which a host can gather locally without probing the supposed source of a packet.

2.3.1. Active

Active host-based mechanisms include cryptographic solutions, active probing solutions, and IP puzzles.

Cryptographic solutions, such as IPsec [25], require a handshaking operation to set up secret keys between two hosts. Further communication between the two hosts can be encrypted or signed to ensure any message received by one host was sent by the other host. An attacker would not be able to successfully spoof packets to create a connection, because the attacker would not receive the replies necessary to complete the handshaking process. Similarly, an attacker attempting to spoof packets of an existing connection would fail, because the attacker could not know the secret key. Although IPsec is useful in many situations to prevent spoofing, and provide confidentiality and integrity guarantees, it is unrealistic to use in *all* situations. First of all, it is not feasible to require all end-hosts connect through IPsec. Furthermore, the computational cost of encrypting or signing every packet would mean hosts cannot maintain as many active connections. The computational cost also precludes routers from forming IPsec tunnels for all connections.

Active probing solutions can involve a number of types of probes, including active operating system fingerprinting, IP identification field probes, and TCP specific probes. Note that these probing mechanisms are not designed to be used for spoofing defense, but they can be used for spoofing defense.

Probing a host to determine its operating system, using tools such as NMAP [26], can prove valuable in detecting spoofing packets. These tools send carefully crafted packets to an end-host and observe its response. Although most operating systems generally follow standard protocol specifications, their specific implementations may measurably differ—these implementation differences act as an operating system fingerprint. If a host can actively fingerprint the supposed source host and find that it is running operating system X, while passive fingerprinting [27, 28, 29] on the original received packet suggests it is running Y, it is likely the original packet was spoofing. Unfortunately, fingerprinting involves a high amount of overhead. A host must send numerous probes and buffer suspicious packets if the host requires sources be verified before processing the suspicious packets. Furthermore, fingerprinting cannot identify spoofing packets if the attacker uses the same operating system as the host located at the spoofed address.

Probing the identification field of IP packets can also identify spoofing packets. After receiving a suspicious packet, a host can send packets to the supposed source to observe the identification field in its response. Different IP stacks may set the identification field of IP packets differently. For instance, some hosts may choose a random identification number, and others may simply increment the identification number for every packet. Assuming the source host simply increments the identification field for every packet, the identification number in the probe response should be near the identification number of the suspicious packet; otherwise, the suspicious packet was a spoofing packet. Unfortunately, if the source host sets the identification field in a more complicated manner, it may be difficult or impossible to decide whether or not a suspicious packet carries a spoofed source address.

Using TCP-specific probes is another clever way of defending against spoofing packets. Simply requiring a TCP handshake may not be enough to prevent attackers from spoofing TCP packets, since attackers may be able to predict TCP sequence numbers. Although many operating systems use a random sequence number selection, the pseudo-random number generators they use may not be random enough [30, 31]. TCP-specific probes intelligently craft TCP acknowledgment messages to add another layer of protection. Since the sender of spoofing packets is often unable to see any replies, a recipient host can send acknowledgments that should change the TCP window size or cause packet retransmission, and then observe whether or not the supposed source responds correctly. If the supposed source does not change the window size, or does not retransmit the packet, the recipient host considers the packets' source to be spoofed.

Some servers use SYN cookies [32] to prevent opening connections to spoofed source addresses. The main reason to use SYN cookies, however, is to mitigate the effects of SYN floods by making the TCP handshake stateless [33]. When a server uses SYN cookies it does not allocate resources to a connection until the 3-way TCP handshake completes. First the server sends a SYN + ACK packet with a specially encoded initial sequence number, or cookie, that includes a hash of the TCP headers from the client's initial SYN packet, a timestamp, and the client's maximum segment size (MSS). Then when it receives the client's response, the server can check the sequence number and create the necessary state only if the client's sequence number is the cookie value plus one. Because the cookie uses a hash involving the server's secret key, attackers should not be able to guess the correct cookie values. However, because of performance concerns and some incompatibilities with TCP extensions, such as large windows, operating systems generally do not activate the SYN cookie mechanism until the host's SYN queue fills up. An attacker sending spoofing traffic at a low rate may avoid triggering the SYN cookie mechanism. Administrators may be able to forcibly enable SYN cookies for all connections, but should be aware of the side effects.

IP puzzles [34] are another mechanism hosts can employ to actively defend against spoofing. A server sends an IP puzzle to a client, then the client needs to "solve" the puzzle by performing some computational task. Only after the server receives the puzzle solution from the client will the server allow the client to connect. While the main goal of IP puzzles is to make it prohibitively expensive for malicious hosts to send large numbers of packets, a side effect is preventing attackers from successfully sending spoofing packets. Since the IP puzzle would be sent to the listed source and not the attacker, an attacker could not send a puzzle solution, thus preventing the attacker from spoofing.

2.3.2. Passive

Passive host-based spoofing defense mechanisms decide whether or not a packet is spoofing by passively observing incoming traffic.

Hop-count filtering [35, 36], or HCF, observes the hop count of packets arriving at a given host. First, by measuring the hop-counts during normal times, HCF creates a mapping of IP addresses to hop-counts. Then, if an attacker sends a spoofing packet to the host, it is likely the hop-count of the packet will not match the expected hop-count for packets from the spoofed source address. Because legitimate hop-counts may change due to routing changes, strictly filtering all packets that do not match would lead to false positives. In order to minimize false positives, HCF only begins filtering traffic if some threshold amount of packets do not match their expected hop-counts. This threshold protects against mistakenly filtering legitimate packets, but it also makes HCF ineffective against low amounts of spoofing packets that do not reach the threshold. Furthermore, because the range of expected hop-counts on the Internet is narrow, around 10% of the spoofing packets can be expected to have the correct hop-count [35].

An earlier paper [37] also covered many of these active and passive host-based methods, and they have remained largely unchanged. For more detailed descriptions we refer readers to [37].

2.4. Router-Based Defense Methods

Router-based spoofing defense methods generally take a different approach from host-based mechanisms. Although in principle most host-based methods could also be used by routers, researchers generally only consider a few, such as IPsec or IP puzzles, for use at the router level. Other host-based methods generally require too much overhead that would impact router performance.

Router-based defense mechanisms are all similar in that they perform some sort of filtering to prevent spoofing packets from reaching their intended destinations. The various mechanisms' differences lie in what information they use to decide whether a packet contains a spoofed source address, and where in the network the filtering takes place. We will discuss more basic, traditional mechanisms along with state-of-the-art distributed filtering solutions.

2.4.1. Basic Filtering

One of the earliest and simplest methods of filtering packets with spoofed source addresses is "Martian" filtering [38]. Martian filtering simply involves examining IP header fields and looking for invalid IP addresses, for instance non-unicast source addresses, loopback addresses, or some other "special" addresses. By design, this mechanism can only stop the most obvious and simple types of IP spoofing.

Ingress [39] and egress [40] filtering are the most well-known filtering methods. Run on a router at the border of a network, ingress/egress filtering checks the addresses of packets flowing into and out of an edge network. For a packet originating from the edge network, if the source address does *not* belong to the edge network, the packet is a spoofing packet. Similarly, for a packet originating outside the edge network, if the source address *does* belong to the edge network, the packet is a spoofing packet. The border router is easily able to filter out any packet it identifies as a spoofing packet. If this simple filtering mechanism were implemented on all networks, attackers would be limited to only spoofing addresses within their own local network. Unfortunately, as the Spoofer Project shows us, such universal deployment is not a reality. Furthermore, there is not much incentive for a network to deploy ingress/egress filtering—when a network deploys ingress/egress filtering, it has only a minimal effect on how many other networks are able to spoof its source addresses. But incentive is not the only problem—without nearly 100% deployment, ingress/egress filtering is ineffective [11]. Even if only a small number of networks do not implement ingress/egress filtering, hosts within those unprotected networks would still be able to spoof nearly any source address.

Another basic filtering method is reverse path forwarding, or RPF [41]. Similar to ingress/egress filtering, routers running RPF attempt to filter packets depending on where a packet with a given source address should originate from. But whereas ingress/egress filtering only considered two directions—into an edge network or out of an edge network—RPF attempts to deal with traffic passing through a given router from any direction to any direction. RPF uses the forwarding table information at a router. It assumes that whichever direction, or interface, a packet with destination address γ should be forwarded to is the same direction a packet with source address γ should arrive from. Unfortunately, with the high amount of routing asymmetry in the Internet today [42, 43], this assumption is not valid. Without this assumption holding true, RPF cannot be deployed in many places.

2.4.2. Distributed Defense Methods

In distributed methods of spoofing defense, routers cooperate in order to discover information for distinguishing valid and spoofing packets. The information may be related to a key which only valid packets will carry, or to the incoming direction for packets from a given source. First we discuss Spoofing Prevention Method (SPM) and Passport, and then cover Distributed Packet Filtering (DPF) and other path-based filtering works including Source Address Validity Enforcement (SAVE), Inter-Domain Packet Filters (IDPF), and BGP Anti-Spoofing Extension (BASE). Routers using SPM and Passport mark outgoing packets with secret keys, while routers using DPF and other path-based filtering systems need to learn the correct incoming direction of packets for a given source.

2.4.2.1. Spoofing Prevention Method

Routers implementing Spoofing Prevention Method, or SPM [10], validate a packet by checking for a secret key embedded into the packet. A source Autonomous System (AS), s, decides upon a key for every (s, d) pair, where d is a destination AS. When a packet reaches a router in AS d, the router checks for the presence of the secret key. Any packet with the key is valid, and any packet without the key is spoofing. Packets from ASes not deploying SPM do not have associated keys, so a router cannot know if a packet purporting to be from an unprotected AS is spoofing nor not. Packets from these unprotected ASes are allowed through, but when a router's network is under attack the router gives preferential service to legitimate packets from ASes deploying SPM.

In order to disseminate the secret keys that a source router uses to mark valid packets, SPM can use either a passive or active key distribution protocol. Using a passive distribution protocol, SPM routers must infer the correct key based on connections they observe. Active distribution uses a mechanism similar to the Inter-domain Routing Validator (IRV) [44] used for securing BGP. To maintain a lighter load on routers, each AS has a server which is in charge of all key-related communication. The server keeps track of all keys which routers should check on incoming packets, and disseminates to other ASes' servers the keys that its routers will embed in outgoing packets.

Passive distribution requires a router at a destination AS to monitor connections that require some handshaking process, such as TCP connections, since it is regarded as difficult to successfully initiate such connections using spoofing packets. In the case of routing asymmetry, the router may not be able to see both sides of a connection and must utilize TCP intercept [45] to verify source validity. If a router can only see the incoming side of a supposed connection, it cannot know if the connection actually exists, or if an attacker is simply pretending to receive valid responses. The key that is embedded into a verified valid connection is taken to be the key that the source AS uses to send to the destination router's AS. Since it is possible, albeit difficult, to complete a TCP handshake with spoofing packets by predicting TCP sequence numbers, an attacker could try to trick SPM into storing a false key. SPM would then identify spoofing traffic as legitimate and legitimate traffic as spoofing. Such an attack would be highly difficult, since SPM only observes a single connection to passively learn a new key: not only would an attacker need to complete the TCP handshake, but it would also need to do it at just the right time for SPM to observe that spoofed handshake instead of some other legitimate handshake.

Active distribution requires each AS to maintain a server that keeps track of all key related information. The server knows how to contact key servers in all other ASes, and propagates key information to the other ASes. Each server also maintains all the necessary incoming key information it receives, and ensures local routers have up-to-date incoming key information. The local routers use the incoming key information to verify incoming packets.

In order to maintain reasonable storage requirements, a router only stores key information for source/destination key pairs in which the router's AS is either the source or destination AS. Unfortunately, this means intermediate routers cannot assist in filtering spoofing packets—only the source or destination AS can filter a spoofing packet.

2.4.2.2. Passport

Routers running the Passport [46] system also use secrets embedded in packets to verify source addresses. Instead of embedding a single secret key in the IP header, Passport defines its own header, to allow for a much larger space to hold secrets. This header can be thought of as a “passport” that contains multiple “visas,” with each visa corresponding to a Passport-enabled AS along the path that a packet will travel. As the packet travels towards its destination, Passport-enabled ASes verify the visas. Each visa in the passport is a Message Authentication Code (MAC). A packet’s source AS computes the MACs using secret keys shared between itself and each AS along the path to the packet’s destination. Each MAC covers the packet’s source address, destination address, IP identification field, packet length field, and the first 8 bytes of the payload. When a downstream AS-level router encounters a packet with a passport, it can verify the passport by recalculating the MAC value using the secret key it shares with the source AS. If the verification fails, the router will then demote or drop the packet.

In order for each pair of ASes to have a unique shared secret key, each participating AS performs a Diffie-Hellman [47] key exchange. Each AS tells all other ASes its public key by piggybacking its public key onto BGP update messages. When one AS learns the public key of another AS, it can construct a shared secret key from its own private key and the other AS’s public key. The other AS can construct the same shared secret key using a similar process.

2.4.2.3. DPF

Distributed Packet Filtering, or DPF [11], proposed having routers throughout the network maintain incoming direction knowledge (knowledge of which interface a packet from a given source to a given destination should arrive on). When a packet with a spoofed source address arrives at an incorrect interface, the router can detect this and filter the packet.

When a portion of routers in the network have such incoming direction knowledge, the set of addresses that an attacker can successfully spoof decreases. For instance, if an attacker behind some router, X, wants to successfully spoof addresses behind some other router, Y, packets from behind X and those from behind Y must arrive on the same interface at every DPF-enabled router en route to the destination.

The major omission from the DPF research was the actual method for routers to learn the incoming direction information. Instead, the research assumed such knowledge was available to routers and focused on defining efficacy metrics, and analyzing the effects of using various topologies and deployment distributions.

Below, we present works which provide the missing piece of how routers can learn incoming direction knowledge.

2.4.2.4. SAVE

The Source Address Validity Enforcement protocol, or SAVE [48], while not designed with DPF in mind, operates similarly in that routers filter packets based on their incoming direction. Whereas the original DPF work did not provide a means of discovering valid incoming direction knowledge—DPF simply assumes routers have that knowledge—SAVE is the first protocol that helps routers learn the information and leverage the knowledge for filtering spoofing packets.

The SAVE protocol operates alongside any routing protocol by having the router (or set of routers) in charge of a given source address space send SAVE updates corresponding to each forwarding table entry. Each SAVE update travels through the network along the same path as normal traffic from that source address space. When a downstream router receives the update, it records the incoming interface of the update as the valid incoming direction for the corresponding source address space. Updates are sent both periodically, and whenever a router changes its forwarding table.

Another contribution from SAVE is its invention of an "incoming tree." Every router maintains its own incoming tree that keeps track of the topological relationships of upstream source address spaces. Thus, when one routing change affects the incoming direction of many spaces, a router can automatically update the information for every effected space.

SAVE assumes that all routers deploy SAVE. For intra-domain operation such an assumption is feasible. A single domain can easily deploy a new protocol like SAVE across all its routers. For inter-domain operation, however, SAVE is yet to be designed for correct functionality when some routers do not run SAVE.

2.4.2.5. IDPF

Inter-Domain Packet Filters, or IDPF [49], attempts to provide an implementation of the DPF principles. Learning from BGP updates, and assuming that BGP routers adhere to a specific set of exporting rules, routers running IDPF can discover AS relationship information and then use this information to build packet filtering rules. In general, ASes can be in a provider-customer, peer-peer, or sibling-sibling relationship. These relationships put constraints on which AS paths are feasible, and which are not feasible. Packets which arrive from neighbors along an infeasible path can be filtered out.

Whereas the original DPF research specifies that routers know the actual valid incoming direction of packets, routers running IDPF only know *feasible* incoming directions, not actual incoming directions. With this limited knowledge, IDPF is not as effective as a precise DPF implementation—attackers are able to successfully spoof more source address spaces. Also, with IDPF's dependence on AS relationship information gathered from BGP updates, it is limited to only functioning alongside BGP.

2.4.2.6. BASE

BGP Anti-Spoofing Extension, or BASE [50], is another path-based packet filtering mechanism. As its name implies, BASE's implementation relies on BGP. The basic operation of BASE is similar to that of SAVE; it sends updates that routers use to learn the correct incoming direction of packets. However, instead of treating the incoming interface as the incoming direction, each BASE router marks packets with a unique key and uses the key as the incoming direction. Using markings as opposed to a physical interface is useful for incremental deployment when BASE routers are not physical neighbors. Another important difference is that BASE-enabled routers send updates by piggybacking them on top of BGP updates, as opposed to sending updates on their own. These piggybacked updates include marking information and control messages. Updates for distributing marking information include a source AS and a corresponding 16-bit marking. When a BASE router receives such an update, the router records the enclosed marking as the "incoming direction" for packets from the specified source prefix. Relying on BGP updates means BASE updates must travel the same path as BGP updates. BGP updates do not always travel the same path as normal traffic, however. The path a BGP update for prefix P takes to reach

AS X does not define the path that packets from P follow to reach AS X—it defines the path that AS X can use to forward traffic towards P. The path of normal traffic from P to AS X may be different. When updates and normal traffic travel different paths, routers will expect the incorrect marking and misidentify legitimate packets as spoofing packets. To minimize such false positives, BASE uses control messages to enable or disable filtering. Thus, BASE routers are only able to filter spoofing packets after receiving instructions to filter.

2.5. Combination-based Defense Mechanisms

Spoofing defense methods which utilize both routers and hosts generally mark packets. First, routers mark packets as they travel through the network. Then, when a packet reaches an end-host, the end-host can take action by using the marking, such as tracing the true origin of a packet regardless of its source address field.

A major problem with these solutions is that action is not taken upon the attack packets until they reach the victim at the edge of the network. Allowing attack packets to get so close to their destination may mean the attack cannot be mitigated, even if the system can identify the packets.

2.5.1. Pi

Path Identifier, or *Pi* [51], originally designed to defend against denial-of-service attacks, also provides an IP spoofing defense solution. Pi reuses the fragmentation field of an IP packet to identify the path the packet traveled. As a packet travels the network, each router it encounters sets a bit in the fragmentation field. When the packet reaches its destination, the fragmentation field will contain a marking that is (almost) unique to the path the packet traveled. The end-host does not know what path the packet traveled, but if multiple packets have the same marking it is highly likely that they traveled the same path. If an end-host can identify a packet as an attack packet, then the end-host can filter out subsequent packets which have the same path identifier. Initially identifying which packet is an attack packet is left as a separate problem for the end-host to solve on its own.

Since the marking is not entirely unique for each path, and routes are dynamic, some legitimate packets will be lost if Pi simply drops all packets with an "attack path" marking. In order to minimize any such false positives, the authors recommend only dropping packets when a host is under attack.

2.5.2. StackPi

StackPi [52], essentially an improved version of Pi that functions better than Pi with incremental deployment, also added mechanisms to protect against spoofing packets. To improve the performance with incremental deployment, a router running StackPi is able to set a bit not only on behalf of itself, but also on behalf of the next-hop router. To protect against IP spoofing, an end-host can remember markings it sees in legitimate packets from different source addresses during normal periods of time. Then, when under attack, the end-host can filter out any packet which has a marking that does not match the stored marking for the source of that packet. Note, there is no technical reason end-hosts in the Pi system cannot perform a similar comparison of markings—the authors simply did not evaluate such usage in the original Pi paper.

StackPi, just as Pi, recommends only dropping packets when a host is under attack, since the marking is not entirely unique.

2.6. Analysis

With a basic understanding of how the various spoofing defense mechanisms work, we now analyze the capabilities and characteristics of each mechanism. We first look at how well they can identify spoofing packets, and what sort of deployability issues they may have. We then consider what sort of traffic characteristics the spoofing defense mechanisms rely on, and their routing protocol independence. Next we look at how well they can mitigate attacks and discover an attacker's true location. Finally we compare the overhead of these mechanisms.

For each capability and characteristic there is at least one spoofing defense mechanism that excels in that area—but there is no single mechanisms that excels in all areas. For instance, some mechanisms may be able to identify all spoofing packets when deployed across the entire Internet, but would not work in the real world because of deployability issues.

2.6.1. Identifying Spoofing Packets

All of the spoofing defense mechanisms can of course identify *some* spoofing packets, but they often cannot identify *all* spoofing packets. Table 2.2 shows how the different spoofing defense mechanisms perform at identifying spoofing packets.

IPsec can consistently identify spoofing packets. An attacker cannot setup a connection because it will not be able to receive a response. And after the connection setup, an attacker cannot know the key needed to spoof a packet.

OS Fingerprinting can detect spoofing packets if the spoofed source can be actively fingerprinted *and* the resulting fingerprint is different from the passive fingerprint of the spoofing packet. Even then, results can be complicated by a firewall

TABLE 2.2. Identifying Spoofing Packets

Mechanism	Efficacy	Note
Hop-count filtering	$\approx$ 90%	
TCP/IP Probes & OS Fingerprinting	unknown	Has not been analyzed
SYN cookies	100%	Only for use with TCP connections when SYN queue is full
IPsec	100%	Requires client to use IPsec as well
IP Puzzles	100%	Requires client to understand IP puzzles as well
Martian filtering	100%	Only when attackers spoof a few special addresses
Ingress/egress filtering	100%	Requires 100% deployment, poor otherwise
RPF	unknown	Has not been analyzed
SPM	$\approx$ % of deployment	Spoofing packets only identified if destination runs SPM
Passport	$\approx$ % of deployment	
DPF	$\approx$ 96% of AS pairs w/ vertex cover	$\approx$ 76% with 50% random placement
	$\approx$ 88% of ASes w/ vertex cover	$\approx$ 65% with 50% random placement
SAVE	100% w/ full deployment	Yet to be analyzed without full deployment
BASE	$\approx$ 90% w/ 10% priority placement	Top 10% of ASes by degree
	$\approx$ 20% w/ 10% random placement	
IDPF	$\approx$ 80% of ASes w/ vertex cover	$\approx$ 60% w/ 50% random placement
StackPi	$>$ 99% w/ full router deployment	Yet to be analyzed without full deployment

between the target and the spoofed source, if the firewall filters the fingerprinting probes, or alters the responses. Fingerprinting is not reliable enough to depend on.

IP identification field probing can identify spoofing packets, but only if the spoofed source does not use any sophisticated methods for identification number assignment. Furthermore, results can be complicated by a firewall if it either filters the probing, or alters the response; this mechanism cannot be relied upon.

TCP-specific probes can identify spoofing packets fairly easily and reliably. An attacker will not receive the TCP control messages, and thus cannot react correctly. Of course, this mechanism is useless against UDP or other non-TCP packets.

IP puzzles could also identify spoofing packets fairly easily, for the same reason—the attacker will not receive the puzzle and thus cannot send a puzzle solution. Since IP puzzles work at the IP layer, below UDP or TCP, all Internet traffic could be validated.

Hop-count filtering can identify spoofing packets when the hop-count from the attacker to the destination is different from the hop-count from the spoofed source to the destination. Since the range of likely hop-counts is narrow, many spoofing packets may arrive at the destination with the correct hop-count.

Basic router-level filtering methods can effectively identify all spoofing packets (assuming the spoofed source belongs to a different network), but only if all routers employ filtering. As the Spoofer Project [1] shows, this has not happened. When considering filtering only deployed at a subset of routers, RPF can provide greater efficacy than ingress/egress filtering, but such a strategy has yet to be analyzed.

SPM can identify all spoofing packets whose destination and spoofed source both belong to different SPM-protected domains—but only if the keys used for marking remain secret. Packets with a spoofed source belonging to a non-SPM-protected domain will not be identified.

Passport can identify a spoofing packet when the spoofed source's AS is Passport-enabled, and at least one AS on the path of the spoofing packet is also Passport-enabled. Packets with a spoofed source belonging to a non-Passport-enabled AS will not be identified.

DPF, assuming a vertex-cover deployment and an oracle-based method of building incoming direction information, cannot identify all spoofing packets—but it can identify a large majority. When AS-level routers deploying DPF form a minimum size vertex cover, DPF can identify all spoofing packets from around 88% of all ASes, and if we consider all source-destination AS pairs, only 4% are feasible attack pairs, or source and destination AS pairs between which DPF cannot identify spoofing packets.

SAVE can identify all spoofing packets where the attacker and destination are in different networks. Without being fully deployed on all routers, however, SAVE could not identify all spoofing packets—it has yet to function correctly when not all routers run SAVE.

IDPF is not an ideal implementation of DPF, and it cannot identify as many spoofing packets as an optimal DPF. IDPF filters spoofing packets by checking against feasible paths that packets may travel, whereas DPF knows the actual paths. When AS-level routers deploying IDPF form a vertex cover, IDPF can identify all spoofing packets from around 80% of all ASes. Further details regarding the effectiveness of IDPF at identifying spoofing packets are unavailable.

BASE also performs similarly to DPF, and is able to identify most spoofing packets. With the top 30% of ASes (according to AS-connectivity degree) BASE-enabled, BASE can identify around 97% of spoofing packets. Measurements are not available that are more directly comparable to DPF and IDPF. To avoid false positives because of AS-level routing asymmetry, BASE only enables filtering *after* a spoofing attack is detected.

Pi and StackPi can identify spoofing packets as long as the target host has received legitimate packets from the spoofed source network (otherwise it cannot know what the correct marking should be). It cannot identify all spoofing packets since some markings will overlap. Assuming full deployment, and that the target host has seen legitimate packets from every network, the probability that an attacker (with a random IP address) could successfully send a spoofing packet (with a random source IP address) is less than one percent. Efficacies with lower deployment levels are unknown.

2.6.2. Deployability

In general, host-based mechanisms are easier to deploy than router-based methods. It is easier to install new software at end-hosts than on routers. Also, host-based methods generally do not need as much cooperation as router-based methods. For instance, active probing and passive host-based methods can offer protection for a given host—even when that host is the only host using such a mechanism. However, IPsec can only be relied on by an end-host if all end-hosts it communicates with also implement IPsec. Similarly, in order for administrators to rely on IP puzzles, IP puzzles must first be standardized and deployed so that all network nodes can understand the puzzles.

Some router-based mechanisms, including SPM, Passport, BASE, Pi, and StackPi, rely on packet markings. Deployment problems arise when using IP header fields reserved for other purposes. Excluding Passport, all of the other proposed packet markings use the IP identification field, which is needed to properly reassemble fragmented packets. Additionally, there are many IP traceback mechanisms that want to re-use the identification field for their own purposes such as [53, 54, 55, 56]. There is no standard for any alternative usage of the IP identification field, and at the present time, we cannot know which usage will prevail. Passport uses the IP Options field to insert multiple markings in a packet. This avoids direct incompatibilities with the IP identification field, but fragmentation will still invalidate the markings, since an intermediate router cannot recalculate the correct markings. Therefore, Passport treats all fragmented packets with a lower priority. Using IP Options also makes every packet's IP header require more space. IDPF does not use any packet markings and can be deployed by even a single router, but there is not a large incentive for administrators to deploy the protocol—deploying IDPF does not directly benefit the deploying AS more than another AS. SAVE is more deployable in that it uses no packet markings and can run alongside any routing protocol, but it has yet to be designed to function without full deployment—work is ongoing to allow SAVE to work with incremental deployment [57]. Basic router filtering also uses no markings and is easy to deploy, but lacks incentive for deployment.

2.6.3. Essential Traffic Characteristics

In order to be resilient against more intelligent attackers, spoofing defenses cannot rely on traffic characteristics that an attacker can easily manipulate and spoof the correct values. First we will discuss the resiliency of router-based and combination-based systems to manipulation, then we will discuss the resiliency of host-based systems.

SAVE, basic router filtering, DPF, and IDPF validate packets using their physical incoming interface. An attacker cannot manipulate which interface attack packets enter at a router.

The rest, including SPM, Passport, BASE, Pi, and StackPi rely on packet markings that an attacker can attempt to manipulate, which attackers can manipulate to attempt falsifying marking values. Using SPM, routers do not change a packet's marking once it leaves the source AS, and only the destination AS checks the marking. Furthermore, all packets from a given source AS to a given destination AS contain the same marking. Thus, if an attacker sniffs or otherwise learns a valid marking, the attacker can use that marking to successfully spoof that source to that destination from any non-SPM location. In contrast, markings in Passport change for every packet; an attacker cannot use markings from a legitimate packet to craft markings for spoofing packets. With BASE, Pi, and StackPi, routers along a packet's path will modify the packet's markings—even if an attacker knows the legitimate marking values set at the source and seen at the destination, setting the correct initial marking is much more difficult. Since the path from the attacker to the destination is different from the legitimate source to the destination, the marking will be modified in a different manner.

Router-based and combination-based solutions that use packet marking also need to be careful in dealing with packet fragmentation. SPM, BASE, Pi, and StackPi use the IP identification field to store their markings. But the IP identification field is needed to properly reassemble fragmented packets. Passport uses the IP identification field and the length field to calculate its markings. But when a router fragments a packet, the router changes the length field—making any previously calculated marking invalid. In order to avoid affecting (or being affected by) fragment reassembly, a mechanism can avoid marking and processing fragmented packets—but then an attacker can break through the defense mechanism simply by making fragmented packets. One might choose to simply drop all fragmented packets when under attack, but then the question is how does the defense mechanism decide when an attack is occurring? Furthermore, even if one knows an attack is occurring, how large must an attack be before the cost of losing fragmented legitimate traffic is outweighed by the benefit of dropping spoofing traffic?

Regarding host-based defenses, attackers can also manipulate certain traffic characteristics. If using hop-count filtering and OS fingerprinting, for example, attackers can easily set a packet's initial TTL value to any value, and craft packets which seem to originate from any operating system. While it is difficult for an attacker to know the correct TTL value or OS type to set for a specific source address, if they have a way of probing to see when an attack succeeds they can try multiple values until the attack succeeds. Once they find the correct value, they can continue to use that value.

Attackers can hardly manipulate the host-based defense methods such as IPsec, IP puzzles, SYN cookies, and TCP probes, that rely on end-hosts knowing a secret key or receiving secret control messages. Although these secrets are somewhat similar to markings router-based methods use—both involve the source knowing a secret—it is not as problematic as markings used in router-based methods. Since the secrets are unique for every connection, the danger of an attacker learning a secret is not as great. Furthermore the secrets do not interfere with fragmentation.

2.6.4. Routing Protocol Independence

Although BGP version 4 is the de-facto inter-AS routing protocol of the Internet, we cannot know how future versions will change. There are also a number of intra-AS routing protocols currently in use. It would be ideal to have a solution that would work across any routing protocol, be it BGP [58], OSPF [59], IS-IS [60], EIGRP [61], or some other routing protocol.

All host-based and combination-based defense mechanisms are routing protocol independent, but some router-based mechanisms are dependent on a specific routing protocol. IDPF relies on BGP to learn AS relationships and which AS-level paths are feasible and which are not. BASE piggybacks its own updates on top of BGP updates to send control messages and marking information. SPM does not use BGP directly, but it limits key granularity to the AS level, and routers within an AS share a server to handle all the key maintenance operations. Passport piggybacks public keys on top of BGP updates to facilitate a Diffie-Hellman key exchange, and requires routers to know the AS path outgoing packets will travel. For router-based mechanisms, only basic filtering mechanisms and SAVE can function along side any routing protocol. SPM may be modified to use prefix-level granularity, but how such a modification would alter the performance and operation of the protocol has yet to be researched. BASE and Passport may also be modified to piggyback keys and control messages on top of another routing protocol, or to send keys and control messages independently of a routing protocol, but such modifications would be changing operations central to BASE and Passport, and have yet to be researched.

TABLE 2.3. Mitigating Spoofing Attacks

Mechanism	Efficacy
Host-based systems	Poor, reaches end-host
Ingress/egress filtering	Good, filtered either by source or destination router
Martian filtering and RPF	Excellent, filtered by intermediate routers
SPM	Good, reaches target AS
Passport	Better, priority lowered by intermediate ASes, filtered by target AS
DPF	Excellent, filtered by intermediate ASes
SAVE	Excellent, filtered by intermediate routers
BASE	Excellent, filtered by intermediate ASes
IDPF	Excellent, filtered by intermediate ASes
StackPi	Poor, reaches end-host

2.6.5. Mitigating Attacks

Filtering spoofing packets may not do much good if the attacker can still achieve his goal. Even if a defense mechanism may detect and drop a spoofing packet, additional spoofing packets may keep coming. In some cases, an attack may be mitigated by a host simply by ignoring the spoofing packets. But in case the attack actually targets an end-host's bandwidth, ignoring the packets when they reach the end-host is of no use—the spoofing packets already did their damage. We now look at how well the spoofing defense mechanisms mitigate attacks. Table 2.3 shows how they compare.

In general, all of the **host-based** mechanisms fail at mitigating bandwidth-based denial-of-service attacks. The end-host may not waste resources responding to the spoofing packets, but since the packets cannot be stopped before reaching the end-host, the spoofing packets still waste the end-host's bandwidth.

IPsec is completely effective at preventing an attacker from interfering with an existing connection, and preventing an attacker from successfully initiating a connection. But it is ineffective at mitigating a bandwidth-based denial-of-service attack, and may in fact exacerbate the problem, since initializing a connection may require more resources when using IPsec than when not.

OS Fingerprinting and **IP identification field probing** can prevent a host from processing spoofing packets, but with a high cost. First, actively probing the spoofed source will require resources. Second, in order to avoid processing the spoofing packets before verifying the source, packets may require buffering. Even with this cost many spoofing packets may remain unidentified, as reported above.

TCP-specific probes offer some protection by preventing a host from processing spoofing packets. But if it is necessary to check every packet, the probing overhead could pose a problem—even more so when defending against a bandwidth-based denial-of-service attack.

IP puzzles offer some protection against attack. Targets would not waste resources processing spoofing packets, but the packets would still reach the target host. Bandwidth-related attacks would still succeed. If puzzles are sent by intermediate routers, instead of end-hosts, the spoofing packets may be detected closer to an attacker and IP puzzles could then mitigate bandwidth-based attacks as well.

Hop-count filtering can prevent the target from wasting resources responding to spoofing packets. But, since the packets have already arrived at the target, bandwidth resources would have already been wasted. Hop-count filtering does not protect against spoofing packets which attack the bandwidth of a target.

Basic router-level filtering, if deployed everywhere, would protect against nearly all spoofing attacks since any spoofing packet would be dropped at the attacker's network. If we assume spoofing packets escape the attacker's network, RPF and Martian filtering could catch the packets at intermediate routers; however, ingress/egress filtering would fail to catch the spoofing packets, unless the packets were spoofing an address from the destination router's internal network.

SPM would offer some protection to SPM-protected networks. Identifiably spoofing packets would not reach the end-hosts, but would still reach the target AS. Unfortunately this means any bandwidth-related attacks would still succeed, if the bottleneck link is an inter-AS link, not an intra-AS link.

Passport offers better protection against most spoofing attacks, assuming the attacker spoofs a source address belonging to a Passport-enabled AS, and that there are Passport enabled ASes between the attacker and destination. Note that intermediate routers merely put spoofing packets into a lower priority queue; only the last Passport-enabled AS actively drops spoofing packets. This would mitigate bandwidth-based attacks, but not entirely, since legitimate traffic from non-Passport-enabled ASes is not in the same high priority queue as Passport-verified traffic.

DPF, **IDPF**, and **BASE** could mitigate most spoofing attacks, given proper deployment locations. Spoofing packets would normally be dropped before reaching the target AS, even offering protection against bandwidth-based attacks.

SAVE would also be able to protect against all spoofing attacks, including bandwidth-based attacks, since any spoofing packet would be dropped at the attacker's network or at an intermediate router.

Pi and StackPi offer some protection against attack. Targets would not waste resources processing spoofing packets, but the packets would still reach the target host. Bandwidth-related attacks would still succeed.

TABLE 2.4. Locating Attackers

Mechanism	Efficacy
Host-based systems	None
Basic router-level filtering	Minimal
SPM	Only if attacker is in SPM-enabled AS
Passport	Only if attacker is in Passport-enabled AS
DPF	Narrow down to 5 or fewer ASes with vertex cover
SAVE	100% with full deployment Incremental deployment not analyzed
BASE	Not analyzed
IDPF	Narrow down to 28 or fewer ASes with vertex cover
StackPi	Possible if end-host has a record of the spoofing packet's marking

2.6.6. Locating Attackers

When spoofing packets are received, can the spoofing defense mechanisms identify where an attacker is located? Without being able to locate an attacker, an attacker has no risk of being caught. The more likely defenders can locate an attacker or an attacker's zombies, the less likely an attacker will risk mounting an attack. Table 2.4 shows how well the various spoofing defense mechanisms can locate an attacker.

All of the **host-based** mechanisms cannot identify where an attacker is located. When using a purely host-based defense mechanism, a spoofing packet will not contain any information providing a hint to an attacker's true location.

Basic router-level filtering would only be able to locate an attacker if the attacker's local router captured the spoofing packets.

SPM may or may not identify where an attacker is located. If the attacker is not in an SPM-protected AS, then the spoofing packets will have no marking and SPM has no way of identifying the true source. If the attacker is in an SPM-protected AS, SPM can identify the attacker's location: if the attacker's AS performs ingress/egress filtering, the attacker is located by the attacker's SPM-enabled router; if the attacker's spoofing packet reaches the target network, the SPM-enabled router there can identify the attacker's network by looking up the key embedded in the packet.

Passport could identify an attacker's location in a manner similar to SPM. If an attacker's spoofing packets originate from a Passport-enabled AS, and that AS does not drop the spoofing packet using ingress/egress filtering, a downstream Passport-enabled router may be able to identify the attacker's location. The difference is that a Passport-enabled router cannot perform a simple lookup operation on the embedded MAC. The router would have to compare the packet's MAC with MACs computed using the shared secret key with every possible source AS.

DPF cannot identify exactly where an attacker is located, but it can narrow down the possible locations. When AS-level routers deploying DPF form a vertex cover, DPF can narrow down the possible locations of an attacker to 5 or fewer.

SAVE can identify where an attacker is located, since the router at the attacker's network would catch the spoofing packet. SAVE's ability to locate an attacker without full deployment has not been analyzed, but should be similar to DPF as SAVE provides routers with exactly the information DPF requires.

IDPF functions similarly to DPF, and can narrow down possible locations of an attacker. When AS-level routers deploying IDPF form a vertex cover, IDPF can narrow down the possible locations of an attacker to 28 or fewer.

BASE also functions similarly to DPF, and should be able to narrow down possible locations of an attacker, but this aspect of BASE has not been analyzed. The ability to locate an attacker would be hindered, however, if an attacker is able to spoof BASE markings.

Pi and StackPi can possibly narrow down the possible locations of an attacker. If an end-host keeps track of which Pi markings it has seen in legitimate packets, then that end-host might be able to narrow down the location of an attacker. When a spoofing packet arrives, the end-host can identify which networks it has received packets from with the same marking as that of the spoofing packet. If the end-host never received a legitimate packet from the attacker's network, this would not work. There may also be multiple networks which produce the same Pi marking, since the markings are not entirely unique.

2.6.7. Overhead

All of the spoofing defense mechanisms require at least some level of overhead. They may incur some storage cost, computational cost, and bandwidth cost. The overhead cost of a defense mechanism must be acceptable for administrators to consider deploying it. Host-based mechanisms generally have higher overhead costs than router-based mechanisms, but this is a conscious design decision—end-hosts can afford to be more heavily loaded than routers.

Nearly all of the defense mechanisms have a storage cost. Table 2.5 shows the storage overhead of the various spoofing defense mechanisms. Cryptographic solutions such as IPsec must keep track of any negotiated keys for each protected connection. Active host-based solutions must also incur a storage cost, such as results from probing or state information regarding which hosts provided a puzzle solution. Passive host-based methods maintain information such as IP-to-hop-count mappings. Router-based methods maintain valid incoming direction or key information. Basic router filtering mechanisms such as ingress/egress filtering obtain this information essentially for "free," since a border router must already know what addresses are internal. SPM

TABLE 2.5. Storage Overhead

Mechanism	Storage Overhead
Hop-count filtering	16MB for /24 granularity hop-count tables (+ ≈20% for higher granularity clustering)
SYN cookies	Negligible
Other host-based systems	O(# of active flows)
Basic router-level filtering	Essentially none, routers already have necessary information
SPM	O(# of SPM-enabled ASes)
Passport	O(# of Passport-enabled ASes)
DPF	Unknown (no actual implementation)
SAVE	O(# of SAVE-enabled network prefixes)
BASE	16 bits per protected prefix (≈ 0.5 MB with 275,000 prefixes)
IDPF	O(size of AS-path graph)
StackPi	16 bit marking for each known IP at end-host

maintains a list of incoming and outgoing keys, one of each for every SPM-enabled AS. Passport routers maintain a list of shared secret keys and key contexts, one of each for every Passport-enabled AS. SAVE maintains interface-based incoming direction information for all address spaces, while BASE maintains marking-based "incoming direction" information for ASes. IDPF keeps track of AS relationships and incoming direction information for ASes. Pi and StackPi not only require a small amount of state in routers, but also require end-hosts to keep track of markings seen in packets from different IP addresses.

The computational cost of the defense mechanisms is more diverse. Some have only a negligible amount of computation required, such as sending out probing messages or comparing expected TTL values to actual TTL values. In contrast, IP puzzles are specifically meant to incur a computational cost on clients, acting as a deterrent to flooding the network. The cryptographic computations of IPsec also incur a high cost on communicating hosts, and in fact will decrease the number of packets a host can process during a given period of time. The computational costs of router-based and combination-based mechanisms are carefully considered by protocol designers, since nobody wants to propose a mechanism which will increase latencies in the Internet. Disregarding computation triggered by control messages (which generally only occurs during routing changes), SAVE and IDPF simply need to compare a packet's incoming direction with the valid incoming direction. Mechanisms which mark packets incur a higher cost, since routers must take time to alter the packet markings. With SPM, routers at edge networks mark keys on outgoing packets and check the keys of incoming packets. Passport has a higher cost since both edge routers and intermediate routers check packet markings. Additionally, each source router has to digitally sign all outgoing packets with multiple MACs. BASE also has a high cost since not only edge routers but any BASE router will set and check packet markings. Pi and StackPi also mark packets at each router, but each router only sets one or two bits of the marking. End-hosts using Pi or StackPi only need to map an IP address to a marking.

The bandwidth cost of the defense mechanisms is also very variable. Some mechanisms do not incur any bandwidth overhead: Pi, StackPi, IDPF, basic router filtering, and passive host-based methods. Active host-based mechanisms and other router-based mechanisms, however, incur a bandwidth cost. For example, IPsec not only increases the bandwidth needed to initiate a connection, but also adds an additional header to every packet. Passport, BASE, and SAVE use bandwidth for control messages to update key or incoming direction information. Passport and BASE piggyback on top of BGP updates and so BGP updates will increase in size. SAVE sends its own update messages separate from any routing protocol, and the bandwidth overhead is comparable to that of a routing protocol. SPM only requires

TABLE 2.6. Spoofing Defense Mechanism Recap

Mechanism	Identifying Spoofing Packets	Deploy-ability	Essential Character-istics	Routing Protocol Indepen-dence	Attack Mitigation	Locating Attackers
Hop-count filtering	⋆⋆⋆	⋆⋆⋆⋆	⋆	⋆⋆⋆⋆	⋆	-
TCP/IP Probes & OS Fingerprinting	?	⋆⋆⋆⋆	⋆	⋆⋆⋆⋆	⋆	-
SYN cookies	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆	-
IPsec	⋆⋆⋆⋆	⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆	-
IP Puzzles	⋆⋆⋆⋆	⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆	-
Martian filtering	⋆	⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	-
Ingress/egress filtering	⋆	⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆	-
RPF	?	⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	-
Passport	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆	⋆⋆⋆	⋆
SPM	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆
DPF	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	?	⋆⋆⋆⋆	⋆⋆⋆⋆
SAVE	⋆⋆⋆⋆	⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆
BASE	⋆⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆	⋆⋆⋆⋆	?
IDPF	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	-	⋆⋆⋆⋆	⋆⋆⋆
StackPi	⋆⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆	⋆⋆

bandwidth when an AS informs other ASes of its corresponding keys. Routers would send and receive around 120 kilobytes every few hours when keys are updated.

2.6.8. Recap

Table 2.6 gives an overview of how the spoofing defense mechanisms stack up against each other with respect to the above characteristics. For details on each characteristic and capability refer to the relevant sections above. Each mechanism is given zero to four stars for each characteristic and capability. We use a question mark to indicate an unknown value, and a dash to indicate a value of zero stars. For brevity we do not list ratings for overhead.

2.7. Conclusion

IP spoofing remains a severe problem in today's Internet. Not only are there still many areas where spoofing is possible, but attackers also have motivation for performing IP spoofing. Researchers have developed numerous spoofing defense mechanisms, all with advantages and disadvantages. All the spoofing defense mechanisms are able to identify some amount of spoofing traffic, but they show a variety of efficacies, including when considering their capabilities of locating an attacker or mitigating an attack.

In this chapter, we surveyed host-based solutions, router-based solutions, and their combination. None of the host-based methods can identify an attacker's location, nor defend against bandwidth-based denial-of-service attacks. They may be used to offer extra protection to a specific service, but they cannot be viewed as a final solution. They are more easily deployable than other mechanisms, and may be useful while we wait for more complete solutions to become available. Meanwhile, current router-based methods are promising yet inadequate. Basic router-level filtering, especially ingress/egress filtering, would be effective in all aspects, but its full deployment requirement is prohibitive. SAVE would also be very effective, but suffers from a similar requirement—not being able to function correctly without full deployment. SPM offers hope with the possibility of incremental deployment. It also has better attack mitigation and attacker identification capabilities than the host-based methods, but still allows spoofing traffic to reach the target AS. Passport can catch spoofing packets at intermediate routers, but in order to avoid false positives due to routing changes, intermediate routers do not drop the packets until they reach the final Passport-enabled AS. Furthermore, Passport requires BGP for key exchange, and it remains to be seen if the cryptographic calculations required by

Passport can be done at high-speed routers. DPF, along with its relatives IDPF and BASE, offer the best performance all around when incremental deployment is required. They can identify spoofing packets, narrow down possible attacker locations, and effectively mitigate even bandwidth-based spoofing attacks. However, both DPF implementations (IDPF and BASE) rely on BGP to function, affecting their portability to wherever or whenever a different routing protocol is used. Furthermore, IDPF requires routers follow a specific set of exporting rules in order to function. BASE relies on packet markings and only enables filtering after an attack is discovered in order to avoid false positives. Finally, Pi and StackPi, requiring both routers and end-hosts, brings some capabilities to locate attackers, but still cannot stop spoofing packets from reaching end-hosts.

For deploying a defense mechanism today, we can recommend some host-based mechanisms. SYN cookies should be enabled to defend against both IP spoofing and SYN flooding. However, SYN cookies only work with TCP connections, and are generally only enabled when a host's SYN queue fills up—additional protection may be required. Hop-count filtering would be a good choice for protection against spoofing with any type of IP traffic. Although the host-based mechanisms may not be as effective as a well-deployed, router-based defense mechanism, we cannot recommend a router-based mechanism at this point since there is no single mechanism clearly superior to the others.

We believe a "future-proof," router-based solution should be developed which is not only incrementally deployable, but also has no reliance on manipulatable traffic characteristics or a specific routing protocol, and is proactive without suffering from false positives. Incremental deployment is required since we cannot assume all routers, or hosts, could implement a new protocol at the same time. Note that incentive is particularly necessary for incremental deployment—if the incremental benefit of deploying a defense mechanism is too low, networks will be hesitant to deploy it. Defenses cannot rely on traffic characteristics that an attacker can easily manipulate and spoof the correct values—intelligent attackers may be able to pass through the defense mechanism. Routing protocol independence is required since, although BGP is the de facto routing protocol in the Internet today, there is no guarantee that will always be the case. Relying on BGP could also deny protection to most intra-AS networks. It would be a mistake to assume that IP spoofing only happens at the inter-AS level. And finally, since an attack can happen at any time, it is not acceptable to activate a solution only after an attack is detected. The system should automatically accurately detect and effectively mitigate an attack, instead of waiting for instructions to detect and mitigate an attack.

One recent trend some researchers are looking into is attempting to piece together different existing works. For example, Source Address Validation Architecture [62], or SAVA, uses different mechanisms at different levels to address IP spoofing in IPv6. At the subnet level, every IP address is bound to a specific MAC address and switch port; Within an AS, SAVA requires ingress/egress filtering; Between ASes, SAVA is either similar to IDPF by building filtering rules between SAVA-enabled ASes, or similar to SPM by embedding a signature in packets when non-SAVA-compliant ASes are involved.

Whereas we have yet to see if simply using different mechanisms in different situations is the best way to create effective defense mechanisms, there is no question we should build upon the lessons learned from past research. Borrowing ideas and methods from each of these mechanisms may prove beneficial to future development of improved defenses. As a starting point for a future spoofing defense mechanism, we suggest taking concepts from two existing works: DPF and SAVE. A DPF-like system will work well with incremental deployment, and a SAVE-like system can provide routers the necessary information to create DPF tables—without relying on any specific routing protocol and without using any manipulatable characteristics. The following chapter will present such a system, building off of SAVE and DPF.

CHAPTER III

A NEW SOLUTION FOR IP SPOOFING DEFENSE

Text from this chapter was published in the *Proceedings of the Conference on Security and Privacy in Communications Networks*, in September, 2010 [19]. Patrick McDaniel assisted in editing and designing some earlier versions of the work. My professor, Jun Li, and I were the principle investigators for this work.

Now that we understand what's missing from the current state of the art IP spoofing defenses, we propose a new solution. Our solution meets all of the requirements we analyzed in Chapter II: It is effective in identifying spoofing packets, can mitigate spoofing attacks, assists in pin-pointing an attackers true location, is resilient against circumvention, is easily deployable, is routing protocol independent, and has low overhead.

This chapter is organized as follows. First we will review the necessity for a new spoofing defense solution. Then we will describe the basics of the earlier version of SAVE [48, 63], which our solution builds upon. With a basic understanding of SAVE, we will describe the new mechanisms we created which enhance SAVE's ability to meet all of the requirements from Chapter II. We finish the chapter with an analysis of our enhanced version of SAVE.

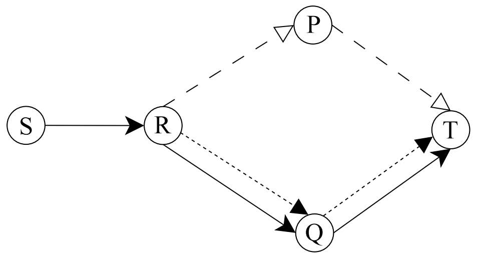
3.1. Need for a New Spoofing Defense

If every network in the world was able to coordinate a deployment of even simple ingress filtering [39] and unicast reverse path forwarding [41] checks, the threat of IP spoofing would be all but eliminated. Unfortunately due to both technical and logistical reasons, this is an unattainable goal [18]. When even a small percentage of networks do not deploy such basic filtering methods, nobody is safe. Everyone's Internet address is still at risk of being spoofed. Researchers have proposed more sophisticated spoofing prevention mechanisms over the years [18], but all have failed to neutralize the threat of IP spoofing.

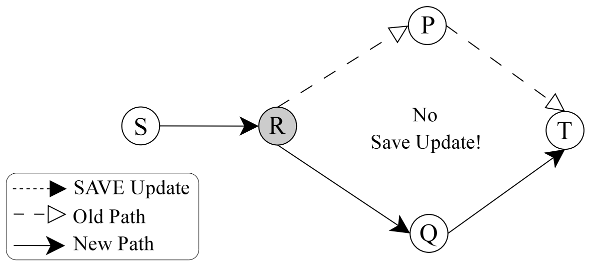
Fortunately there has been promising research, showing that if even a *small* percentage of routers on the Internet deploy a more finely grained filtering table to discard packets with a forged source address, a synergistic filtering effect can be achieved to stop a large fraction of spoofed IP packets [11, 64]. SAVE [63, 48] is a light-weight protocol to build such a filtering table, called an *incoming table*, at routers. As a router has multiple physical interfaces to receive incoming packets, every entry of the incoming table specifies the valid incoming interface for packets from a specific IP address prefix to arrive at the router. Chapter II further showed that compared to other IP spoofing prevention methods, using an incoming table to filter spoofed packets is the most effective.

Although SAVE provides the incoming tables necessary for effective filtering with only a small deployment, the SAVE protocol itself faces a serious challenge when incrementally deployed. A router's incoming table that is up-to-date at time t may become obsolete at some time after t because of routing changes on the Internet. We describe how a router's incoming table becomes obsolete below.

In the SAVE protocol, every SAVE-capable router that is in charge of a source address space *periodically* sends updates to its downstream routers about the *current* incoming interface for the source address space. Furthermore, when a routing change occurs at any downstream router, that router must send out a new update immediately to ensure routers further downstream learn the new valid incoming interface of the source address space in question. Otherwise, those routers will stay out of date until the next periodical update. However, legacy routers that do not run SAVE will simply do nothing when a routing change occurs; legacy routers will never initiate a SAVE update! As shown by the example in Figure 3.1a and 3.1b, if R was a legacy router, the lack of SAVE update from R after its routing change will cause the SAVE-capable router T's incoming table about packets from router S to be out of date (until the next periodic SAVE update from S reaches T).



(a) R is a SAVE router and initiates a SAVE update when the link from R to P is broken. T's incoming table is updated to show that packets from S should come from its lower left incoming interface.



(b) R is a legacy router so it does *not* initiate a SAVE update when the link from R to P is broken. T's incoming table is out of date, still showing packets from S should come through upper left incoming interface.

FIGURE 3.1. An example showing how a router's incoming table can contain obsolete entries.

As SAVE is incrementally deployed, there will be many legacy routers, probably even outnumbering SAVE-capable routers for a long time; *it is highly likely that incoming tables at SAVE-capable routers will often contain obsolete entries.* While it is promising to use incoming tables to stop spoofed packets, it is also difficult to use them if they carry obsolete entries. While SAVE scores well against the requirements of a good spoofing defense when not considering partial deployment, it falls apart when partially deployed. In a partial deployment scenario SAVE is no longer effective, unable to mitigate attacks well, and unable to locate an attacker.

This chapter makes the following fundamental contributions: We study if we can introduce new mechanisms to enable SAVE-capable routers to reliably discard spoofing packets, even though their incoming table may be obsolete. In particular, we devise and evaluate three new elements that a SAVE-capable router can employ: a blacklist data structure, an on-demand-update mechanism, and a pushback mechanism.

– *Blacklist*: The blacklist complements the incoming table at a router in classifying an incoming packet, including determining if the packet is spoofed.

– *On-demand update*: A SAVE-capable router can request SAVE updates on demand to verify possibly incorrect or outdated information.

– *Pushback*: SAVE-capable routers along the way of a spoofing flow can push the filtering of spoofed packets to the router that is the closest to the spoofer.

In combination, these new elements allow SAVE to function properly even in the presence of legacy routers. Referring back to the example in Figure 3.1b, T can request an on-demand update from S, essentially replacing the triggered SAVE update in Figure 3.1a. The blacklist gives a router more state information, so the router does not need to request an on-demand update for every packet that does not match the incoming table. Finally, the pushback mechanism serves two purposes. First, it helps to reduce spoofing traffic by dropping spoofing packets as close to the attacker as possible. Second, it tells a router in charge of a source address space when downstream routers have incorrect information in their blacklists regarding its address space.

Also of great importance is the security of SAVE with these new mechanisms. SAVE must secure itself against all possible attacks. Not only may attackers try to evade the IP spoofing detection at SAVE routers, they may also attempt to introduce illegal control messages. For example, an attacker (or a bot machine it controls) could try to establish wrong incoming information at SAVE routers by injecting a SAVE update about a source address it is going to spoof. In this paper, we also discuss how security can be addressed.

Our evaluation demonstrates the viability of these new mechanisms. We perform a detailed simulation to evaluate their effectiveness at detecting spoofing packets, and explore the relationship between efficacy and adoption rates. These Internet-scale simulations show that with as little as 0.08% deployment, attackers cannot spoof protected source addresses in over 90% of all cases. Moreover, we evaluate the storage, traffic, and computational overhead of SAVE with the new mechanisms, showing that the overhead is low.

The rest of this chapter is organized as follows. We first describe the original SAVE mechanisms, including the incoming table and incoming tree. Then we show how a SAVE router can introduce a blacklist alongside the incoming table to help classify incoming packets in Section 3.3. We then describe the on-demand-update mechanism and the pushback mechanism in Section 3.4 and 3.5, respectively. Section 3.6 discusses how SAVE can secure itself. We present our evaluation in Section 3.7.

3.2. Original SAVE System

The original SAVE system provides routers the mechanisms to efficiently learn the correct incoming direction of packets. This is done by having SAVE routers send SAVE updates to downstream routers, both periodically and whenever there is a routing change. Each update represents the group of source addresses, or the source address space, that the originating SAVE router is responsible for. The downstream routers use the incoming updates to create and maintain their incoming tables and incoming trees.

3.2.1. SAVE Updates

SAVE routers send out two kinds of updates: periodic updates and updates triggered by routing changes. Periodic updates are periodically sent out for every address space in a router's forwarding table, and updates triggered by routing changes are sent out only for those address spaces affected by the routing change. Both types of updates function identically once the originating SAVE router sends it out. An update contains an Address Space Vector (ASV) and the destination address space. The ASV is of the form $\langle S_1, S_2, \cdots S_N \rangle$ where S_i is the address space associated with router i. When a router initiates a SAVE update, the ASV contains only the address space of the source router, and the destination address space equals an entry from the source router's forwarding table. The source router forwards the update towards the destination address space, and the next SAVE router that receives the update processes the update as follows:

1. Record the incoming direction of the SAVE update as the incoming direction of any packets from the address spaces listed in the ASV.

2. Append it's own address space to the ASV.

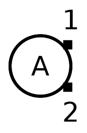
3. Forward the update towards the destination address space listed in the update. If the router's forwarding table has multiple entries that match the destination address space, it will split the update into multiple updates with each destination address space being the intersection of the original address space and the matching forwarding table entry.

Therefore, as a SAVE update travels through SAVE routers, the ASV will show the path the update travelled from the first address space in the ASV to the current location. The update will have passed through the router in charge of each listed address space, in the order listed. If an update originated at router X, then passed through router Y, then Z, the ASV would be $\langle S_X, S_Y, S_Z \rangle$ when it arrived at the next router downstream from Z.

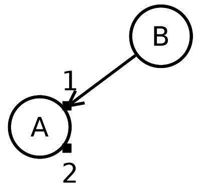
3.2.2. Incoming Tree

A router uses its incoming tree to create and maintain its incoming table. The incoming tree stores all the information about which incoming direction packets from a given source address space should arrive on. The tree is built using the SAVE updates described above. For a given SAVE router, the root of the tree represents that SAVE router. All other nodes in the tree represent an upstream SAVE router and its associated address spaces. Which link a child node uses to connect to the root represents the incoming direction of packets from the corresponding address spaces. Figure 3.2 shows how an incoming tree is constructed and maintained.

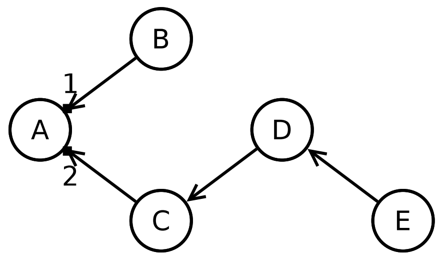
Consider router A pictured in Figure 3.2. Initially, router A has no incoming direction knowledge, so A's incoming tree is empty except for a root node representing itself (Figure 3.2a. A then receives a SAVE update from incoming direction 1. It is



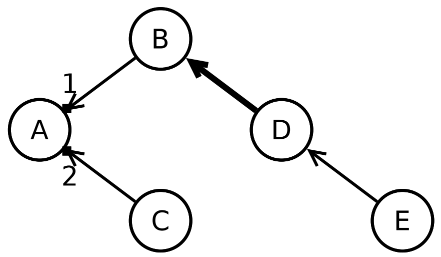
(a) Initially the incoming tree at router A is empty. A has two incoming interfaces, or incoming directions.



(b) Router A receives a SAVE update on interface 1 from router B with ASV $= \langle S_B \rangle$.



(c) Router A receives a SAVE update on interface 2 from router C with ASV $= \langle S_E, S_D, S_C \rangle$.



(d) After a routing change at router D, router A receives a SAVE update on interface 1 from router B with ASV $= \langle S_D, S_B \rangle$.

FIGURE 3.2. Constructing and maintaining an incoming tree at router A.

an update from router B, with an ASV containing only B's source address space S_B. A adds B to its incoming tree with a link from B to A's incoming interface 1 (Figure 3.2b. At a later time, A receives another SAVE update from incoming direction 2. It is an update from router C, with ASV $= \langle S_E, S_D, S_C \rangle$. A adds C to it's incoming tree with a link from C to A's incoming interface 2, then links D to C and E to D (Figure 3.2c. At a later time, router D makes a routing change. Router D then generates an update message and sends it downstream. When A receives the update from B, the ASV is $\langle S_D, S_B \rangle$. Note, that the incoming interface between A and B did not change. But A sees that D now forwards packets through B instead of C, and changes the incoming tree to reflect the new topology (Figure 3.2d. All routers which forwarded traffic through D will also be affected by the routing change. Since A keeps track of the links between routers in the tree, A knows that the entire sub-tree rooted at the router which had a routing change will use the new incoming interface. This way, A will learn the new incoming direction of packets from other routers affected by the routing change even if they do not send updates. In this case, since E was previously upstream from D, A knows that packets from E's address space will also be affected by the change.

3.2.3. Incoming Table

The **incoming table** contains the incoming direction information for protected source address spaces. It does not contain more information than the incoming tree, since it is built from the incoming tree, but it is built to be faster and more compact than the incoming tree. Recall routers are highly optimized for table lookup functions so putting incoming direction information into a table that can be looked up according to the source address allows SAVE to take advantage of this optimization. This helps to minimize the computational overhead associated with SAVE. Furthermore, in cases where routing symmetry exists, SAVE takes advantage of this symmetry: If the *incoming* direction *from* a given address is the same as the *outgoing* direction *to* that address, the router does not need to store the information twice. Instead of creating incoming table entries for *every* source address space, a router only creates entries for those whose incoming direction is different from the outgoing direction. The router can set a flag in its forwarding table to indicate which address spaces have entries in its incoming table. This way, when looking for the incoming direction for an address space, the router can first check the forwarding table, and then, only if necessary, consult the incoming table. Figure 3.3 shows an example incoming table.

The incoming table can be built from the incoming tree. Whenever the incoming interface for a node in the incoming tree changes, the router can update that node's incoming table entry along with all nodes upstream from that node.

source address space	incoming direction
S_k	id_k

FIGURE 3.3. Incoming table entry (to identify legitimate packets).

3.3. Blacklist and Packet Classification

In addition to the legacy SAVE mechanisms such as the incoming tree and incoming table described above, We also add to every SAVE-capable router a blacklist data structure. With both the incoming table and the blacklist, a router classifies incoming packets into several different types, and takes some action specific to each of those types. In this section we describe the blacklist data structure, and the packet classification mechanism.

3.3.1. Blacklist

Whereas the incoming table of a router specifies the legitimate incoming interfaces for different source address spaces, the blacklist indicates whether an incoming packet is *spoofing* based on its source address, destination address, and the incoming interface. More specifically, a router's blacklist is maintained through two separate blacklists corresponding to two different ways of matching an incoming packet against a blacklist entry:

SI: Matches the source address and the incoming interface of the packet.

SD: Matches the source address and the destination of the packet.

A router receiving spoofing packets that match its SI blacklist could be on the legitimate path from the spoofed source to the destination, but not from the spoofing packet's incoming direction. A router receiving spoofing packets that match its

SD blacklist should not be on the legitimate path from the spoofed source to the destination, and so should never see a packet with such a source and destination.

3.3.2. Packet Classification

With both its incoming table and blacklist in place, a router classifies an incoming packet as described in Figure 3.4 and below:

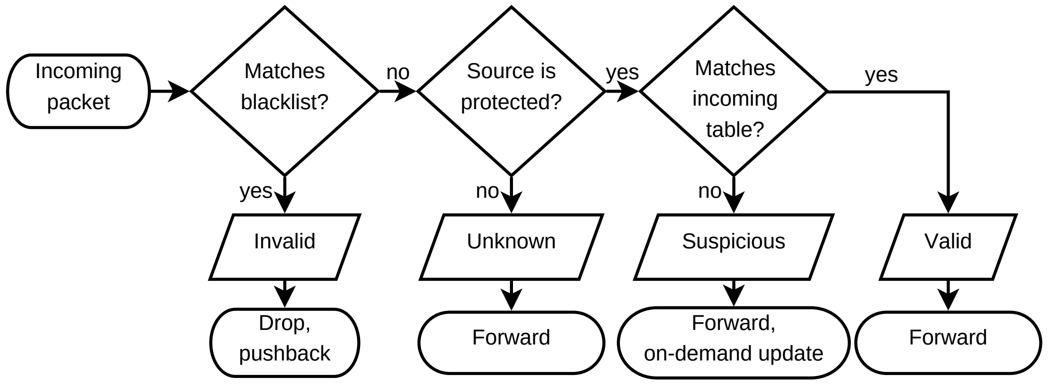


FIGURE 3.4. Incoming packet classification at a SAVE router.

– *Valid* if it matches the incoming table but not the blacklist.

– *Suspicious* if it matches neither the incoming table nor the blacklist.

– *Invalid* if it matches the blacklist.

– *Unknown* if there is no information regarding the packet's source address.

Only when the packet is classified as invalid will the router drop the packet. For the other three types the router will forward the packet.

Furthermore, if a packet is suspicious, the router will initiate the on-demand-update mechanism. The packet is suspicious either because the packet is spoofing, or because the packet is legitimate but the router's incoming direction information is outdated. As we described in Section 3.1, the routing change at a legacy router upstream will not lead to an immediate SAVE update for this router to update its incoming table.

If a packet is invalid, the router will initiate the pushback mechanism. No further actions are taken for valid or unknown packets.

We describe both on-demand-update and pushback in the following sections, including how they deal with obsolete incoming table entries.

3.4. On-Demand Update

When a router classifies a packet as suspicious, it still forwards the packet as usual, but it will also initiate an on-demand update. From the incoming table entry that matches the packet's source address, the router determines the source address space in question and the source router in charge of the source address space. It then requests that the source router sends an on-demand update—which is on behalf of the entire source address space—towards the destination of the suspicious packet.

Following the same design as in SAVE [63, 48], the on-demand update will travel the same path as the legitimate packets that originate from the source router's address space. When the on-demand update arrives at the router from a specific incoming interface, this interface is then also the legitimate interface for the source address space in question. The router then makes sure its incoming table records *this* interface as the legitimate incoming interface for the source address space.

If the on-demand update does *not* arrive from the same incoming interface as the suspicious packet, the suspicious packet was in fact spoofing. Furthermore, the router updates its blacklists. Denote the spoofing packet's spoofed source address *space* as S and its incoming interface as i. It adds to the SI blacklist a new entry

$\langle S,\ i \rangle$. If in the future a packet matches the newly added blacklist entry, the router will classify it as invalid. The router does not add a new entry to the SD blacklist, because the router could legitimately see packets from the suspicious packet's source to its destination—just not from the incoming interface that the suspicious packet used.

If the on-demand update *does* arrive from the same direction as the suspicious packet, the packet was not spoofing. Note the router already forwarded the packet earlier so no false positive occurs.

It is also possible the on-demand update never reaches the router. This could be because the router is not on the path from the source to the destination, or because congestion caused the update request or the update itself to be dropped. Since the router cannot know for sure, it takes no action. Assuming similar suspicious packets continue to arrive, the router will continue to request updates. We use a truncated binary exponential backoff scheme for subsequent requests.

3.5. Pushback of Spoofed Packets

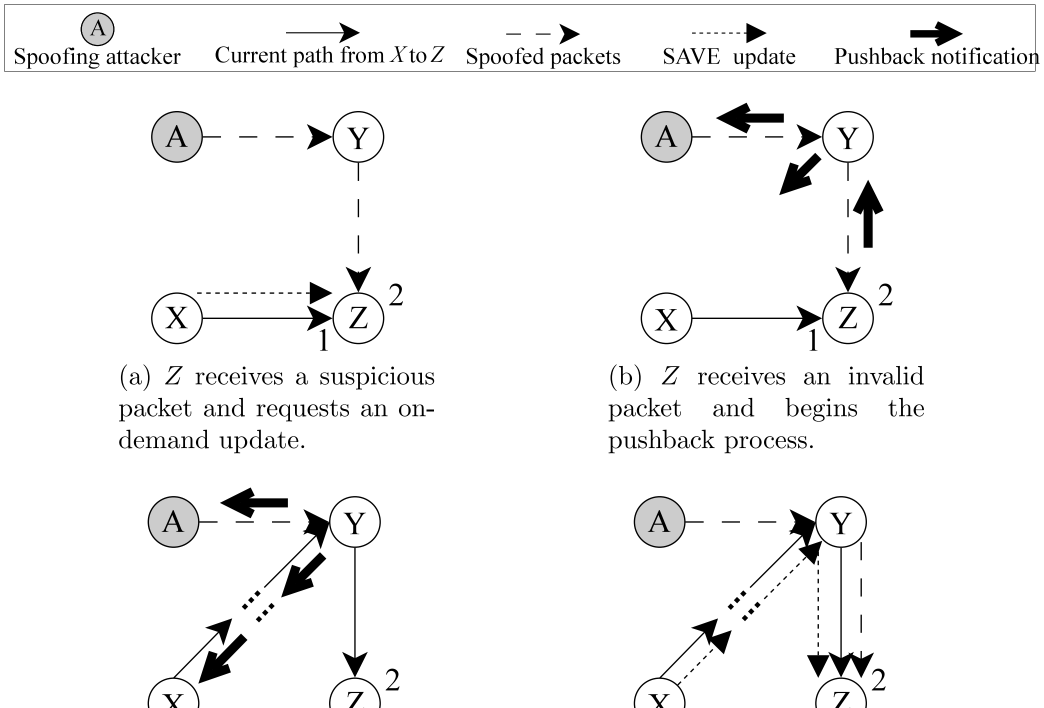
The aim of the pushback is to push the filtering of spoofed packets all the way toward the router that is the closest to the spoofer(s). The pushback procedure is packet-driven and it is triggered when a SAVE router receives an invalid packet.

Once the pushback procedure is triggered by an invalid packet, the router sends pushback messages to immediate upstream SAVE routers that the packet possibly passed through. (The router uses incoming SAVE updates to record upstream SAVE routers along every incoming interface, and can easily identify those upstream along the incoming interface of the packet.) Assume the packet is from source address space S to destination address d. When an upstream SAVE router receives a pushback message, it adds an entry $\langle S,\ d\rangle$ to its SD blacklist. The incoming interface does not matter—the upstream router is not on the legitimate path from the packet's inscribed source to its destination at all. Upon receiving subsequent packets that match this new blacklist entry, the upstream router will classify them as invalid and continue to propagate the pushback further upstream.

Blacklist entries can become outdated if a routing change causes the legitimate path from the spoofing victim to become the same as that of the spoofing packets. If that happens, the pushback procedure will finally reach the source router in charge of the victim source address space. The source router can in turn send out an update that travels along the path and reaches every SAVE router en route. Every SAVE router can then remove its outdated blacklist entries.

Figure 3.5 shows a pushback example. An attacker at legacy router A sends spoofing packets with a source from space S_X (X's source address space) and a destination dst_Z (an address towards which Z is downstream from X and Y). The spoofing packets arrive at router Y along the same interface as legitimate packets from S_X. Y classifies the packets as valid and forwards them. Z however expects packets from S_X to arrive on interface 1 according to its incoming table, so it classifies the first spoofing packet arriving at incoming interface 2 as suspicious. Z requests an on-demand update. Upon receipt of the requested update, Z confirms its incoming table information was correct, and the suspicious packet was in fact invalid (Figure 3.5a). Z then adds a new entry to its SI blacklist: Based on source S_X and incoming interface 2 of the suspicious packet, the new blacklist entry is $\langle S_X,\ 2\rangle$.

Z classifies later spoofing packets as invalid, and initiates the pushback process (Figure 3.5b). Z knows Y is its neighbor along the spoofing packet's incoming interface. Z sends Y a pushback message, instructing Y to add an entry to its



(a) Z receives a suspicious packet and requests an on-demand update.

(b) Z receives an invalid packet and begins the pushback process.

(c) After routing change, legitimate packets trigger pushback to X.

(d) X discovers legitimate packets are misclassified and sends an update.

FIGURE 3.5. A pushback example. An attacker, A, sends packets spoofing X's address space, S_X, towards dst_Z.

SD blacklist for all packets from S_X to dst_Z. When Y receives a packet matching the new blacklist entry, it classifies the packet as invalid and continues the pushback. Y finds all neighbors along the spoofing packet's incoming interface, and propagates the pushback towards such neighbors. Y's neighbors do not see matching packets, so do not further propagate the pushback. Y is the closest SAVE router to the attacker.

Later, if there is a routing change at a legacy router which causes the originally invalid "$X \cdots Y \cdots Z$" path to become valid and legitimate packets begin to flow along the path, SAVE will quickly converge to correct the error. During the transient period, router Y and Z will misclassify valid packets from S_X towards dst_Z as invalid. But now that Y's upstream neighboring SAVE routers also see packets matching the pushback request, the upstream routers will relay the pushback all the way to the source router X (Figure 3.5c). After X receives the pushback, it realizes that downstream routers have incorrect blacklist entries matching its legitimate traffic. X sends an update towards dst_Z, causing all routers along the newly valid "$X \cdots Y \cdots Z$" path to remove their incorrect blacklist entries (Figure 3.5d).

3.6. Security Considerations

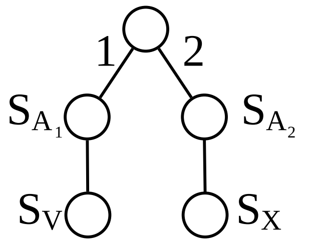
SAVE must also be secure. The security of SAVE encompasses securing SAVE itself against attack and keeping attackers from being able to use SAVE to launch attacks. In addition to basic security functions such as confidentiality, integrity, and replay prevention, we must consider (1) *origin authentication* to ensure a router is authorized to speak for a source address space, and (2) *collusion prevention* to ensure attackers cannot collude to manipulate incoming direction information at SAVE routers.

3.6.1. Origin Authentication

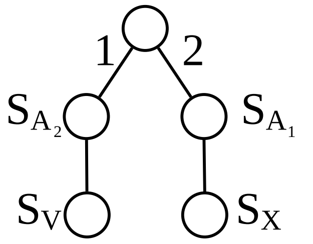
Origin authentication ensures SAVE routers are authorized to speak for their corresponding source address space. This requires a trusted authority to sign a certificate that an address space owner can present. A public key infrastructure as described in [65] can provide such certificates of address ownership. The root certificate authority can be ICANN, with regional Internet registries such as ARIN or RIPE at the next level, and ISPs below. If SAVE is simply deployed inside an AS, the AS can simply use its self-signed certificates.

3.6.2. Collusion Prevention

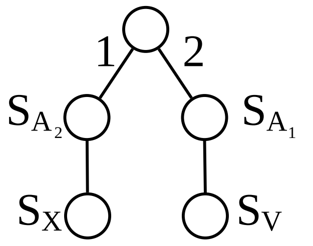
Attackers may attempt collusion in order to manipulate the incoming tree or incoming table of a SAVE router. They may collude by masquerading as each other or copying an update from an upstream space to each other, causing downstream routers to receive the update about a source address space along a wrong incoming interface. Using Figure 3.6 as an example, and assuming SAVE router R initially maintains a correct incoming tree (Figure 3.6a), we show that attackers (A_1 and A_2) have three choices of collusion strategies to manipulate R's incoming tree (thus also R's incoming table):



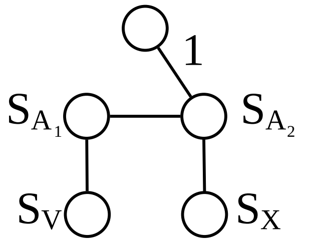
(a) R's correct incoming tree.



(b) R's incoming tree after A_1 and A_2 masquerade as each other when forwarding updates.



(c) R's incoming tree after A_1 and A_2 masquerade as each other when initiating updates.



(d) R's incoming tree after A_1 appends A_2's address space to a passing update's address space vector.

FIGURE 3.6. R's incoming tree. Attacker A_1 is upstream along R's interface 1, while A_2 is upstream along interface 2.

1. A_1 and A_2 masquerade as each other when *forwarding* updates. (They can masquerade as each other by sharing their private keys.) In this case, S_{A_1} and S_{A_2} would swap positions in R's incoming tree. However, since the update from V still arrives at R's interface 1, and the update from X still arrives at R's interface 2, this swap does not affect other source address spaces: they would still be associated with the correct incoming direction (Figure 3.6b).

2. A_1 and A_2 masquerade as each other when *initiating* updates. In this case, not only would S_{A_1} and S_{A_2} change positions in R's incoming tree, but all nodes below S_{A_1} and S_{A_2} would also change positions. Specifically, S_V's incoming direction would become interface 2, and S_X's incoming direction would become interface 1 (Figure 3.6c).

3. A_1 appends A_2's address space to a passing update. This causes the route from V to R to appear as V, A_1, A_2, and arrives via R's interface 1. S_{A_2} and all nodes below S_{A_2} will appear behind interface 1 in R's incoming tree. S_V's incoming direction information would remain valid, but S_X's incoming direction would be incorrectly changed to interface 1 (Figure 3.6d).

In defending against collusion strategies 2 and 3 (we do not consider strategy 1 as it does not affect benign source address spaces), a key fact is that, because the incoming direction is based on the physical incoming interface, attackers cannot change the incoming direction of updates. Thus, R can correct the incoming direction information for a node, e.g., S_X, when that node's router sends or forwards an update towards R. Instead of waiting for such an update to occur, R can proactively request an on-demand update from a router, e.g., X, when its source address space is moved to a different incoming interface. R should not ask the router corresponding to the root of the subtree that changed interfaces since, if the change was malicious, that router is likely to be a colluding attacker. Instead, R can request the update from a router corresponding any node below the root of the subtree that changed interfaces.

If R's verification discovers that updates from A_1 and A_2 often cause its incoming tree to become incorrect, R may suspect A_1 and A_2 are malicious and begin to ignore any updates they initiate. Updates that pass through A_1 and A_2 would still be processed, but address space vector entries corresponding to the attackers would be ignored. R would essentially remove the attackers' source address space from its incoming tree, and S_X and S_V would be direct children of the root of R's incoming tree.

3.6.3. Confidentiality, Integrity, and Replay Prevention

When considering confidentiality, we must examine what messages need to be confidential. SAVE messages include updates, on-demand update requests, and pushback notifications. Some messages cannot be confidential: an update must be understandable by whichever downstream router receives it, and the identity of the downstream router cannot be known ahead of time. The remaining messages can be encrypted for confidentiality.

SAVE uses public key cryptography for encrypting messages; in other words, when a SAVE router sends a message, it uses the public key of the recipient to encrypt the message. No secure channel is set up. This makes sense because a SAVE router may need to communicate with many other peers over the Internet, instead of just physically neighboring routers, and communications with them are often short-lived.

3.6.3.1. Integrity

Verifying the integrity of a SAVE control message is straightforward. We showed above that public key cryptography should be used for confidentiality, and we can use public key cryptography to create digital signatures for the control messages as well.

3.6.3.2. Replay Attacks

Replay attacks must be prevented in order to ensure that a previous update cannot be copied and resent at a later time. Downstream routers must receive the most up-to-date incoming direction information. Replay attacks can be prevented by adding an ID number to control messages. The ID number of later control messages should be greater than the ID number of earlier control messages. If some router, X, receives a control message supposedly from another SAVE router, Y, with the same ID number twice, or with an ID number lower than the most recent message, then X knows the message was replayed and will ignore the replayed message.

3.6.4. An implementation note

The security issues that SAVE faces are similar to those faced by BGP. Both need to ensure that a router can speak for an address space (SAVE's source address space and BGP's destination address space), both need to prevent conclusion of attackers, and both need to provide integrity, replay prevention, and sometimes confidentiality. In particular, to implement SAVE's security, we can borrow some ideas from IRV [44], an incrementally deployable BGP security solution. Basically, each network can contain a *validation server* to be responsible for security purposes, including keeping track of certificates and keys, performing signature creation and validation for SAVE messages, and managing security policies. Doing so would also maintain a lighter load on SAVE routers, allowing them to focus on its main purpose of receiving, validating, and forwarding packets.

3.7. Evaluation

In this section we discuss the performance of SAVE with the new mechanisms we introduced in this paper. First we present SAVE's efficacy in catching spoofed packets. We then evaluate SAVE's false positives. Finally we show that SAVE's storage, network, and computational overhead are reasonable.

3.7.1. Efficacy

3.7.1.1. Methodology

For efficacy evaluation we use a modified static distributed packet filtering (DPF) [11] simulator. The DPF simulator allows us to evaluate the efficacy of SAVE on Internet-scale topologies by calculating efficacies based on the Internet AS graph and SAVE router locations. It uses Internet Autonomous System (AS) topologies from Route Views [66]. The efficacy metrics are similar to those in [11]. Specifically, we report:

- $\Phi_2(1)$ that represents the percentage of ASes that an attacker cannot send spoofing packets from—any spoofed packets from those ASes would be detected and filtered out;
- $\Phi_3(1)$ that represents the percentage of all attacker-victim AS pairs where the attacker *cannot* send spoofed packets to the victim; and

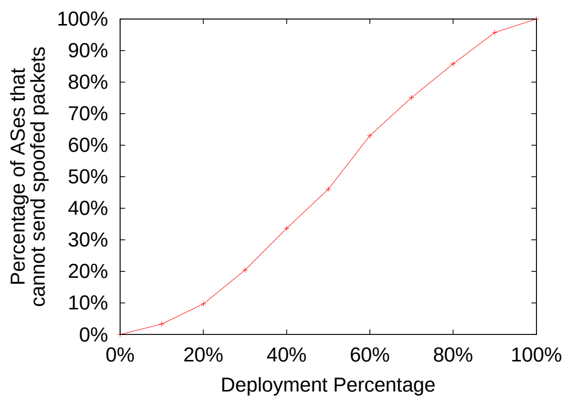
– $\Psi_1(\tau)$ that represents the fraction of target ASes which can narrow down an attacker's location to within τ possible attack ASes.

We consider a variety of placement strategies of SAVE routers. First, we deploy SAVE routers so they form a vertex cover (as in the original DPF work [11]). Then, we look at random deployments, with deployment percentages between 0% and 100% in 10% increments. Finally, we deploy SAVE routers at the top ASes by degree.

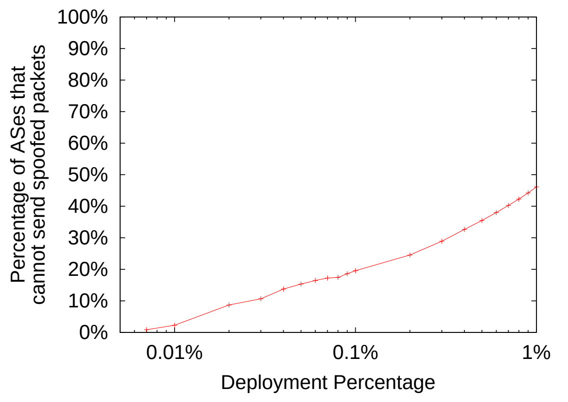
Results and Analysis. The efficacy of SAVE in catching spoofed packets depends upon both the deployment strategy and the percentage of deployment. With a random deployment, the efficacy increases along with the deployment percentage. With deployment at high-degree ASes or using a vertex cover for deployment, the efficacy is much higher than a random deployment, even with a much lower percentage of deployment.

Figure 3.7 shows $\Phi_2(1)$, the percentage of ASes on an Internet AS topology from which an attacker cannot send *any* packets that spoof a protected source address. We can clearly see that deployment strategies are an important factor. $\Phi_2(1)$ with a vertex cover deployment is around 99% (not shown), where the vertex cover consisted of around 14.5% of all routers. Figure 3.7a shows $\Phi_2(1)$ for random deployments; with 15% or less deployment percentage, $\Phi_2(1)$ is even no more than 10%. Figure 3.7b shows $\Phi_2(1)$ for deployments at ASes with the highest degree on the same topology; as a sharp contrast to random deployment, even with less than 1% of ASes deploying SAVE, over 40% of all ASes are unable to spoof *any* protected source.

Figure 3.8 shows $\Phi_3(1)$, the percentage of attacker-victim AS pairs where the attacker cannot send spoofed packets to the victim. Deployment strategy, again, plays an important role. A random deployment is not very effective—high efficacy requires high levels of deployment. More targeted deployments, however, can be



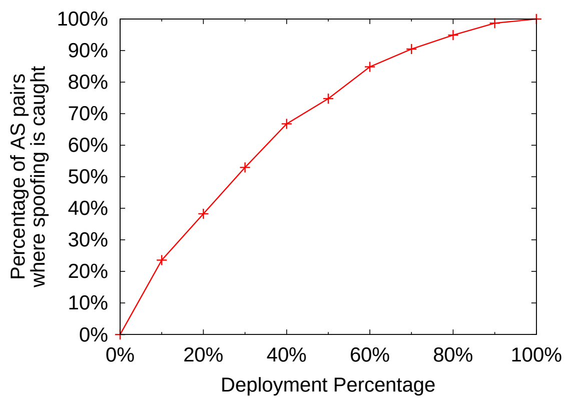
(a) Random deployments.



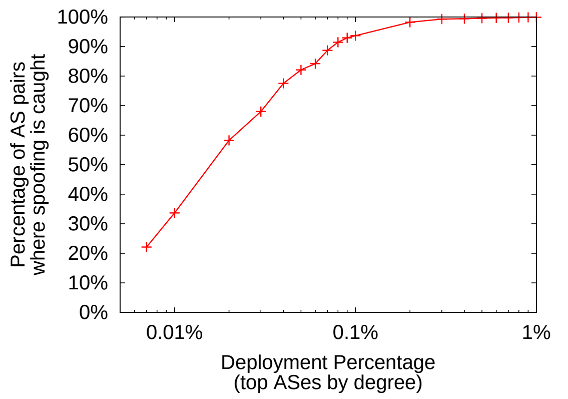
(b) Deployments at highest-degree ASes.

FIGURE 3.7. $\Phi_2(1)$: The percentage of ASes on an Internet AS topology from which an attacker cannot send spoofed packets.

extremely effective. With a vertex cover deployment the efficacy is over 99.9% (not shown). Even very small targeted deployments can be effective: With deployment at only the top 0.08% of ASes by degree (21 ASes in this case), efficacy is over 90%. This is due to the hierarchical nature of Internet routing, but if a group of core Internet routers were able to get together and start the deployment of SAVE it would bring great benefit to them and their customers. Furthermore, it would allow other early adopters to also reap the benefits of such a highly effective deployment, even if the other early adopters are not also at the core of the Internet.



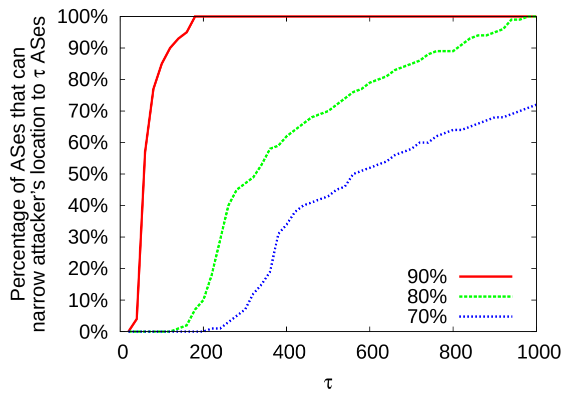
(a) Random deployments.



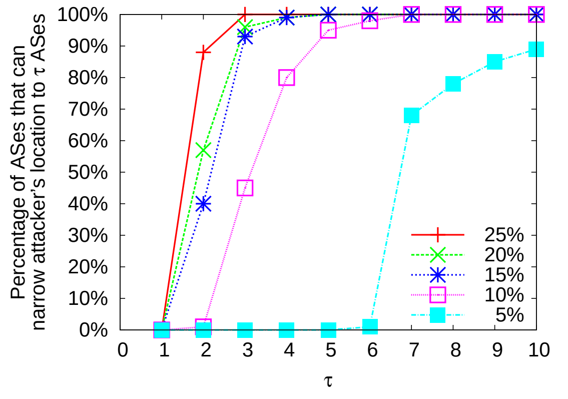
(b) Deployments at highest-degree ASes.

FIGURE 3.8. $\Phi_3(1)$: The percentage of attacker-victim AS pairs where the attacker cannot send spoofed packets to the victim.

Figure 3.9 shows $\Psi_1(\tau)$ with again the same deployments as above. $\Psi_1(\tau)$ is the percentage of destination ASes that can narrow down an attacker's location to within τ source ASes, when intermediate routers were unable to filter the spoofed packet. Note that this percentage is for instantly narrowing down an attacker's location based on the network topology, the location of SAVE routers, and the fact that the spoofed packet reached its destination. Vertex cover deployments (not shown) and high-degree AS deployments have excellent performance, generally being able to narrow down an attacker's actual location to within 5 locations or fewer. More random deployments are not able to reliably narrow down an attacker's location, with possible attacker locations measured in the hundreds. (SAVE includes additional location capabilities through the use of its pushback mechanism, which we plan to evaluate further.)



(a) Random deployments.



(b) Deployments at highest-degree ASes.

FIGURE 3.9. $\Psi_1(\tau)$: The percentage of destination ASes that can narrow down an attacker's location to within τ source ASes.

Finally, the high efficacy with the highest-degree ASes (as shown in Figures 3.7b, 3.8b, and 3.9b) shows that such ASes—if they deploy SAVE—can filter spoofing packets and locate attackers very effectively. They thus will have a strong incentive to deploy SAVE. Moreover, doing so provides an incentive for other ASes to follow; with the highest-degree ASes deploying SAVE, when the followers also deploy SAVE they will protect their own address space much more effectively.

3.7.2. False Positives

False positives only occur when the following conditions are *all* met:

– A pushback path that spoofing packets travel becomes a valid path due to a sudden underneath routing change;

– Legitimate packets begin to flow along the path;

– The SAVE update from the source router of the legitimate packets has not reached SAVE routers along the path to void the blacklist entries that cause the legitimate packets to be dropped.

The transient time that all three conditions are met is short-lived. Assume S is the source router and R is a SAVE router on the path. False positives will occur at R from the time the first legitimate packet arrives along the new valid path to the time R's blacklist is updated. Assuming there is a steady stream of packets from S passing through R, it will take the following amount of time to update R's information:

$$rtt + \sum_{i=next(S)}^{R} C_i + \sum_{i=S}^{prev(R)} P_i + \sum_{i=next(S)}^{R} U_i \qquad \text{(Equation 3.1)}$$

rtt is the round trip time between R and S, $next(S)$ is the SAVE router downstream from S towards R, C_i is the time it takes for router i to classify a packet, $prev(R)$ is the SAVE router upstream from R towards S, P_i is the time it takes for router i to process a pushback message, and U_i is the time it takes for router i to propagate an update.

The values of these parameters vary. Assuming values of 20ms for rtt, 100μs for C_i, P_i, U_i, and 10 SAVE hops from S to R, the transient period will be 50ms.

3.7.3. Overhead

Here we discuss SAVE's storage, network, and computational overhead.

3.7.3.1. Methodology

For overhead evaluation we use the J-Sim [67] network simulation framework. (Note the DPF simulator cannot calculate SAVE storage or network overheads.) The J-Sim framework simulates all routers, links, and messages in a network topology in order to conduct detailed overhead evaluation. This, however, limits the size of the topology to generally 5,000 nodes—even with our fairly high-end evaluation environment. (We performed all the evaluations on a computer with 16 GB of RAM, dual 2.6 GHz AMD dual-core Opteron 285 CPUs, running CentOS Linux 4.6.) To solve this problem, we note that SAVE can run at two separate levels: intra-AS level and inter-AS level, and we can evaluate the overhead at these two separate levels. At the intra-AS level, although a small number of ASes may have more than 5,000 routers, most ASes will fall into the range that the J-Sim framework can simulate. At the inter-AS level, all border routers of an AS can act as *one* "virtual router" with the entire AS as its source address space, and therefore SAVE's overhead at the inter-AS level can be analyzed using a topology of all virtual routers—which is equivalent to an Internet AS topology. As the Internet has approximately 26,000 ASes, a detailed J-Sim simulation with up to 5,000 nodes should be close enough for us to understand SAVE's overhead at a large scale.

The overhead analyses at intra-AS level and inter-AS level are similar, except that different topology models probably should be used. In this paper, we focus on the inter-AS level where each node is an AS (or a virtual router), and use network topologies generated by shrinking AS topologies with Orbis [68]—such topologies are smaller than the AS topology of the real Internet but they have similar structure patterns.

Also, we again evaluate multiple placement strategies of SAVE routers. We evaluate vertex cover deployments, deployments at the top 1% of routers by degree, and biased 1% deployments consisting of a random half of the top 2% of routers by degree. They are all effective from our efficacy analysis above. We simulate networks ranging in size from 500 to 5,000 nodes (only up to 3,000 nodes for vertex cover deployments due to computation power limitation).

3.7.3.2. Storage Overhead

In this paper, the blacklist is the only new data structure. We now implement it as a cache of fixed size that runs the Least Recently Used (LRU) algorithm to replace old entries. Further work is needed to evaluate how the size affects the efficacy of the system. We do not worry about losing old blacklist entries; neighboring routers will still filter spoofing traffic, and the on-demand-update mechanism can recreate entries if necessary in any case.

3.7.3.3. Network Traffic Overhead

Figures 3.10 and 3.11 show the per-router traffic during spoofing attacks. As the network size increases, network overhead decreases—because spoofing traffic, and thus control traffic, is more spread out (Figures 3.10a and 3.11a). With a fixed network size (1,000 nodes) and increasing spoofing traffic, the network overhead increases linearly (Figures 3.10b, 3.10c, 3.11b, and 3.11c). This overhead is offset significantly as the SAVE system is also removing spoofing traffic from the network. Instead of the spoofing traffic overloading its target at the edge of the network, routers drop the spoofing traffic and SAVE's traffic overhead is spread out inside the network. Note that due to simulation limitations we cannot simulate a larger number of attackers, but results from Figures 3.10 and 3.11 is still indicative about the network overhead effects from the size of the network, the number of attackers, and the number of spoofing packets.

3.7.3.4. Computational Overhead

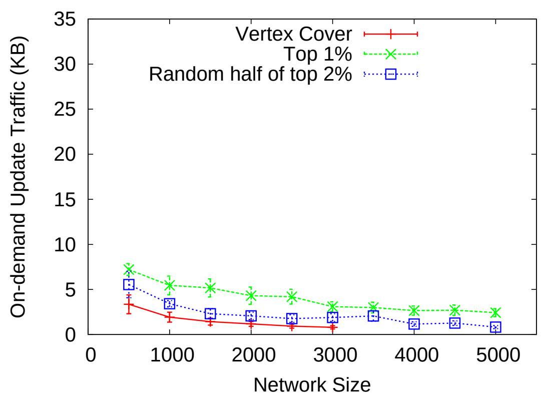
SAVE's most crucial computational overhead is the time taken for a router to classify packets, which mainly consists of table lookup operations (using a router's incoming table and blacklist). We do not have actual measurements for computational overhead since the system is only implemented as a simulation, but we expect SAVE will impose only a minimal computational cost. Today's routers are designed for fast, efficient table lookups (a router's main function is forwarding table lookup).

3.8. Discussion

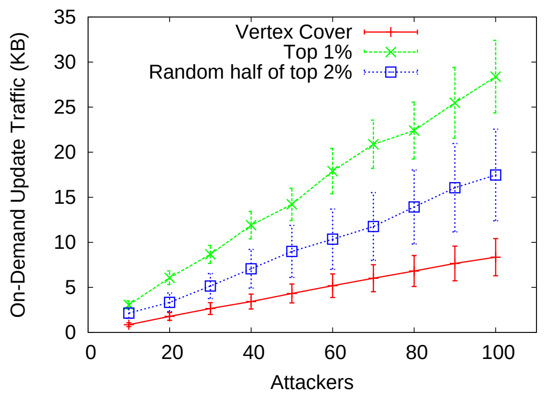
Several issues related to incoming-table-based IP spoofing detection warrant further investigation. These issues include false positives, incentives for deploying SAVE, and spoofing strategies attackers can employ to avoid SAVE.

3.8.1. False Positives

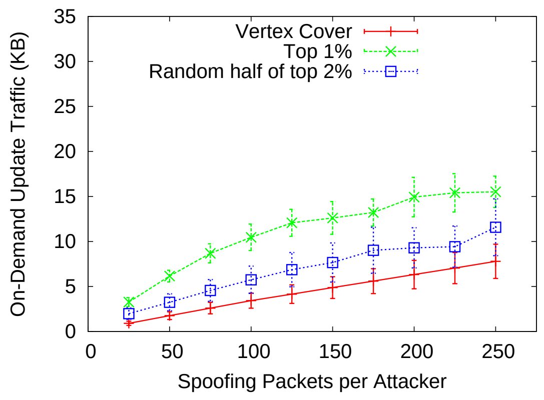
As discussed in Section 3.7.2, our pushback mechanism is subject to false positives when certain conditions are met. It is important to remember that it is only for a transient period which is very short—only after a routing change which makes legitimate traffic match outdated blacklists. Further minimizing the false positives is important and we plan to explore the possibilities in a future work.



(a) On-Demand Update Traffic

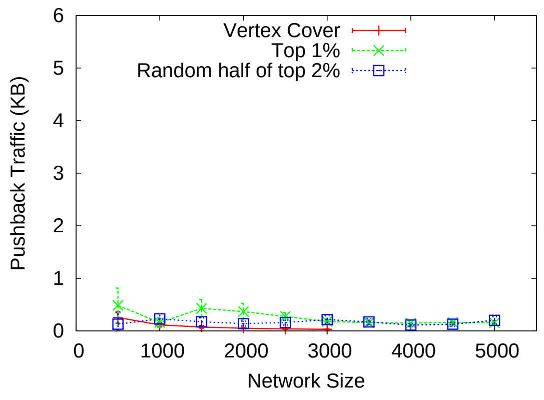


(b) On-Demand Update Traffic (varying attackers)

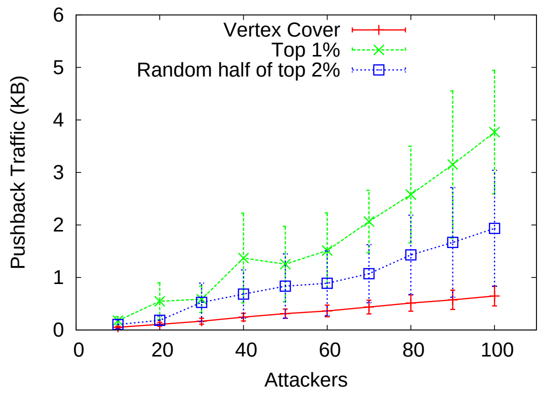


(c) On-Demand Update Traffic (varying packets per attacker)

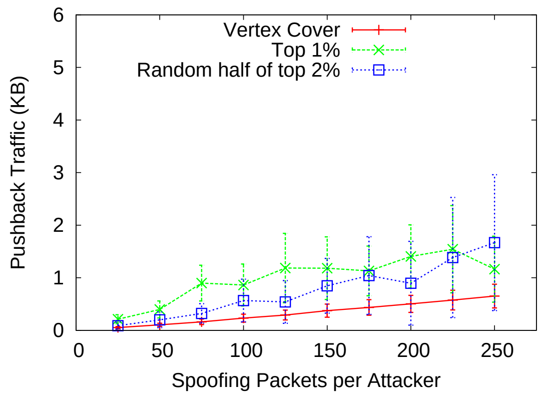
FIGURE 3.10. On-demand update network traffic due to spoofing (with 95% confidence intervals).



(a) Pushback Traffic



(b) Pushback Traffic (varying attackers)



(c) Pushback Traffic (varying packets per attacker)

FIGURE 3.11. Pushback message network traffic due to spoofing (with 95% confidence intervals).

3.8.2. Incentives

As SAVE can only be deployed incrementally, for successful incremental deployment, domains must *want* to deploy SAVE. There must be incentives that include a clear benefit for the deploying domain. “Early adopters” of the protocol should be attracted when the deployment level is still low.

We already know the following incentives for deploying SAVE on a network: Attackers are less likely to successfully spoof source addresses belonging to a SAVE-protected domain, protecting a domain from misplaced blame and reflection style attacks. A protected domain will also allow fewer spoofing packets to enter its network, protecting internal hosts from receiving spoofing packets. Furthermore, SAVE routers can assign higher priorities to packets from SAVE-protected source address spaces, giving clients with protected sources higher quality of service.

What needs to be further studied is how highest-degree ASes can become incentivized to deploy SAVE. From Section 3.7.1 we know that deployments at highest-degree ASes will be mostly effective while random deployments will be least. One driving force here could be that knowing the correct source of traffic may help large ISPs monitor and manage their traffic more reliably according to contractual agreements with their customers.

3.8.3. Spoofing Strategies

Attackers may employ peculiar spoofing strategies to evade SAVE’s filtering. One such spoofing is random source spoofing in which the attacker stamps a random source address on every packet it sends out. When a SAVE router receives any such packet, it will locate the incoming table entry that matches the inscribed source of the packet. Since for each such packet the SAVE router probably has not seen its inscribed source before (i.e., no blacklist entry established), it will treat everyone of them as suspicious, and still forward them. The router will request on-demand updates, but only to confirm that its incoming table is up-to-date.

Fortunately, the damage an attacker could cause with random spoofing is limited. Random spoofing could hide the attacker's true identity, but random spoofing cannot be used in attacks such as DNS amplification [4, 5], DNS cache poisoning [7], in-window TCP resets [6], and spam filter circumvention [8, 9]. Any type of reflection attack cannot succeed, since traffic triggered by the spoofing packets will spread out through the network instead of becoming concentrated in one area. Similarly, the effect on the SAVE infrastructure is manageable since any requests for on-demand updates will also be spread throughout the network.

We are investigating the most cost-effective way of addressing this spoofing strategy. Our concern with random spoofing is the effect it might have on SAVE itself, so our solution focuses on minimizing the overhead it could generate. In our current solution, described only briefly for space considerations, a router does not request an on-demand update for every suspicious packet. Instead, routers use a truncated binary exponential back off strategy. Initially, a router will request an on-demand update after it sees $n = 1$ suspicious packet. If the requested update shows the incoming table was in fact correct, the router will not request another on-demand update until it sees $n = 2$ more suspicious packets. Every time a requested update arrives, if it agrees with the incoming table, the router subsequently waits for $n = 2n$ more suspicious packets before requesting another on-demand update. We do not allow n to increase over 1024. On the other hand, if the requested update shows the incoming table was incorrect, the router decreases the wait to $n = 1$ suspicious packet. In this manner, random spoofing by attackers cannot cause too much network overhead nor fill up a router's blacklist; at the same time, SAVE can continue to quickly update its incoming table.

3.9. Recap

In this chapter we presented a new and enhanced version of SAVE which provides an efficient, effective, and incrementally deployable IP spoofing defense. We identified the incremental deployment problem in the original SAVE system, and in response we created the blacklist, classification, and pushback mechanisms. With the added mechanisms, SAVE has become the only IP spoofing defense to adequately meet all the requirements we outlined in Chapter II. With the IP source address side of our defense taken care of, we will move on to IP prefix hijacking and the IP destination address side in Chapter IV.

CHAPTER IV

LIMITATIONS OF CURRENT PREFIX HIJACKING DEFENSES

Now that we have explored the source address side of identity attacks at the IP layer, it is time to move to the other side and focus on attacks upon destination addresses. First we must review the existing research that has been done in this area.

The security of destination addresses is tightly coupled with routing protocols, since that is what routers use to decide how to forward packets toward a destination address. On the Internet, the de-facto inter-AS routing protocol is border gateway protocol, or BGP. There are a number of BGP anomalies that have been studied, and IP prefix hijacking is just one specific type of anomalous BGP behavior. In order to better understand prefix hijacking, we should also understand other sorts of BGP anomalies and the security research relating to those BGP anomalies.

In this chapter we will first broadly review anomalous BGP behavior, including understanding, detecting, and monitoring anomalous BGP behavior. After we have a general understanding of BGP anomalies, we will then delve deeper to look at IP prefix hijacking. Our main focus will be specifically on hijacking detection, since even detection by itself is not a solved problem and detection must be tackled before we can take on further defenses. Once we are armed with an understanding of the existing research, we contribute a new system for detecting IP prefix hijacking in Chapter V.

4.1. Understanding, Detecting, and Monitoring Anomalous BGP Behavior

Before looking specifically into IP prefix hijacking research, it is beneficial to further understand anomalous BGP behavior in general. Prefix hijacking is one specific type of anomalous BGP behavior, and understanding other areas of BGP anomaly research may be beneficial to understanding IP prefix hijacking.

4.1.1. Understanding BGP Dynamics and Anomalies

A number of works have contributed to a better understanding of BGP routing dynamics, often about anomalous BGP behavior; most of them, however, are not about individual prefixes. The seminal work by Labovitz et al. [69] found many trends in routing dynamics, including some pathological behaviors. Their follow-up work [70] and the work in [71] further investigated the origins of BGP dynamics. Further research [72] has also revisited BGP dynamics, a decade after the original study in [69]. In addition, research in [73] also studied the effects of BGP router misconfiguration [74, 75], and works in [76, 77, 78, 79] investigated the effects of Internet worms, electricity outage, and other significant events on BGP dynamics.

4.1.2. Detecting BGP Anomalies

Other works have tried to detect BGP anomalies. Research in [80] presented a visualization-based approach to detecting BGP anomalies, but requiring significant manual intervention. Zhang et al. proposed two methods for detecting certain anomalous BGP dynamics [81]: a signature-based method that analyzes BGP update bursts and a statistics-based method that attempts to learn “expected” behavior. There are also studies on how to detect and classify the impact on BGP from disruptive events [82, 83, 84], but those works considered events at the global level, not at the prefix level.

4.1.3. Monitoring BGP

There are also BGP monitoring systems which are fairly general. RIPE's Routing Information Service [85] includes various tools that can be used to monitor details of a prefix, Cyclops [86] provides a service allowing network operators to compare their network's observed behavior to its intended behavior, which also includes some hijacking detection capabilities. More recently, BGP-lens [87] uses data mining to automate the analysis of BGP updates for detecting anomalies. A common shortcoming of these existing monitoring systems is that, although they provide some level of automation, they rely heavily on prefix owners to know what paths to their prefix are valid, or more generally, what is normal to their prefix.

4.2. IP Prefix Hijacking

We review existing research on prefix hijacking into understanding, mitigating, and preventing hijacking. We elaborate upon hijacking detection research in Section 4.3.

4.2.1. Understanding Hijacking

To begin with we need to understand what IP prefix hijacking is. Latt et al. [88] provide a good overview of what hijacking is, and how hijacking works. In the simplest terms, IP prefix hijacking is when an attacker is able to capture traffic that was intended for someone else. Attackers can accomplish this by poisoning the routing tables of Internet AS routers, through malicious BGP updates that advertise false routes; they make it appear that they have a good route to a victim prefix, so that AS routers will choose their false path over the legitimate path. When a router chooses the false path instead of the legitimate path, to reach a victim prefix, then that victim prefix is said to be hijacked.

It is also very important to understand how susceptible the Internet is to prefix hijacking. Lad et al. studied the topological connectivity of Internet networks and studied how resilient they are to prefix hijacking attacks [89]. They found that an alarming portion of the Internet is susceptible to IP prefix hijacking. Additionally, there have been numerous real world cases of prefix hijacking [90, 91, 92, 15, 3, 13, 2].

4.2.1.1. IP Prefix Interception

Digging further, Ballani et al. performed the first in-depth study of IP prefix interception, and reported on how successful different ASes could be at performing prefix hijacking [14]. Latt et al. further studied various hijacking and interception methods, and surveyed known hijacking incidents [88]. Both works show the threat of prefix hijacking is serious, and even more serious when interception is considered. IP prefix interception is a special type of hijacking. Interception is when an attacker redirects traffic for a victim IP prefix to themselves, but purposefully does not advertise its malicious BGP updates along all of its connections. The attacker leaves some paths going to the legitimate owner, so that it still has a forwarding path to reach the legitimate owner. Then, when the attacker receives a packet for the hijacked (intercepted) prefix, it will forward the intercepted packets along its path to the legitimate owner. In this way the owner might never even know their traffic was being intercepted by a malicious third party. This sort of hijacking is important to remember when analyzing hijacking detection methods, since detection will fail if it is assumed that the prefix owner will not receive packets destined for their network when it is hijacked. Furthermore, it is not an empty threat: as a proof of concept, researchers [14] were able to intercept their own prefix from a second location, and further analyses showed that Tier 1 and 2 ASes would be able to hijack and intercept any prefix with a greater than 50% probability.

4.2.1.2. Sub Prefix Hijacking

Another important type of hijacking is sub prefix hijacking. It is simply when an attacker advertises a sub prefix of a victim prefix. Due to the way BGP operates, AS routers will generally route towards the most specific prefix advertised, so sub prefix hijacking is generally very successful in hijacking a prefix. The YouTube hijacking event [3] was one such sub prefix hijacking event, where a Pakistani ISP accidentally hijacked YouTube's IP prefix. In this case, the ISP really did mean to hijack the prefix, but they meant to only hijack the prefix for customers inside Pakistan. Instead, they advertised their false routes to the whole world, and all of YouTube's traffic was temporarily redirected to Pakistan.

4.2.2. Mitigating Hijacking

Moving beyond detection, Zhang et al. provide us with the first in-depth look at methods of mitigating hijacking [93]. The goal of the work is to decrease the percentage of ASes that are polluted with bad routes. The system works by having some ASes stop announcing routes identified to be bad, and some ASes promote valid routes by making the routes appear shorter. Qiu et al. provide a piece of the mitigation puzzle with LOCK, which can help pinpoint an attacker's location in the Internet [94]. Learning the location of an attacker helps defenders know the best way to mitigate an attack.

4.2.3. Preventing Hijacking

Since IP prefix hijacking occurs due to falsified BGP updates, hijacking prevention centers around securing BGP. Securing BGP has been studied extensively, with S-BGP [95] being the first complete solution. Unfortunately, the cryptography involved in S-BGP requires too much overhead for routers to handle, mostly because of the nested signatures and extensive use of public key cryptography. More recent works [96, 97, 98, 99, 100, 44, 101, 102] try to improve the overhead through mechanisms such as offloading some functions to server machines, taking advantage of path stability, using lighter cryptography, or providing weaker forms of authentication. It is not clear when any secure BGP implementation will be deployed widely enough to be relied upon.

4.3. Detecting IP Prefix Hijacking

In this section, we discuss the limitations of current prefix hijacking detection systems, which is our primary focus. First we will go over what we consider requirements of a good IP prefix hijacking detection system:

1. Ability to detect hijacking: While this may seem obvious, some methods of hijacking a prefix are more easily detected. A good detection system must be flexible enough to detect more than just certain hijacking cases.

2. Accuracy in detecting hijacking: Not only must a defense system detect hijacks, but it must do so accurately. If no hijacking occurs, there should be minimal amounts of false positives, and minimal amounts of false negatives during times with no hijack.

3. Resiliency against circumvention: Once a detection system is in place, intelligent attackers will attempt to circumvent the detection mechanisms. A good detection system must anticipate such attackers while designing its detection mechanisms.

4. Speed of detection: A prefix hijacking attack must be detected before it does too much damage to the victim network. The faster the better.

5. Deployability: If a detection system requires drastic changes to the routing infrastructure, or requires widespread deployment to be effective, it is not a good design. No matter how well it works, without being easily deployed it simply won't be deployed in the real world at all.

Using the above requirements we will measure existing works against them to see how well they measure up. Note that accuracy and resiliency are tightly related to the ability to detect various forms of hijacking. Not being able to detect certain types of hijacking may lead to false positives, and lead to openings for circumvention. But some systems may be able to detect all types of hijacking in principle, but still be inaccurate in the detection and/or have other possibilities for circumvention.

4.4. Categorizing and Analyzing Hijacking Detection Systems

Hijacking detection systems can roughly be broken off in to three distinct groups. One is control-plane based systems, which attempt to detect anomalous BGP updates. These systems passively collect BGP data and perform some analysis on the data to detect if any of the updates are suspicious. The second group is data-plane based systems. These systems send some probing packets to monitored prefixes, and based on the results from the probe attempt to decide if the prefix is hijacked or not. A third group uses a mixture of control-plane data and data-plane probes to detect hijacking.

We categorize the existing works in Table 4.1.

TABLE 4.1. Hijacking Detection Systems

Control-plane Systems	**Data-plane Systems**	**Combination Systems**
[103, 104, 80, 105, 106, 107, 108]	[109, 110]	[111, 14, 112]

4.4.1. Control-plane Systems

Since IP prefix hijacking is performed by attackers sending malicious updates at the control-plane, it logically makes sense to also perform detection at the control-plane. By being able to see the updates that cause a hijacking, it gives a detection system the fastest notification that something may be suspicious. Thus, most of the detection systems have been proposed at the control-plane. We provide an overview of how the systems measure up against our requirements of a good detection system in Table 4.2.

TABLE 4.2. Analysis of Control-plane Based Systems

	Detection	Accuracy	Resiliency	Speed	Deployability
[104]	⋆	⋆	⋆	⋆⋆⋆	⋆⋆
[80]	⋆	⋆	⋆	⋆	⋆
[105]	⋆	⋆	⋆	⋆⋆⋆	⋆⋆⋆
[103]	⋆	⋆	⋆	⋆⋆⋆	⋆⋆⋆
[106]	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆
[107]	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
[108]	⋆	⋆	⋆	⋆⋆⋆	⋆⋆⋆

One of the earliest detection systems was proposed by Zhao et al. [104]. The research proposed some additions to BGP that would allow a BGP router to detect invalid route announcements from the wrong origin AS. The detection was focused solely on invalid MOAS (Multiple Origin AS) announcements, so it misses more advanced methods of hijacking. As long as an attacker leaves the origin AS the same as the legitimate route, their hijack will go undetected. Due to being at the control-plane, the system is fast. The system works with only a partial deployment, however it requires changes to BGP, and relies somewhat on optional BGP values that routers are not obligated to forward. So even though it can be partially deployed, it faces difficult deployment issues.

Another early control-plane based system was proposed by Teoh et al. [80]. More of a tool to help network operators, the research provides a visualization that an operator can use to visually identify prefix hijacking. It does not provide an automated detection mechanism. It is also focused only on invalid origin AS announcement based hijacking, missing more complicated attacks or attackers attempting circumvention. Due to its reliance on an operator for visual identification, it cannot be considered especially accurate, resilient, fast, or deployable.

Kruegel et al. [105] proposed another control-plane based system focused on invalid origin AS announcements. Their approach detects hijacking by assuming that if two ASes advertise themselves as a prefix's origin, they should also be geographically close based on WHOIS [113, 114, 115] data. This provides fast and automated detection, but as it is still focused on invalid origin AS announcements it can only detect the simplest of hijacks. The assumption of ASes being geographically close is also not always a valid assumption in legitimate cases. Since it simply observes BGP routing advertisements and WHOIS data, the system is fairly easily deployed.

Some later control-plane based systems attempt to improve upon the types of hijacking that they can detect, but they require greater involvement of prefix owners.

RIPE's MyASN [106] requires a prefix owner register their prefix with the system, and then the system records changes to that prefix's path and reports those changes to the prefix owner. This allows the prefix owner to check the changes to ensure they are valid. Some automation is provided, as long as the prefix owner is able to provide the system with some rules that the prefix's AS path should match. This provides greater detection and accuracy capabilities, but relies too heavily on prefix owners to know what is valid and what is not. In many cases a prefix owner cannot know what would be valid beyond perhaps their origin AS.

PHAS [107] also provides control-plane data to the prefix owner, and has a similar set of characteristics as MyASN. PHAS provides a report of any changes to the set of origin ASes or the set of ASes one hop from the origin, and then relies on the prefix owner to decide what is a false alarm and what is not. This will detect hijacks that have an invalid origin AS, and that have an invalid AS one hop from the origin, but does not go any further. Since all changes are reported to the monitored prefix, there is likely to be a large number of false positives that prefix owners must ignore. The speed of the system is somewhat debatable—PHAS itself is very fast in detecting the changes to the set of origin ASes and the set of ASes one hop from the origin, but the prefix owners must further process the reports to decide what is a false alarm or not. The system is easily deployed, since it only requires some monitors to process BGP updates, and some notification system to send reports to prefix owners.

The Internet Alert Registry (IAR) [103] was another system that monitored BGP announcements for invalid origin ASes. It uses the mechanisms from Pretty Good BGP [101, 102] to detect invalid route announcements. It keeps track of route history information to determine who is likely to be an actual prefix owner. When an AS announces a prefix that it is not listed as likely owning, then the announcement is considered suspicious. To avoid false positives it considers a path legitimate if the likely owner of the prefix is listed in the path announcement somewhere else, assuming that if the AS owning the prefix is listed then traffic must be go through that AS before reaching the new origin AS. The system will miss hijacks where the attacker does not list their AS as the origin AS. Legitimate routing changes will also cause false positives. The system is fast at detecting suspicious route announcements. It can be deployed incrementally, and can perform detection simply by using existing BGP data archives and monitors.

The Neighborhood Watch system [108] detects suspicious routing updates by creating filters based on data from the Regional Internet Registries (RIR), which contain information on which IP prefixes are assigned to which AS, and Internet Routing Registries (IRR), which contain information on AS routing policies. Although the RIR and IRR are not always up to date, the system is able to still provide some benefit. Detection still relies on attackers using an invalid origin AS, however, so other hijacking methods and more intelligent attackers will go unnoticed.

4.4.2. Data-plane Systems

Some systems have begun relying solely on data-plane measurements as opposed to control-plane monitoring for IP prefix hijacking detection. The rationale for moving to the data-plane is that although routing announcements are done at the control-plane, using probing messages at the data-plane can provide a more accurate representation of how packets are really being routed. We provide an overview of how these systems measure up in Table 4.3.

iSPY [110] operates in the data-plane, using a reachability detection mechanism. In iSPY, a prefix sends out traceroute messages to hosts in different ASes throughout

TABLE 4.3. Analysis of Data-plane Based Systems

	Detection	Accuracy	Resiliency	Speed	Deployability
[110]	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
[109]	⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆

the Internet, and measures whether or not it receives a response. The intuition is that if a prefix is hijacked, the prefix will not be able to receive all the traceroute replies. Then, depending on where in the network it detects replies becoming lost, and the nature of the cut in the network graph it produces, iSPY decides if the lost replies are due to hijacking or simply network failures. This method is easily deployable by requiring all the monitoring to be done by individual prefix owners. It is able to detect hijacks no matter the mechanism used (origin AS, AS near the origin, or anywhere else along the path), but in the case of interception the system will fail. When a prefix is intercepted, although the attacker receives the victims traffic, the attacker will also forward the traffic to the legitimate owner—which means that no traceroute replies would be lost, and the attack would go undetected. Furthermore, it is also susceptible to sub-prefix hijacking, where a hijacker advertises an invalid path to a subspace of a prefix to hijack only that subspace. Even if routers maintain a correct path to the prefix, due to the preference for more specific routes, they will adopt this invalid path for reaching the subspace. In order for data-plane based systems to detect sub-prefix hijacks they would have to probe based on each individual IP address, as opposed to a single IP address per prefix. For iSPY, the speed of the system depends on how often the prefix sends out probing messages.

Zheng et al. [109] propose a system for detecting prefix hijacking based only on data-plane measurements. The system detects whether the route to a monitored prefix differs too much from the route to a topologically "nearby" reference point. By providing a more generalized definition of what "normal" should be, this method is able to detect many more hijacking cases. It does not matter which portion of the path differs, whether it is the origin AS, one hop away from the AS, or even at any other point in the path. Due to this property, it would also work against prefix interception and not only normal hijacking. This is a huge improvement over previous works. As iSPY, however, it is susceptible to miss sub-prefix hijacks unless probing is done for every IP address in a monitored prefix. Unfortunately, attackers may still be able to circumvent detection, as we will show. The problem is essentially that when an attacker hijacks a given prefix, it is highly likely that the prefix of its nearby reference point is in the same AS as the monitored prefix. Thus, if an attacker is able to hijack the monitored prefix it would also, just as easily, be able to hijack the reference point's prefix. Thus it is not entirely resilient to determined and intelligent hijackers. The speed may also be an issue, since it relies on probing packets at the data-plane. The speed can only be as fast as the time between probing messages. To monitor a huge number of prefixes quickly will require sending many messages all over the Internet, and thus may not scale well.

To be clearer, we will go in to more detail about how relying on the reference point can be circumvented For detection to work, the system uses monitors distributed throughout the Internet to check whether each monitor's route to the prefix deviates significantly from its route to a topologically nearby reference point. However, an attacker can discover which IP prefix(es) likely contain the IP address of the reference point, and hijack these prefixes and the monitored prefix altogether, thus causing the hijack to go undetected. In particular, if the reference point shares the same origin AS as the monitored prefix, hijacking both in one fell swoop is trivial. Indeed, following the reference point selection process in [109], we found it very common for a monitor to use such a reference point. More specifically, using traceroute data from the iPlane project [116] for about 91,019 prefix atoms, where every **atom** is a group of prefixes that are reachable through the same routes from all locations [117], we found 54.6% of atoms share the same origin with their reference point, while 31.8% of atoms do so from *all* monitors and 45.5% of atoms do so from 90% or more of the monitors (Figure 4.1).

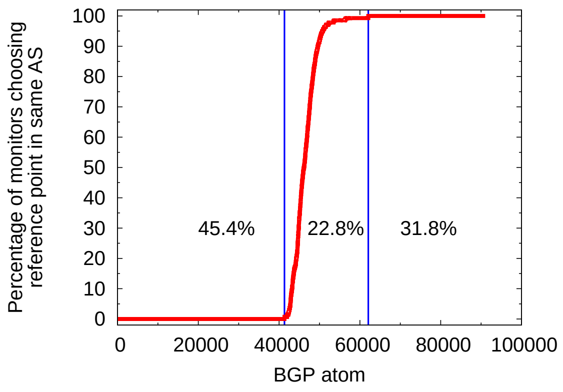


FIGURE 4.1. For each prefix atom the percentage of monitors that use a reference point from the same origin AS as the atom. While 45.4% of atoms do not share the same origin with their reference point from any monitor, 22.8% do so from some but not all monitors, and 31.8% do so from all monitors.

4.4.3. Combination Systems

Another more recent development in detection systems has been mixing data-plane measurements with control-plane information. The hope is that by combining control-plane and data-plane information a more complete picture could be created and provide better detection. See Table 4.4.

TABLE 4.4. Analysis of Combination Systems

	Detection	Accuracy	Resiliency	Speed	Deployability
[111]	⋆	⋆	⋆	⋆⋆⋆	⋆
[14]	⋆	⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆
[112]	⋆⋆	⋆⋆	⋆	⋆⋆	⋆⋆

The Listen and Whisper system [111] combines passively listening at the data-plane with some modifications to BGP at the control-plane. At the data-plane a router would listen to TCP flows to detect unreachable prefixes ("listen"), and the modifications to BGP consist of adding some hash-based signatures to updates in order to help detect invalid path updates ("whisper"). Unfortunately, the listen portion cannot detect hijacks as long as the attacker continues to respond to incoming TCP packets. And even if you assume the whisper portion's changes to BGP could be deployed, it cannot authenticate a path; it can only detect potential inconsistencies. The functionality added by the signatures is too weak to be really useful, and its changes to BGP are great enough to hinder any real world deployment.

Ballani et al. presented a control-plane/data-plane correlation detection mechanism [14] that looks for discrepancies between the topology defined by the control-plane and the topology observed in the data-plane through traceroute probes. Their approach focused on detecting prefix interception, rather than every hijacking case, and is vulnerable to false positives and circumvention. If the control-plane and data-plane match, it does not necessarily mean the prefix was not hijacked. By intercepting the traceroute probes, a hijacker could also send falsified responses to hide some amount of control-plane / data-plane mismatch. The speed improves upon other data-plane based systems since probing is only needed when new routes show up in the control-plane. Deployment can be done incrementally; new monitors will have to be placed throughout the Internet to provide good coverage, and each monitor must be able to observer BGP route announcements and perform probing from the same location.

Hu and Mao [112] detect hijacks by first looking for a variety of suspicious routing updates from the control-plane, and then filtering out events that would be false positives through data-plane fingerprinting and probing. By using the data-plane probes to lower false positives the system can be more aggressive in considering updates to be malicious, and catching more types of hijacking. Unfortunately, attackers could perform similar fingerprinting on the victims and then fool the probes by responding as the victim would respond. Additionally, with an interception attack, the attacker can even forward such probes along to the victim and detection would be impossible.

4.5. Recap

While hijacking defenses have been improving, there are still missing gaps. There is yet to be a hijacking detection method which does well when measured against all of the requirements of a good system. The best systems have done very well in nearly all aspects, but most importantly did not adequately consider the perseverance of attackers and what mechanisms they may use to circumvent detection. A new system must be created that does well at detecting all types of hijacking, accurate in detection, resilient against circumvention, fast in detection, and easily deployed. We present just such a detection system in our next chapter.

CHAPTER V

A BUDDY SYSTEM FOR RELIABLE IP PREFIX HIJACKING DETECTION

Text from this chapter was published as a technical report at the University of Oregon in May, 2012 [20]. Paul Elliott assisted in writing code to implement the skewering algorithm and running evaluations. My professor, Jun Li, and I were the principle investigators for this work.

5.1. Introduction

We present a new approach to IP prefix monitoring. Dubbed *Buddyguard*, it surrounds a prefix with a buddy system composed of buddy prefixes, or *buddies*, and monitors the behavior of the prefix against that of its buddies. Not only does Buddyguard accurately and quickly detect various prefix anomalies including prefix hijacking, but it is also resilient against circumvention by attackers.

We monitor prefix anomalies mainly in the domain of BGP. While we view our research belonging to network monitoring in general, we also treat it as a specific topic in *BGP monitoring*. The main function of every BGP router is to exchange with its peers the route information regarding reaching different IP prefixes, and to determine its own route for reaching them. If this main BGP function goes awry or is attacked, anomalies associated with one or more prefixes then occur.

Key to monitoring an IP prefix is knowing what is normal behavior and what is not, and a buddy system will greatly help that. When inspecting a prefix in isolation, it is difficult to know what behaviors are abnormal. For example, when the path to a prefix from a vantage point suddenly disappears or changes, it can be either a normal routing change, or an abnormal misconfiguration, or that an attacker has just misled routers to adopt a new path under the control of the attacker. In contrast, a buddy system provides a more reliable basis to determine if anything is abnormal with a prefix. By ensuring that (1) under normal conditions a prefix is similar to most of its buddies in terms of the behaviors being monitored, and (2) there are enough buddies for the prefix, we can use these buddies to define what behaviors of a prefix are normal and what are not. Basically, if the behavior of a prefix matches that of most of its buddies, the behavior is normal; otherwise, the behavior is abnormal.

This methodology has the following advantages:

(i) *It is flexible and extensible.* No matter what anomalous behavior of a prefix we want to monitor and detect, we can always first determine the type of behavior and how to measure it, and then select its buddies in terms of that behavior. While we focus on prefix hijacking and misconfiguration in this paper as a means of demonstrating Buddyguard, it would be easy to add additional types of anomalous behavior to monitor as well.

(ii) *It is resilient.* A key feature with Buddyguard is that a prefix is allowed to have *hundreds* or even *thousands* of buddies. These buddies can be from different ASes, and attackers will have extreme difficulty locating and simultaneously attacking enough buddy prefixes to circumvent Buddyguard. Even if the attacker is successful in doing so, an attempt to attack that many prefixes at once would itself appear suspicious.

(iii) *It is easy to deploy.* Buddyguard requires only passive measurement using existing BGP monitoring systems.

The rest of this chapter is organized as follows. In section 5.2 we describe our design of Buddyguard, and in section 5.3 we detail how we evaluate that design through monitoring prefix hijacks and misconfiguration.

5.2. Design of Buddyguard

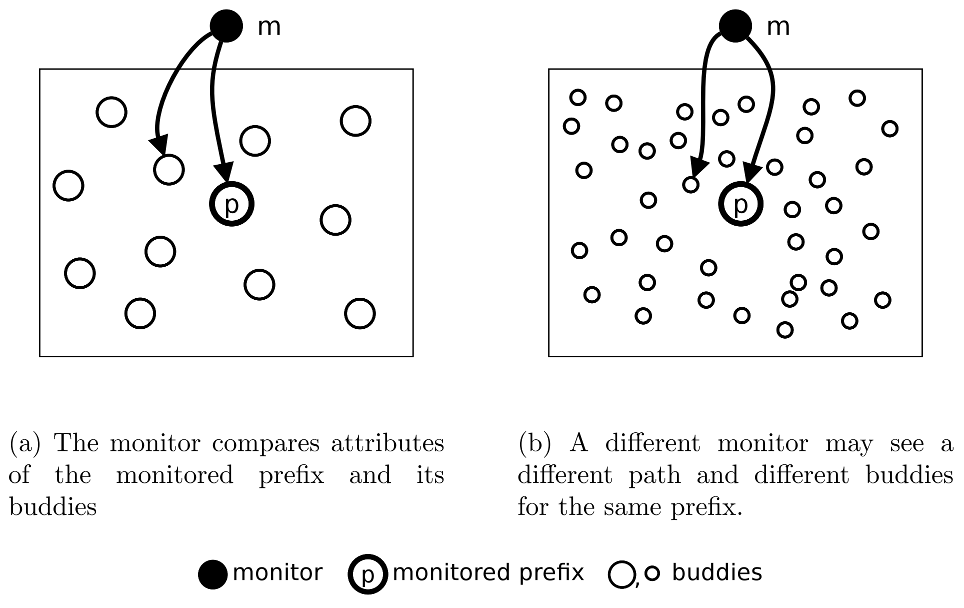
In this section, we present our design of Buddyguard, a control-plane prefix monitoring system that addresses the shortcomings of previous systems described in Chapter IV. We first describe the general architecture, then cover our strategy for discovering and selecting buddies (referred to as **training**), define how Buddyguard performs **monitoring** on selected prefixes using the buddies selected from Training, and finally explain how the system is maintained.

5.2.1. System Overview

At its core, Buddyguard includes the monitored prefix, a set of **monitors**, and the **buddies** of every monitored prefix (Figure 5.1). The monitored prefix is any IP prefix whose owner requests the Buddyguard service for detecting whether or not there is an attempted prefix hijacking attack against that prefix. Buddyguard is able to monitor multiple prefixes in parallel (as we demonstrate in our evaluations, up to hundreds or thousands of prefixes at once). We define the remaining components, monitors and buddies, as follows.

5.2.1.1. Monitors

A monitor is defined as a networked entity that can observe a prefix and its buddies in conjunction. Under normal conditions, the observed behavior of the monitored prefix and its buddies should match. Monitors detect anomalous conditions



(a) The monitor compares attributes of the monitored prefix and its buddies

(b) A different monitor may see a different path and different buddies for the same prefix.

FIGURE 5.1. Core components of Buddygaurd: monitors, a monitored prefix, and buddy prefixes.

when the behavior of a prefix deviates significantly from that of its buddies. We leave the details of the training and monitoring algorithm for a later section.

What behaviors should monitors observe and compare? In trying to detect prefix hijacking and route leaks, the relevant behavior is captured in the BGP updates containing an announcement of a new path to that prefix. The main property we inspect is the difference between this new path and the paths to the prefix's buddies.

Since monitors must be able to measure these BGP operations, monitor placement is critical. Ideally, monitors must be able to hear conversations between BGP routers as close to real-time as possible. One solution involves peering monitors with existing BGP data collection systems such as RouteViews and RIPE [85] collectors or BGPMon [118], which collect real-time BGP updates from routers around the globe. This deployment scheme has the advantage of costing low overhead, as it does not require continuous data-plane queries like other solutions. We will return to the efficacy of this monitor placement strategy in a later section.

5.2.1.2. Buddies

A buddy can be defined as IP prefix that behaves similarly to the monitored prefix under normal conditions, and diverges when anomalous conditions occur. Recall that for prefix hijacking, the behaviors we are concerned with are path updates associated with the prefix and its buddies. To detect whether a prefix is hijacked, we compare paths to b and p such that:

(i) Under normal circumstances, the path from a monitor to b is similar to the path from that monitor to p;

(ii) If a legitimate routing change occurs so that the monitor has a new path to p, the monitor will also have a similar new path to b; and

(iii) If p is hijacked, the monitor will switch to a "bad" new path to p, but will still use the old path to b, causing the two paths to be dissimilar.

Clearly, if b perfectly meets these standards then p will only need that single buddy. However, in many situations buddies can only partially meet the above conditions. Sometimes a buddy may experience the same anomaly as the monitored prefix; for example, a hijacker could co-hijack a prefix and its buddy, leaving the anomaly undetected. Therefore, having one buddy for a prefix is typically not sufficient.

To solve this, we must obtain many buddies for a monitored prefix, where enough buddies are similar to the prefix when it is behaving normally, and at most a small number of buddies may experience the same anomaly together with the monitored prefix. Therefore, if a prefix deviates from enough of its buddies, we can determine that it is behaving abnormally. When detecting if a prefix is hijacked, for example, we modify the conditions (*i*)–(*iii*) above to:

(i) Each monitor m must have a set $B_m = \{b\}$ of buddies such that monitor m will have similar paths to all of them, including prefix p;

(ii) If a legitimate routing change occurs so that monitor m has a new path to prefix p, *enough* of its buddies will also switch to a similar new path from monitor m; and

(iii) If prefix p is hijacked, m will switch to a "bad" new path to prefix p, but *enough* of p's buddies will not switch.

We leave the definition of *enough* for a later section.

Due to the decentralized nature of BGP, it is likely (but not necessary) that each monitor will have a distinct set of buddies for the monitored prefix. The specific location of a monitor will determine which BGP updates it is able to hear; indeed, certain updates may never reach a given monitor at all. The advantage of a per-monitor buddy system over a common buddy system (where a prefix has the same buddies for all monitors) is that each monitor need only be concerned with the BGP updates to which it is privy.

On the other hand, although each monitor may have different buddies for a given prefix p, a monitor may in fact be able to share buddies across different monitored prefixes p_i and p_j. This is will be very useful for maintaining a lower overhead when scaling up to monitor a large percentage of the prefixes on the Internet. Buddyguard must be able to monitor multiple prefixes simultaneously, and if we monitor two prefixes for the same type of anomaly, and their buddies overlap, we do not want a monitor to measure the shared buddies between the two more than once (Figure 5.2). The cost of Buddyguard per monitor is therefore the cost of monitoring every buddy multiplied by the total number of buddies, plus the cost of monitoring every target prefix.

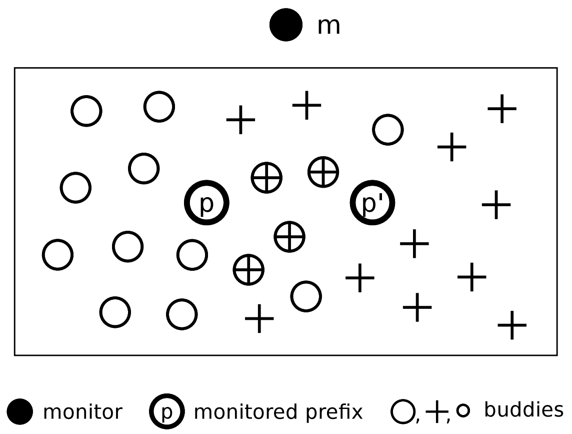


FIGURE 5.2. Sharing buddies.

To build this architecture, we must employ three major classes of algorithms: training algorithms for discovering and selecting buddies for a monitored prefix, a buddy-based monitoring algorithm for detecting prefix hijacking, and a maintenance algorithm to ensure that a prefix always has good buddies.

5.2.2. Phases Of A Monitor

For a given monitored prefix, a monitor has two phases of operation: a training phase and a monitoring phase. The monitor must first complete the training phase before entering the monitoring phase. An overview of these phases is given in Figure 5.3 and below. Details of the training phase and monitoring phase are in Sections 5.2.3 and 5.2.6 respectively.

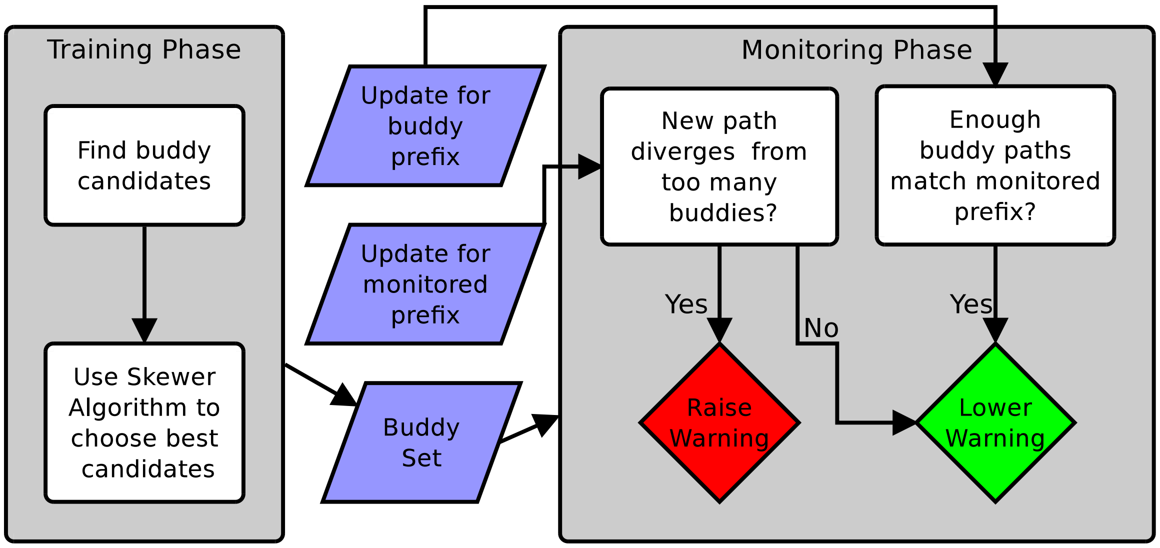

Training Phase
Find buddy candidates
Use Skewer Algorithm to choose best candidates
Update for buddy prefix
Update for monitored prefix
Buddy Set
Monitoring Phase
New path diverges from too many buddies?
Enough buddy paths match monitored prefix?
Yes
No
Yes
Raise Warning
Lower Warning

FIGURE 5.3. The architecture of a Buddyguard Monitor.

During the training phase the monitor first finds buddy candidates. These buddy candidates must match the behavior of the monitored prefix at some point during the training phase, but not all buddy candidates match as well as the others. The monitor narrows down its selection of buddies using the skewering algorithm, and saves its final choices as the buddy set for the monitored prefix. The buddy set is then passed on to the monitoring phase.

In the monitoring phase, a monitor uses the buddy set and observes relevant routing updates. When a routing path update for the monitored prefix is seen, the monitor will check the new path to the prefix against the paths of its buddies. If it diverges from too many of its buddies the monitor will raise a warning. If it matches enough of its buddies the monitor will lower an existing warning, or simply do nothing if no warning was raised. When a routing path update for a buddy prefix is seen, and a warning is already raised, the monitor will see if the buddy's new path would affect its decision regarding the warning. The monitor rechecks the monitored prefix's path against the paths of its buddies, and if it now matches enough of the buddy paths the warning is lowered.

5.2.3. Training: Finding and Selecting Good Buddies

The success of Buddyguard lies in having the best possible set of buddies for a given monitored prefix. To meet this objective, we define algorithms for finding buddy candidates that match the behavior of the monitored prefix and selecting the best matching candidates to be actual buddies. We call this bootstrapping phase: **training**.

5.2.3.1. Finding Good Buddy Candidates

Where should Buddyguard look for buddy candidates for a given monitored prefix? Ideally, we want buddies to be distributed across multiple ASes, making it difficult for a hijacker to co-hijack enough buddies to evade detection. Yet buddies must be well-matched in order for subsequent monitoring to be accurate. Therefore, our task becomes finding well-matching buddy candidates from a diverse set of ASes.

We can achieve this by using a simple path similarity principle. Consider the AS path update $u_p(t_i)$ from a monitor to a prefix p, which consists of an ordered list of autonomous systems (ASes) from the monitor to p. We can define an update u_i as similar to u_p ($u_i \sim u_p$) if both share the first $|u_p| - n$ AS hop. For definition purposes, we use $|x|$ to designate the length or size of x both here and in the remainder of this work. Figure 5.6 shows how $0 \leq n \leq 2$ may affect the distribution of buddy candidates, or which ASes may be eligible to offer buddies. Clearly, when n is 0, buddies can only be from the same AS (the origin AS) as the monitored prefix

(Figure 5.6a). But when n is 1, buddies can be also from the so-called parent and sibling ASes (Figure 5.6b), and so on.

Using this notion of path similarity, we can find well-matching buddy candidates for a given prefix p through observation. Buddyguard observes p over a training period, during which each monitor listens for AS path updates regarding p. For a given monitor m and an AS path update $u_p(t_i)$ to p witnessed at time t_i, m checks for similar path updates $u_c(t_i \pm \Delta)$ to any candidate c that occur at roughly t_i. We specify $t_i \pm \Delta$ to allow time for BGP convergence [119], and use $\Delta = 3\ \mathit{minutes}$ as a conservative measure for this work. After this period, each monitor m will have set $C_m = \{c\}$ of buddy candidates that matched paths from m to p.

5.2.3.2. Buddy Selection

The most frequently matching candidates found during this training period are clearly the best-matching, but how can Buddyguard select buddies such that (1) enough buddies always match the monitored prefix, and (2) the buddies are distributed across multiple ASes, and therefore resilient to co-hijacking? For this task, we employ a **skewering mechanism**. When monitor m hears a path update $u_p(t_i)$, Buddyguard creates a **skewer** data structure for time t_i. By the end of training, we have a set of skewers $S_m = \{s_{t_i}\}$ for every $u_p(t_i)$ that m witnessed (Figure 5.5). We then "skewer" candidates by sorting them by frequency of matching (best to worst), and place the best-matching candidate c on each skewer s_{t_i} where $u_p(t_i) \sim u_c(t_i \pm \Delta)$.

The skewering mechanism enables Buddyguard to select buddies according to the above criteria. We continue to skewer candidates until all of the skewers are full, or more formally:

$$\forall s_{t_i} \in S_m,\ |s_{t_i}| \geq \omega$$

Algorithm 5.2.1: TrainingAlgorithm(p, m)

```
local $C_m$ candidates, $B_m$ buddies, $S_m$ skewers
for each $u_p(t_i)$ seen by $m$
  do {
    find candidates $c$ where
      $u_c(t_i \pm \Delta) \sim u_p(t_i)$
    append $c$ to $C_m$
    create skewer $s_{t_i}$
    append $s_{t_i}$ to $S_m$
  }
sort $C_m$ by frequency of matching
while $|s_{t_i}| < \omega \ \forall s_{t_i} \in S_m$
 and $B_m$ is not diverse
  do {
    place top matching $c$ on every $s_{t_i} \in S_m$
      where $u_c(t_i \pm \Delta) \sim u_p(t_i)$
    append $c$ to $B_m$
  }
return $(B_m)$
```

FIGURE 5.4. Training algorithm. p is the monitored prefix and m is a monitor.

for some lower bound capacity ω (we will wait to define ω until section 5.3.2). This ensures that buddies can account for the full range of normal behavior for the monitored prefix. We can also ensure that buddies are widely distributed by skewering candidates until they cover multiple ASes. Once these conditions are met, Buddyguard selects all skewered candidates as buddies. The training methodology described thus far can be summed up in Figure 5.4.

5.2.4. Using Good Buddies

Key to the success of Buddyguard is that a monitored prefix must have a good set of buddy prefixes. The primary questions that must be addressed in order to meet this objective are (i) what is the definition of *good*; and (ii) how do we obtain such buddies? To address these, we implement a training algorithm in two phases:

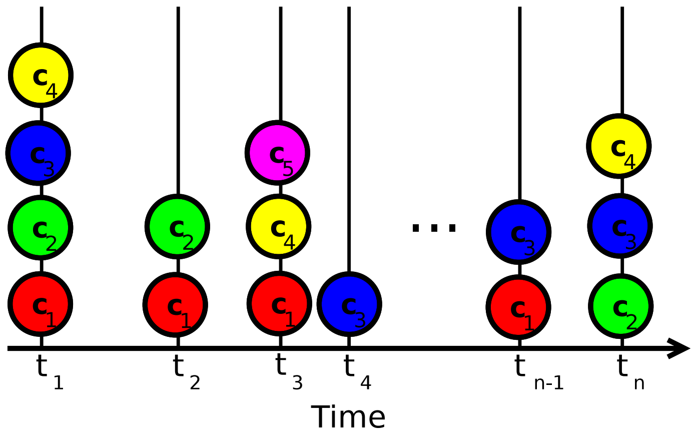


FIGURE 5.5. Buddy selection through the skewer mechanism. A skewer represents a path change for the monitored prefix at time t_i. Each circle represents a buddy candidate c_j.

finding buddy candidates, and selecting the best candidates to be actual buddies for the monitored prefix. Once we have selected a good set of buddies for a prefix, we must also ensure that those buddies stay good over time.

5.2.4.1. What are Good Buddies and Where Are They?

A buddy prefix b is a perfect buddy to the monitored prefix p if the behaviors of b and p are similar under normal circumstances, but differ when an anomaly occurs. Recall that for prefix hijacking, the behaviors we are concerned with are path updates associated with the prefix and its buddies. To detect whether a prefix is hijacked, we compare paths to b and p such that:

1. Under normal circumstances, the path from a monitor to prefix b is similar to the path from that monitor to prefix p;

2. If a legitimate routing change occurs so that the monitor has a new path to prefix p, the monitor will also have a similar new path to prefix b; and

3. If prefix p is hijacked, the monitor will switch to a “bad” new path to prefix p, but will still use the old path to prefix b, causing the two paths to be dissimilar.

In other words, under normal circumstances the perfect buddy b should be **“fate-sharing”**–that is:
for every legitimate path update u_p advertising a path from the monitor to p, there exists a path update u_b advertising a path from the monitor to b such that $u_b \sim u_p$. Clearly, if b is a perfect buddy then p will only need that single buddy.

However, in many situations buddy candidates can only partially meet the above conditions. Sometimes a buddy may experience the same anomaly as the monitored prefix; for example, a hijacker could co-hijack a prefix and its buddy, leaving the anomaly undetected. Or, if p and b are not similar, is it because p is hijacked or because b is hijacked? With only one buddy it would be difficult to say which is right and which is wrong. Therefore, having one buddy for a prefix is not sufficient.

To solve this, we must obtain many buddies for a monitored prefix, where most buddies are similar to the prefix when the prefix is normal, and at most a small number of buddies may experience the same anomaly together with the monitored prefix. Therefore, if a prefix deviates from the majority of its buddies, we can determine that it is behaving abnormally. When detecting if a prefix is hijacked, for example, we modify the conditions (1)–(3) above to:

1. The prefix p will have a set $B = \{b\}$ of buddies distributed throughout multiple ASes, which makes them resilient to attacker countermeasures such as co-hijacking, and the monitor will have similar paths to all of them, including p;

2. If a legitimate routing change occurs so that the monitor has a new path to p, more than $x\%$ (define x as some threshold value) of its buddies will also switch to a similar new path from the monitor; and

3. If the p is hijacked, the monitor will switch to a "bad" new path to the p, but no more than $x\%$ buddies will.

This threshold value x now becomes the deciding factor in determining prefix anomalies. The precision of this threshold is critical—too high a value will result in false negatives (F-), where Buddyguard fails to detect an anomaly; and too low a value will result in false positives (F+), or false alarms. We discuss the fine-tuning of this value in a later section.

Path Similarity and Origin, Parent, Sibling Buddies The modified criteria raise a new question: how do we find buddy prefixes from multiple ASes (thereby making our system resilient) that are still fate-sharing, or as close to fate-sharing as possible? Logically, buddy prefixes that are topologically closer to the monitored prefix will be more likely to share similar behavior with that prefix. Still, we must relax the strict definition of fate-sharing behavior. In the case of prefix hijacking, we use the notion of path similarity as opposed to path sameness.

Consider the AS path from a monitor to a prefix, which consists of an ordered list of autonomous systems (ASes) from the monitor to the prefix. We can define two paths being similar if they differ by at most n AS hops, where n is 0 or 1. Figure 5.6 shows how n may affect the distribution of buddy candidates, or which ASes may be eligible in offering buddies. Clearly, when n is 0, buddies can only be from the same AS (the origin AS) as the monitored prefix (Figure 5.6a). But when n is 1, buddies can be also from the so called parent and sibling ASes (Figure 5.6b). As a monitor

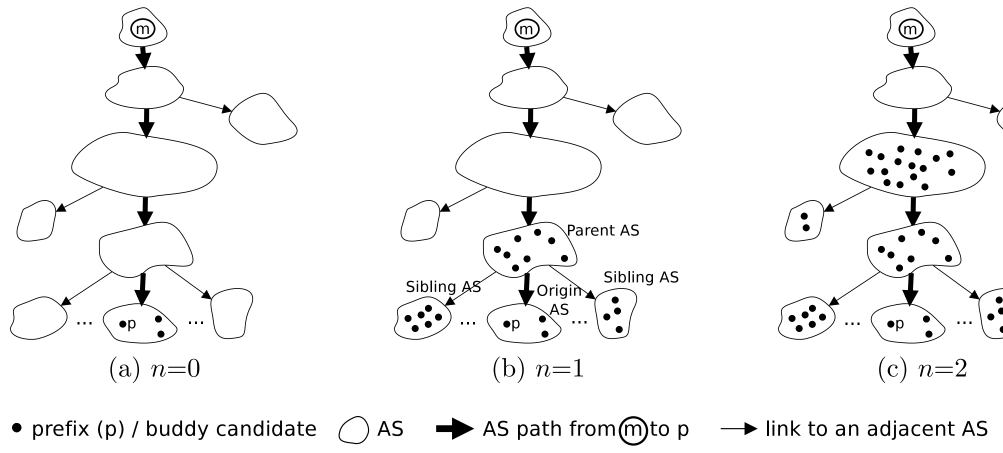


FIGURE 5.6. Buddy distribution with different path similarity definition. (n is the maximal number of different AS hops between two AS paths.)

may have different AS paths to a prefix over time, there can be even more eligible ASes corresponding to every AS path.

5.2.5. Buddy Maintenance

After a monitor selects a set of buddies for a prefix, it is not guaranteed every buddy will always stay a good buddy. How can we maintain a good buddy system after the initial selection? Yet again, the information that Buddyguard collects during training is invaluable for this task. Consider a given monitor m with buddies $b \in B_m$ who matched S_m skewers for the monitored prefix p during training. We can define the minimum quality $min(B_m)$ to be the minimum number of p's path updates matched during training by any buddy $b \in B_m$, and $|S_m|$ to be the number of skewers. During monitoring, we can re-evaluate B_m after $|S_m|$ path updates, and if any $b \in B_m$ matched less paths than $min(B_m)$, we drop b. Similarly, at any point during monitoring we can re-train p, replacing poorer quality buddies with better buddies found during re-training. In this way, we can maintain and improve the quality of B_m to ensure that Buddyguard can effectively monitor p.

Another concern that one might raise is the occurrence of policy changes that cause the monitored prefix to legitimately deviate from its buddies. For example, a prefix may become multihomed or enter a new peering agreement with another AS, while its buddies do not. However, even though these changes may occur at arbitrary times, such changes do not pose a serious problem for Buddyguard. Given that our system would be run as a service, we can simply require prefix owners to report such events when they occur. At that time, we can re-train the prefix according to its newly defined behavior, and continue monitoring with a new set of buddies.

5.2.6. Monitoring: Detecting Prefix Anomalies

Buddyguard provides accurate and fast detection of prefix anomalies using the buddies selected from training. The detection procedure is straightforward—again, using prefix hijacking as an example. If a monitor hears a new path to a prefix p, it will check whether this is one of the "good" paths that is has already seen. If not, the monitor will wait for a short period, and then check if more than $\alpha/|B_m|$ (for some value of α) of p's buddies also switch to a similar new path. If so, the monitor can record that path as a good path for future reference. If not, the monitor raises a warning flag, indicating that the prefix may be hijacked. In this way, buddies help the system determine both normal behavior and anomalous events. Warning flags can be lowered in two ways: (1) more than $\alpha/|B_m|$ of the buddies eventually switch over to the new path, or (2) p later switches to a good path.

Defining this per-monitor warning threshold α would seem to be a difficult task. How can Buddyguard know how many buddies will typically match the monitored prefix? The answer lies in using data from training, and here again the skewering mechanism becomes exceedingly useful. Consider a set of skewers S_m from training, which correspond to paths from monitor m to p. If buddies are normally distributed across these skewers, then we can define $\alpha = \mu - 3\sigma$ where μ and σ are the mean and standard deviation of buddies per skewer, respectively. In other words, based on p's behavior witnessed by m, the probability of having $\alpha/|B_m|$ or less buddies match p is roughly 0.1%. More plainly, the set of p's buddies for a given monitor should match well enough for the majority of p's normal behavior. Of course, it may be the case that buddies are not normally distributed across S_m. For these instances, we define a conservative base case threshold: at least a third of the buddies must match p or a warning is raised.

However, it is not enough to say that the prefix was hijacked if only one monitor raises a warning. The entire Buddyguard system must correlate warning flags from all monitors to decide whether the prefix was hijacked. If more than $X\%$ of the monitors have a warning flag raised at any given time, the system issues a hijacking alert. The alert state can only be dropped if the percent of monitors with warning flags raised drops below threshold X. Given a set of well-matched and widely distributed buddies, this system-wide alert threshold now becomes the deciding factor in determining prefix anomalies. The precision of this threshold is critical—too high values will result in false negatives, where Buddyguard fails to detect an anomaly; and too low values will result in false positives, or false alarms. We discuss the proper calibration of this threshold in our evaluation section.

Finally, the accuracy of a buddy-based monitoring scheme is dependent upon the ability to acquire the right set of buddies during our training period. A careful reader might ask, what if an anomaly occurs during this training period? Clearly this will reduce the well-matching quality of the selected buddies. We must therefore ensure that our training period is “clean”—no anomalies can occur for the monitored prefix during this time window. While this seems like a difficult task, we can exploit a simple principle to make it manageable. If Buddyguard finds buddies for the monitored prefix on the basis of anomalous behavior during training, then subsequent monitoring with those buddies will trigger false alarms for clearly valid path updates. When this occurs, we simply re-train the prefix until we obtain a clean training period. In this manner, we can ensure that our system uses buddies that best match the *normal* behavior of a monitored prefix.

5.2.7. Resiliency of Monitoring

As we alluded to previously, one the primary advantages of the buddy system is that it allows Buddyguard withstand intelligent attacks. With respect to prefix hijacking, there are several measures that a hijacker could employ to avoid detection by monitoring schemes. We now discuss some countermeasures that remain unaddressed in current systems, and explain how our system handles them.

5.2.7.1. Prefix Interception

When a prefix is intercepted, the hijacked traffic is forwarded on to the victim prefix. Since traffic is still routed to the victim prefix, such an attack can easily go undetected. However, from the perspective of a Buddyguard monitor, the path to the victim prefix still changes, and the path comparison with the victim’s buddies will reveal the hijack, enabling Buddyguard to detect the interception.

5.2.7.2. Sub-prefix Hijack

Buddyguard is particularly good at handling sub-prefix hijacking, a case that thwarts most existing solutions. If a monitor has never heard a sub-prefix of a monitored prefix, this monitor will use exactly the same path to reach the sub-prefix and the prefix all the time. It therefore can view the buddies of the monitored prefix as the buddies of the sub-prefix, and use these buddies and the same detection procedure to determine whether the sub-prefix is hijacked.

5.2.7.3. Targeted attacks

What would happen if an attacker was aware of Buddyguard, and specifically attempted to thwart our monitoring scheme? For example, an attacker could try to co-hijack all or most of a prefix's buddies to eliminate the effectiveness of path comparison. However, each prefix we monitor has numerous buddies distributed across multiple ASes, and discovering enough buddies to co-hijack would be an enormous undertaking. An attacker might try to guess enough surrounding ASes and hijack all prefixes within those ASes, but a hijack on that scale would itself be blatantly suspicious. Furthermore, if Buddyguard is monitoring numerous prefixes each with their own buddies, then it is possible that some buddies of a target prefix are also being monitored. An attacker would have to hijack that prefix's buddies plus their own buddies, and possibly their buddies' buddies, resulting in a tedious recursive process (Figure 5.7). In effect, our buddy-based monitoring scheme is resilient against attackers who know how Buddyguard works.

An attacker could also try to exploit BGP routing policies to limit the visibility of the hijack. For example, if a hijacker knew which ASes contained Buddyguard monitors, it might try to make their updates avoid those ASes during route

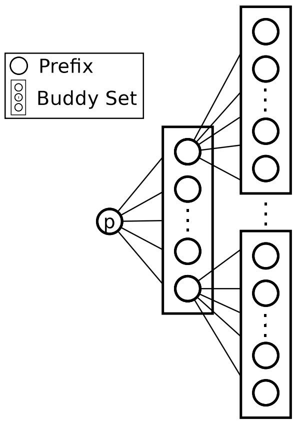


FIGURE 5.7. A prefix has its buddies, which in turn have their own buddies, etc.

propagation. One way it could do this is to insert those ASes into illegitimate path announcements. Since BGP routers discard path updates that contain their AS to avoid routing cycles, these attacks would not be noticed by our monitors. But to do so, a hijacker would have to insert the ASes of many or all of our monitors into an illegitimate path, which would greatly increase its length. Since BGP routers also prefer shorter routes, it is statistically improbable that this path would ever be adopted. Furthermore, even assuming attackers could successfully limit which routers see their malicious updates in some other way, it would limit the effective range of the hijack too severely to be of much use to the hijacker.

5.3. Evaluation

We evaluate Buddyguard in terms of the effectiveness of training (how well it can find and select good buddies), and the accuracy and performance of monitoring. For the latter we look specifically for cases of false negatives (where the system fails to detect an event) and false positives (false alarms). Our results demonstrate that using a system-wide alert threshold $X = 20\%$—recall that this means if 20% or more monitors have a warning raised, issue an alert—gives a reasonable balance for minimizing both false positives and false negatives. Again, without losing generality, we use prefix hijacking and route leaks as a means of demonstrating our system.

5.3.1. Setup

To test Buddyguard, we developed an initial test-bed of monitors, prefixes, and all routing path updates concerning those prefixes during various time periods of interest. We built this test-bed by processing real world BGP updates and RIB table dumps from the RouteViews collection [66], which allowed us to generate a set of over 600,000 prefixes and their associated BGP routing paths. For monitors, we used BGP speakers peered with RouteViews collectors, as these entities would have the same view of BGP routing data as seen in our test data (the coverage of this monitor placement strategy is discussed in Chapter VI). This generated set of prefixes, monitors, and BGP updates enables us to evaluate Buddyguard on today's Internet topology and routing infrastructure.

5.3.2. Defining Thresholds

Our next step was to define threshold values for our training algorithms. To recap, these key thresholds are:

(i) *Path similarity n.* The number of AS hops to compare $(|u_p(t_i)| - n)$ when checking if $u_p(t_i) \sim u_c(t_i \pm \Delta)$.

(ii) *Skewer lower bound capacity ω.* The number of buddy candidates required to fill a skewer.

(iii) *Resiliency factors.* The number of buddies needed for a monitored prefix and the minimum number of ASes covered by these buddies.

To demonstrate the efficacy of our system, we take a conservative approach toward defining the training thresholds. For path similarity, we use $n = 1$ such that Buddyguard may only find buddy candidates from the so-called origin, parent, and sibling ASes. We use the terms **origin, parent, and sibling buddies** to reference buddies from these respective ASes. Restricting our search algorithm in this way forces Buddyguard to rely on a minimal search space when looking for well-matched candidates ($n = 0$ excludes all but the origin AS, and therefore is too strict for our purposes). For the remaining two thresholds, we set $\omega = 30$ and require at least 90 buddies for a given monitor. By intuition, this stipulates that at least a third of the buddies should match the behavior of the monitored prefix during training, and that each monitor has a reasonably large set of buddies. Lastly, we require at least 30 of the buddies for a given monitor to be from sibling ASes, which ensures that the buddies are widely distributed—again, a key contribution to the resiliency of our system. We evaluate these conservative benchmarks in the next section, and discuss in section 5.4 how they might be better calibrated.

Given these values, we proceeded with our tests accordingly. First we selected prefixes randomly from diverse locations of the network topology, including tier 1, tier 2, and tier 3+ ASes. For testing a specific event, we used the prefix(es) involved in that event. Each prefix we monitored was trained over a period of one week, during which we tried to find origin, parent, and sibling buddy candidates for each legitimate path update to the prefix. We define a legitimate update as an announced path that is not part of a routing path oscillation (for an explanation of this phenomenon, see [72]). We then used the skewering mechanism to select the best buddies from these candidates, and after verifying that the training period was clean, we monitored the prefix with the selected buddies. In order to prove the efficacy of using buddies from outside of the prefix's origin ASes, we show the results of monitoring a prefix with origin, parent, and sibling buddies separately. The following sections summarize our test results.

5.3.3. Evaluating Training

An important criteria for the effectiveness of training is whether Buddyguard can find numerous well-matched buddies across multiple ASes. We now show our training results across all the prefixes that we tested.

With respect to buddies selected per monitor for a given prefix, we see that in most cases Buddyguard is able to find hundreds or thousands of eligible buddies (Figure 5.8). However, we note that a very small portion of our samples (about 5%) have insufficient buddies. Fortunately, these edge cases can be attributed to our conservative definition of $n = 1$ for path similarity. Most of them come from very small ASes (tier 3+), where there is a deficiency of potential origin, parent, and sibling candidates. Increasing n adaptively would enable Buddyguard to search for candidates beyond those small ASes, allowing our system to find sufficient buddies for all cases. We discuss in section 5.4 why we did not use this strategy, and for now are satisfied that $n = 1$ gives us enough buddies in the majority of cases.

Our results are very similar when we look at the mean number of buddy or candidate ASes per monitor, as shown in Figure 5.9. Though we restrict candidate searching to origin, parent, and sibling ASes, this is still a large number of eligible ASes. In many cases Buddyguard had hundreds or thousands of ASes to choose from, and again, we can fix the few exceptions by relaxing n for path similarity. Therefore, our initial evaluations show that even under conservative restrictions, Buddyguard is able to find numerous and widely distributed buddies for monitoring prefixes.

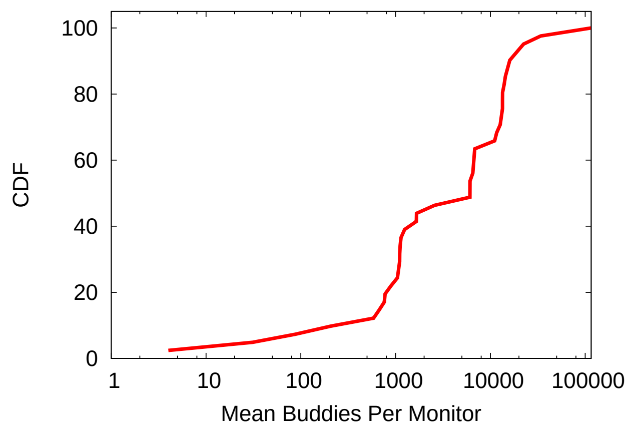


FIGURE 5.8. CDF for mean value of buddies selected per monitor (X axis shown in log scale).

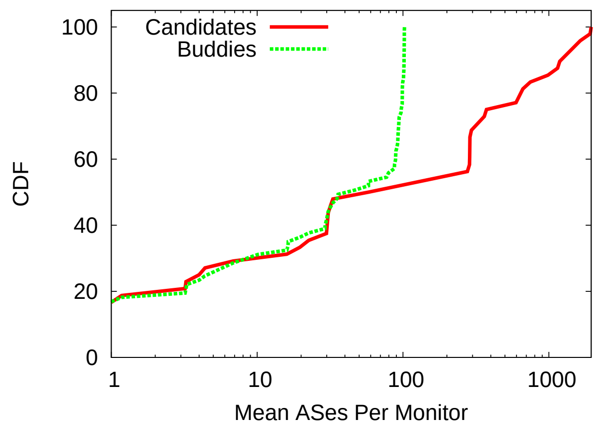


FIGURE 5.9. CDF for mean value of buddy and candidate ASes during training (X axis shown in log scale).

It is also worth noting the general quality of the buddies selected. Figure 5.10 shows the CDF of buddy quality with respect to the percentage of prefix paths matched during training. The results are not exceptional–less than half of the samples match their respective prefix with any degree of regularity. While this may seem problematic, we demonstrate in the following sections that Buddyguard is able to accurately detect anomalies even with mediocre buddies.

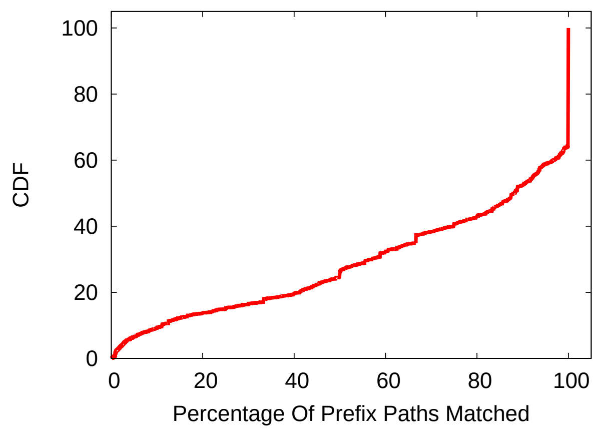


FIGURE 5.10. CDF for buddy quality in terms of percentage of monitored prefix paths matched during training.

5.3.4. Accuracy

5.3.4.1. Detecting Prefix Hijacking

We evaluated Buddyguard's ability to detect prefix hijacking by testing our prototype on a wide manner of known hijacks. For brevity, we only show the results for three well-known hijacking events:

– Cogent's hijack of one of Google's prefixes [2].

– Con Edison's hijack of 30+ prefixes, including some belonging to their customers [13].

– Pakistan Telecom's hijack of a sub-prefix of YouTube's prefix [3].

Our results from these three hijacking events show that Buddyguard is well-suited to detecting prefix hijacks. For most tested events, a system-wide alert threshold $X = 20\%$ monitors with warnings raised (represented in our figures by a horizontal line at 20%) suffices for detecting these hijacks. While in many cases X could be a smaller percentage, we point out that this conservative measure works for most tested events. We will also show that this observed threshold is effective for route leaks *and* maintains low false positives when monitoring normal prefix behavior.

When we inspect results from individual events, the efficacy of our monitoring system is clearly demonstrated. Looking at the Cogent/Google event (Figure 5.11), we see that the percentage of monitors with raised warnings dramatically increases for the duration of the event, and each monitor detects the hijack within 5 seconds of the event. Results are similar across origin, parent, and sibling buddies.

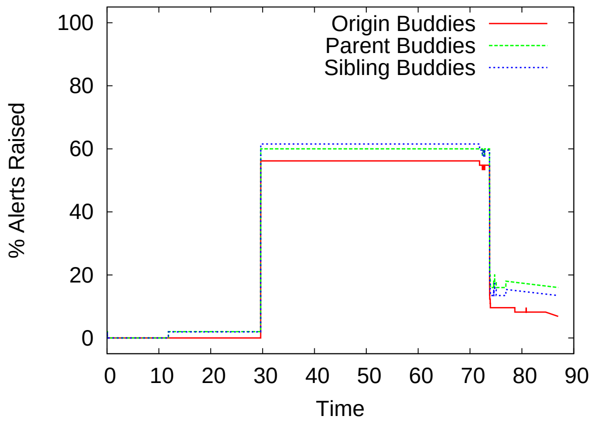
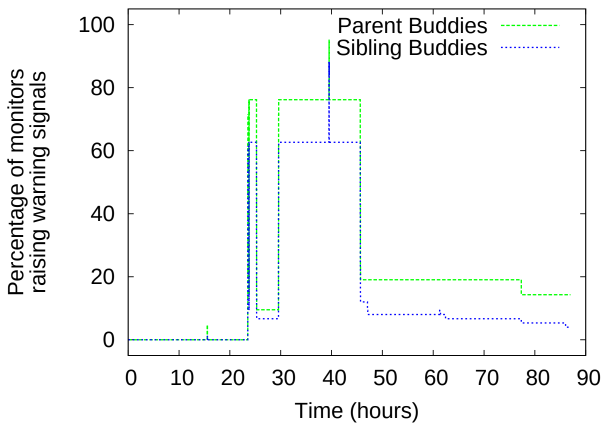


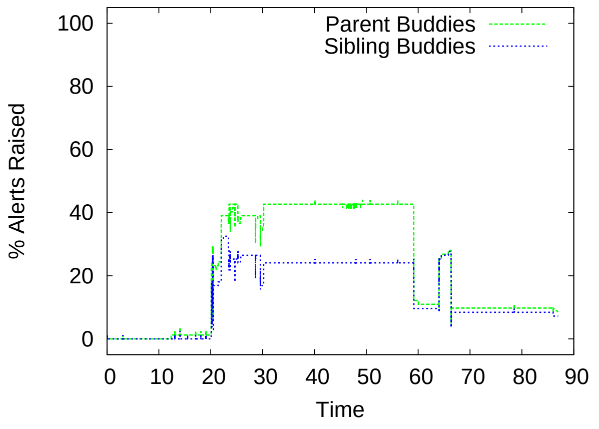
FIGURE 5.11. Warnings for when Cogent hijacked one of Google's prefixes. Time 0 is May 6, 2005 at 09:00:00 UTC. The hijack began May 7 at 14:37:56 UTC [2].

The Con Edison event, which involved more than 30 hijacked prefixes, is also easily detected for most affected prefixes. We show 4 prefixes affected by Con Edison's hijack in Figures 5.12 and 5.13. You can see that even within the same event, the results do not look the same for all the affected prefixes. The hijack's effectiveness depends on both the topological location of the victim and the topological location of the hijacker.

The Con Edison hijacking also shows the advantage to not only using prefixes from the same origin as the monitored prefix. For the prefixes belonging to Martha Stewart Living (Figure 5.12b) and Advanced Digital Internet (Figure 5.12a), there



(a) Warnings for Advanced Digital Internet, Inc.

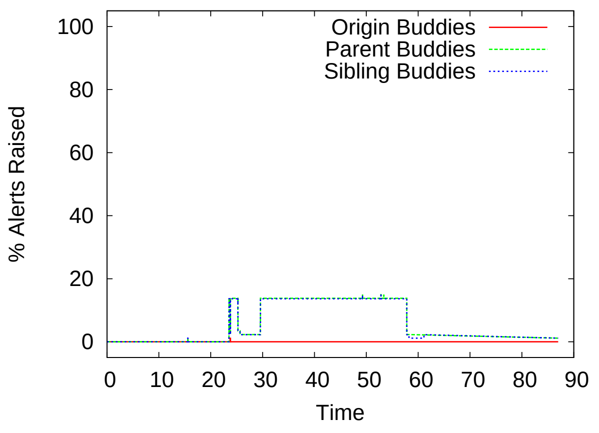


(b) Warnings for Martha Stewart Living.

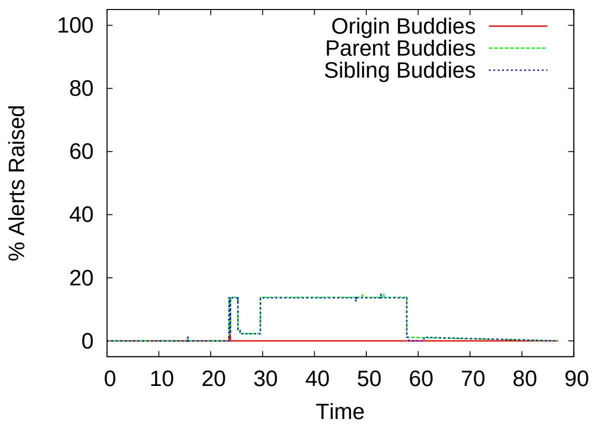
FIGURE 5.12. Percentage of monitors raising warning signals for Con-Edison hijacking events. (Note there are no results for using origin buddies because the hijacked prefix was the only prefix at its origin AS.)

were in fact no other prefixes at the origin AS. Without the sibling and parent buddies, Buddyguard would not have been able to detect anything abnormal for those prefixes. This demonstrates that our system can successfully detect prefix hijacks even when using buddies outside of the origin AS.

Similarly, in the cases of Claren (Figure 5.13a) and NYFIX (Figure 5.13b), Con Edison hijacked multiple prefixes belonging to each of them. Here, comparing to prefixes from the same origin AS does not show anomalous behavior. Since some of



(a) Warnings for Claren Road Asset Management, LLC.



(b) Warnings for NYFIX, Inc.

FIGURE 5.13. Percentage of monitors raising warning signals for Con-Edison hijacking events. (Note the event is not seen using origin buddies because the origin buddies were also hijacked.)

the origin buddies were hijacked as well we would not be able to detect the hijacking event if we could only use origin buddies. By using parent and sibling buddies, which an attacker is less likely to include in their hijack, we can see a significant increase in the number of alerts that monitors raise. Unfortunately, these prefixes also show that not all cases can be detected with the same alert threshold. These prefixes would need a threshold lower than $X = 20$ to detect the hijacking event.

Our monitoring scheme is also proven to be effective on sub-prefix hijacks. Using the YouTube hijacking as a case study (Figure 5.14), we see that when the attacking AS announced an invalid path to a sub-prefix of YouTube's prefix, our system was able to detect the hijack within *one* second of seeing the first invalid path announcement. For this event, we can see that the percentage of monitors observing the hijack is very high. This is because it is a sub-prefix hijacking. With routings running BGP, routes for a sub-prefix are preferred over routes to a parent prefix. Since the Pakistan ISP announced a sub-prefix of YouTube's prefix, the route to Pakistan was universally preferred over the route to YouTube. YouTube fought back by announcing paths for the same sub-prefix, and were able to revert some (but not all) paths around 4 hours into our monitoring period. Since a large portion of the Internet still preferred Pakistan's path, YouTube went one step further and announced a path for a sub-sub-prefix. This caused all the remaining routers to choose YouTube's path, but this is not reflected in Figure 5.14 because this figure represents the sub-prefix and not the sub-sub-prefix.

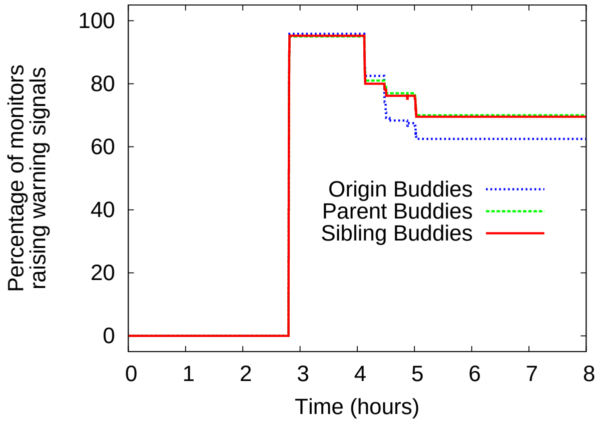


FIGURE 5.14. Warnings for when Pakistan Telecom hijacked a YouTube prefix. Time 0 is February 24, 2008 at 16:00:00 UTC. The hijack began February 24 at 18:47:57 UTC [3].

Taken together, the results show that our system provides accurate and resilient monitoring across a variety of prefix hijacking scenarios.

5.3.4.2. Detecting Route Leaks

We also tested Buddyguard on another well-known recent event: the April 4, 2010 China Telecom route leaks. We randomly selected 100 prefixes from those affected by the route leaks, and monitored them from 15:30 UTC to 16:30 UTC (the route leaks began at roughly 15:54 UTC and lasted until about 16:10 UTC). Figure 5.15 illustrates the percentage of monitor warnings raised across the aggregated sample. Once again, an alert threshold $X = 20\%$ monitors proves to be effective, detecting 90% of the (100) route leaks within seconds of the event onset. False negatives were largely due to not having enough buddies for monitoring, and as stated before such cases can be fixed by relaxing our training thresholds. As such, we see that a well-calibrated Buddyguard is adept at handling a variety of prefix anomalies, including prefix hijacking and route leaks.

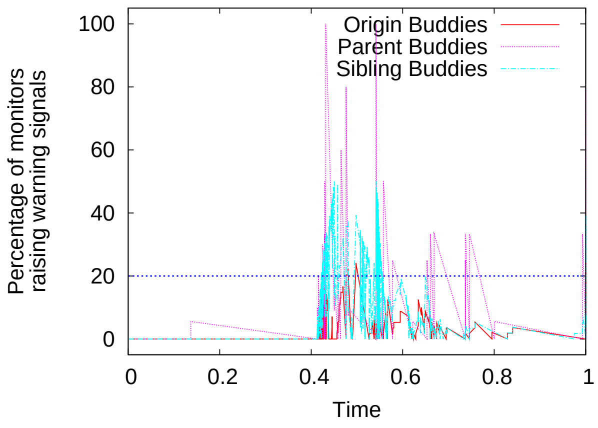


FIGURE 5.15. Aggregated warnings for 100 prefixes leaked by China Telecom on April 8, 2010. Time 0 is 15:30:00 UTC. Route leaks began at approx. 15:54:00 UTC.

It is also worth mentioning that for this event, many of the route leaks co-hijacked multiple origin buddies for several of the prefixes in our sample set. These co-hijacked origin buddies do not trigger warnings, as their (hijacked) routes match that of the monitored prefix. Here we see that such events cannot be detected by origin buddies or reference points alone; a topologically diverse set of buddies outside the origin AS was needed to detect the event. By maintaining enough valid points of comparison to the monitored prefix, Buddyguard remains resilient to such large scale route leaks. This underscores the importance of topologically diverse buddy selection, a critical component of our system.

5.3.4.3. Monitoring Normal Prefix Behavior

While it is important that Buddyguard can quickly and accurately detect prefix anomalies, we also must ensure that our system raises minimal false alerts for normally behaving prefixes. To test false positives, we randomly selected 10 prefixes each from tier 1, tier 2, and tier 3 ASes. Each prefix was trained over the week of March 29-April 5, 2010, after which we verified that training was clean and training thresholds were met. For cases where Buddyguard was able to find enough buddies, we then monitored prefixes during a clean two-day period from April 5-7, 2010. We can reasonably ensure that this time period was clean from an absence of known hijacking events and path mismatches from origin buddies that matched perfectly during training. Our reasoning for the latter is that attackers do not yet know about Buddyguard, and therefore the that an event occurred which co-hijacked all origin buddies is minimal.

Our metric for false positives is the number of warnings raised by a given monitor divided by the number of decision windows, or distinct times when that monitor decided whether to raise a warning. Figure 5.16 shows the distribution of false positives across all samples. We see immediately that with sufficient buddies, our system can monitor IP prefixes with very low false warnings. In fact, nearly 90% of our monitors raise no warnings at all.

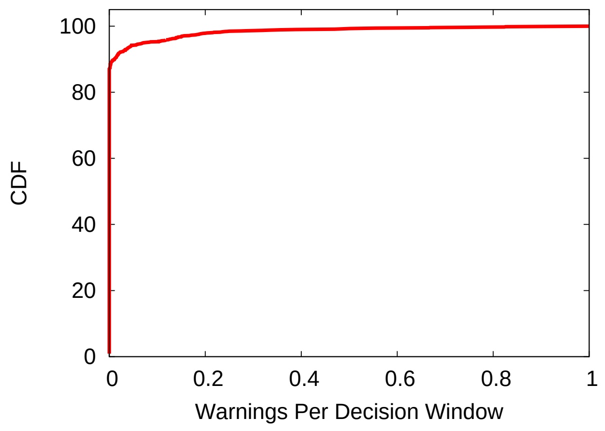


FIGURE 5.16. CDF of the mean value of warnings per decision window across all monitors and prefixes.

Furthermore, a warning raised by a single monitor does not translate directly to a false alert. Figure 5.17 shows the percent of monitors with warnings raised over the two day monitoring period across all samples. If we use the previously observed system-wide alert threshold $X = 20\%$, only about 6% of the decision windows actually peak above this threshold. Moreover, all of these instances occurred *for a single prefix*—in other words, only one prefix would have raised false alerts. While we discuss later how this observed threshold might be better calibrated, for now it is enough to say that $X = 20\%$ works to balance both false negatives and false positives. Therefore, our monitoring tests on prefix hijacking, route leaks, and normal prefix behavior show that our system provides accurate and resilient monitoring of IP prefixes.

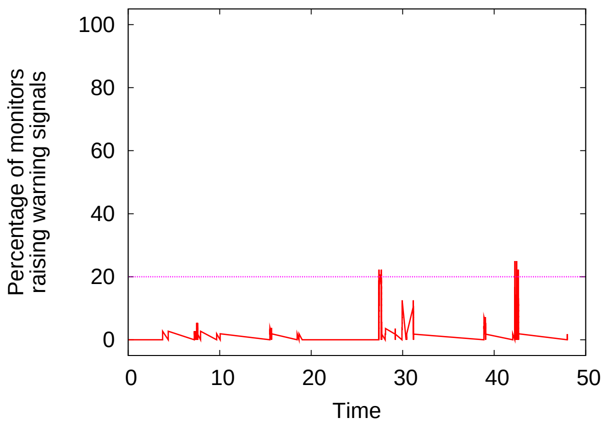


FIGURE 5.17. Warnings raised for all prefixes in our false positive tests.

5.3.5. Performance: Latency and Overhead

Finally, we evaluate Buddyguard with respect to its overall performance—namely, the latency and overhead involved. Since Buddyguard analyzes BGP updates in close to real time, detecting anomalies can be achieved with little latency. In the hijack and misconfiguration cases described above, detection occurs within minutes (and often seconds) of the event. This is a crucial point; prefix anomalies must be detected and addressed immediately. Our system provides fast detection of anomalous events, giving prefix owners the chance to minimize the damage done.

Even with a standard server, a single thread could process 87 hours of monitoring in less than half a second (Figure 5.1). In other words, Buddyguard could process approximately 760,000 prefixes on average at a time with a single thread. CAIDA [120] states that most routers have much less than that–typically around only 320,000 prefixes. Therefore, Buddyguard can handle monitoring every prefix on the Internet with standard resources.

Our system design also maintains low storage cost, an important measure when considering how Buddyguard might scale to monitor numerous IP prefixes. During

TABLE 5.1. Processing Time Used to Monitor a Prefix for 87 Hours, With a Single Thread on a Sun SPARC Enterprise T5120.

	seconds
Average	0.4122107213
5^{th} Percentile	0.00012265
Median	0.0557723
95^{th} Percentile	1.0834156

the monitoring process, the storage cost for every monitored prefix is mainly (1) the current AS path from the monitor to the prefix and (2) a set of buddy prefixes. One bit of auxiliary data that the monitor may also store for the monitored prefix is (3) its set of good AS paths to that prefix. Assuming the average length of an AS path is 4, a prefix has 1,000 buddies on average, and a prefix has on average 10 AS paths known to be legitimate, the total storage cost per monitored prefix will be approximately 4.2 KB. Therefore, monitoring 100 prefixes would only cost about 400 KB, and 1,000 prefixes only about 4 MB. If we again use CAIDA's measurement of most routers typically storing around 320,000 prefixes [120], then total storage comes out to 1.344 GB at most. Thus, it is very feasible for Buddyguard to monitor even every prefix on the Internet. When coupled with pre-existing BGP monitoring systems like RouteViews, Buddyguard is lightweight, scalable, and easily deployable for today's Internet.

5.4. Open Hijacking Issues

Using our conservative approach toward defining the path similarity threshold $n = 1$, we were not able to find enough buddies for a small number of prefixes. However, these edge cases can be addressed by defining n adaptively: if $n = 1$ is not sufficient, continue incrementing n and searching for candidates. Proceeding in this manner, we can easily find enough buddy candidates for all prefixes. However, n has a direct effect on how well buddy candidates match the monitored prefix. For a given prefix p and two buddies b_0 and b_5 that are 0 and 5 AS hops away from p respectively, the path comparison between p and b_0 is much more strict; the paths to p and b_0 will need to share 5 more AS hops than the paths to p and b_5. Furthermore, if an attacker is close to p, what is the chance that the path to b_5 will even change if p is hijacked? Investigating the intricacies of searching for buddy candidates adaptively is beyond the scope of this work. Having proved the efficacy of our monitoring system for the majority of cases, we leave this as a topic for future study.

In addition, many of the other metrics that we use for this work could also be further optimized. The thresholds used for training are by no means absolute; we use them to demonstrate Buddyguard's effectiveness under constrained conditions. Moreover, while the observed system-wide alert threshold $X = 20\%$ works to minimize false negatives and false positives during monitoring, it is certainly not the only or best threshold for this purpose. A better design might involve using a "gradient" alert scale, where we determine system-wide alert status using ranges for suspicious, anomalous, or perhaps even additional monitor warning levels. We could also dynamically configure the alert threshold by normalizing a monitor's warnings by the quality of its buddies, allowing monitors with better buddies to have more weight in Buddyguard's decision making. It is also quite possible to leave the threshold settings as a configurable option for the monitored prefix. Some prefix administrators may be concerned about missing the smallest little hijack, and opt for a very low alert threshold. Other administrators may be concerned about receiving too many false positives, and opt for a higher threshold. An entire study could be dedicated to optimizing this threshold, making such calibrations beyond the scope of this work.

A concern that may arise with Buddyguard is the percentage of AS paths on the Internet that Buddgyguard can actually observe. There are many AS links, and thus AS paths, that we cannot see [121, 122, 123, 124], so one may worry that Buddyguard may miss hijacking attacks. It is important to note that most of the missing links are peer-peer links, not customer-provider links; the paths used by these missing links will be shorter than the paths possible without those links. Because they are peer-peer links, paths using those links do not propagate very far—they are not very useful for a hijacking attack. Similarly, since the paths using the missing links will be shorter than would otherwise be possible, the paths using the links are difficult for an attacker to hijack. Although the missing links do open up some holes that may benefit an attacker, the limitations are so great that it is not a serious concern. If we are able to limit attackers to only be able to hijack if they are able to utilize such hidden peer-peer links, then we would be much better off than we are today and we could arguably have succeeded in protecting the majority of the Internet's prefixes. We will have forced attackers to work too hard to gain too little.

5.5. Recap

In this chapter we presented Buddyguard, a new effective and resilient system for detecting IP prefix hijacking. Buddyguard improves upon existing state of the art research. The core of Buddyguard is the idea that legitimate routing changes should affect similarly routed prefixes in a similar fashion. Through our novel use of "buddies" the system can learn what is normal for a monitored prefix. Buddyguard lives in the control plane of the Internet, observing routing changes for a monitored prefix along with other prefixes that have shown similar behavior. In the next chapter we will provide further discussion on issues facing Buddyguard and SAVE.

CHAPTER VI

DISCUSSION

Now we have presented our contributions to both sides of the IP layer identity problem. On the source address side, we first showed why the existing research was inadequate and then presented a new version of the SAVE protocol. This new SAVE system is able to effectively identify spoofing packets, mitigate spoofing attacks, and assist in pin-pointing an attackers true location. Furthermore, it is resilient against circumvention, is easily deployable, is routing protocol independent, and has low overhead. On the destination address side, we covered the missing gaps in the current state of the art prefix hijacking defenses. Against simple attacks the current defenses are often adequate, but more intelligent attackers are able to circumvent detection without too much difficulty. To address this circumvention issue we presented Buddyguard, which provides a fast and resilient IP prefix hijacking mechanism. In this chapter we discuss additional issues relating to our contributions of SAVE and Buddyguard.

6.1. Who Needs Buddyguard With Secured BGP?

One question readers may have about Buddyguard is, why should research even focus on detecting IP prefix hijacking when securing BGP would simply solve the problem? Certainly, securing BGP is a very active area of research [125]. But it has been an active area for many years and nobody knows when any of the proposed solutions will actually be deployed in the real world. Especially since making drastic changes to BGP is nearly impossible to coordinate across all the BGP speaking routers of the Internet. Furthermore, even assuming BGP is fully secured, misconfigurations and simple human error may also cause routes to become hijacked, even if it is not malicious. In such a scenario we would still need to be able to reliably detect such IP prefix hijacks.

There has even been research showing that some “secure” BGP extensions are still vulnerable to IP prefix hijacking attacks [126]. Without enough adversarial thinking during their design, extensions to BGP that were supposed to *increase* security and *block* prefix hijacking attacks instead created *more* avenues of attack! In some cases, the hijacking attacks made possible by the secure BGP extensions actually end up affecting an even larger portion of the Internet than the standard hijacking attacks possible today! Clearly, an effective and resilient IP prefix hijacking detection system would still be a valuable asset.

Buddyguard also lends itself to further usage beyond just IP prefix hijacking. The notion of using “buddy” prefixes to monitor some prefix-level behavior is a novel concept, and may be used for other prefix-level analyses. Some anomalies to extend the Buddyguard research to include:

– Connectivity degradation: When a prefix has degraded connectivity to its ISP, it would likely notice this on its own. However, if the degradation is due to behavior farther away, a prefix may be oblivious to the change. For instance, if a prefix’s parent AS loses its connection to a preferred peer, and is forced to use a peer with poor routing performance, the degradation may not be quickly evident to the prefix.

– Anomalous routing dynamics: A prefix can experience pathological routing dynamics and thus degraded data delivery performance [72, 69]. However, because of the highly dynamic nature of BGP routing, it is hard to judge whether or not a prefix is experiencing some anomalous dynamics solely based on BGP updates for that prefix.

– Anomalous behavior during disruptive events: Previous studies have looked into how disruptive events such as Internet worms, earthquakes, floods, and power outages have affected the Internet [82, 83, 84]. While it is expected for such events to hinder the performance of prefixes in affected areas, the connectivity of some prefixes will be more resilient to the events and some less resilient.

6.2. Complementary Defenses

With both SAVE and Buddyguard, the systems were created and evaluated in isolation. How would they complement each other if considered together? Does one rely on the other? They both aim to protect the identity at the IP layer—surely they would work better when deployed together, right? If there is already a system guaranteeing source address validity, does that make an IP prefix hijacking system easier to deploy? Similarly, if can know for sure an IP prefix is not hijacked, does that make source address validity easier to prove?

First, let us consider the case where SAVE is deployed throughout a portion of the Internet, and we can be sure nobody is spoofing SAVE-protected source addresses. In such a case, IP prefix hijacking will still be an issue. When an attacker hijacks a prefix with SAVE-protected addresses, SAVE may prevent the hijacker from successfully sending packets from the hijacked subnet, but it would *n*ot prevent the hijacker from receiving packets destined for the hijacked subnet. When a prefix is hijacked it may even prevent the victim network from properly receiving pushback notices, and it would not know to send an on-demand update to correct any outdated blacklist information downstream. This is because SAVE relies on the underlying routing to be correct. If packets are not properly forwarded to their intended destination, SAVE may not function as intended. Solving IP spoofing does not give an advantage to detecting IP prefix hijacking, and in fact SAVE relies on there being no IP prefix hijacking.

Now, for the opposite scenario—suppose we could know for sure an IP prefix would not be hijacked, perhaps by BGP being fully secured. We know that if a packet is sent to a given address it will reach the real owner of that address. But this does not help to learn the incoming direction of a packet or stop attackers from spoofing. Without hijacking a prefix, attackers can still send spoofed IP packets. As mentioned above, SAVE's design assumes packets will reach their intended destinations, since SAVE treats prefix hijacking as an out-of-scope problem. Solving prefix hijacking does not give routers any information about the proper incoming direction of a packet.

6.3. Prevention Versus Detection

One big difference between our contributions of SAVE and Buddyguard is that SAVE is a detection and prevention system, while Buddyguard is a detection only system. Why do we detect and prevent on one side, but only detect on the other side? With IP spoofing, detection goes hand-in-hand with prevention. If a router decides a packet is suspicious, it can ask the source router to confirm its incoming direction by sending an update. If a packet is in fact spoofing, the router can simply drop it instead of forwarding it along. With a router-based detection system such as SAVE, detection essentially equals prevention. A router can decide on its own that a packet is spoofing and take action on its own. For IP prefix hijacking, however, detection and prevention are not so tightly coupled. Our detection mechanism relies on being able to monitor path updates from many vantage points. Due to the dynamic nature of routing, it is not feasible for a router to decide on its own that an update is malicious, and that it should not propagate the update further. Even assuming that a monitoring router receiving a malicious BGP update message can decide on its own that the update is suspicious, how would it ask the monitored prefix to confirm? It cannot forward a query to the prefix, since if it is hijacked the attacker would receive the query. And even if the monitored prefix could receive the query, how would it respond? The path from the monitored prefix to the monitoring router may not be the same as the path from the monitoring router to the monitored prefix. For Buddyguard we focus on detection without prevention because for prefix hijacking they are separate problems. Detection alone is enough of an open problem for prefix hijacking that this is a useful contribution to the field.

6.4. Deployment

We discussed deployment for SAVE and Buddyguard to some degree in their respective Chapters. However, we have seen that with security protocols there may need to be a more forceful push for deployment. DNSSEC has been researched and plans have been made for deployment for years [127, 128, 129, 130, 131, 132]. There were some domains that deployed DNSSEC somewhat earlier, but there was not much progress on deploying to the larger domains until the US government mandated that the root for .gov would be signed by January 2009 and all of .gov would be secured by DNSSEC by December 2009 [133].

When a solid solution is created to an important Internet security problem, such as identity at the IP layer, government mandates could be a good method of ensuring deployment. Without someone to start deployment it becomes somewhat of a chicken and egg problem. With a mandated deployment for at least some portion of the Internet, it makes it much easier for other networks to rationalize a deployment of their own.

For SAVE specifically, one thing that may help it get deployed is developments in software defined networking [134, 135]. With such systems available on routers, it makes it relatively easy to prototype, test, and deploy a router-based system such as SAVE on real networking hardware.

Another important issue relating to deployment is where SAVE or Buddyguard would be deployed.

For SAVE we looked at a vertex cover, random deployment, and deployment as high-degree ASes. With a real world deployment, the distribution of SAVE routers would likely be somewhat different than the cases we evaluated. Initially we could expect to have some high-degree ASes along with some smaller-degree ASes at random points in the Internet. Additional deployment distribution strategies, and what level of effectiveness we could expect from them, is an area of future research.

In evaluating Buddyguard we used BGP speakers peered with RouteViews collectors as our deployment locations. An important question is the extent of coverage provided by these BGP speakers. They provide an excellent resource, with the huge archive of BGP data allowing us easy access to dozens of vantage points throughout the Internet. But is the coverage enough? Would alternate deployment locations be needed to provide better coverage? A previous work by Zhang et al. [136] examined this issue in detail, although the focus was on general route monitoring—not a buddy-based monitoring system. A follow up study to our work would be to compare coverage of other monitoring systems such as RIPE [85] and BGPMon [118], or investigating other monitor deployment strategies not limited to existing BGP monitoring systems.

6.5. Recap

In this chapter we have discussed further issues relating to both of our main contributions to protecting identity at the IP layer of the Internet. With these issues out of the way, we conclude this dissertation in the following chapter.

CHAPTER VII

CONCLUSION

Our research provides new insight into, and mechanisms to defend against, identity theft at the IP layer of the Internet.

Exploring identity from the source address perspective, we evaluated existing spoofing defense mechanisms in terms of three features: identifying spoofing packets, mitigating spoofing attacks, and pinpointing an attacker's real location. We further measured them against four desired properties: resilient against circumvention, easily deployable, independent of routing protocols, and low overhead. Finding the existing systems to be lacking, we then presented a new version of SAVE that has all of the necessary and desired features.

On the destination address side, we similarly show that the existing IP prefix hijacking detection systems leave cracks that intelligent attackers can sneak through to circumvent detection. Our new detection system, Buddyguard, gives defenders an effective and resilient tool to discover IP prefix hijacking attacks.

These contributions help advance the state of the art in protecting identity at the IP layer of the Internet. We further detail the contributions below in Sections 7.1 and 7.2, and then conclude by reviewing some additional lessons learned throughout the research.

7.1. IP Spoofing

Research has shown that if a small percentage of routers throughout the Internet deploy a filtering table to discard packets with a forged source address, a synergistic filtering effect can be achieved to stop a large fraction of spoofed IP packets. Such an approach to IP spoofing has also been found to be the most effective. However, in building such a filtering table, specifically an incoming table, we have found that the previously designed SAVE protocol is susceptible to obsolete incoming table entries as it is incrementally deployed.

This research introduces new mechanisms to address this deficiency. We introduce blacklists at SAVE routers and use both the blacklist and the incoming table to classify and filter incoming packets. Our on-demand mechanism enables a SAVE router to deal with suspicious packets and update its incoming table, and the pushback mechanism further pushes the filtering of spoofing packets toward the SAVE router that is the closest to spoofers. With these new mechanisms, and with both security and performance issues considered, we show that incoming-table-based IP spoofing detection is a viable approach to addressing the critical problem of IP spoofing, and that ASes (beginning with high-degree ASes) will have incentives to deploy such a solution. Simulations show that, for example, with deployment at only the top 0.08% of ASes by degree, the efficacy of catching spoofing packets is over 90%.

7.2. IP Prefix Hijacking

The current state of prefix-level monitoring leaves much to be desired; today's systems are limited in scope and underestimate the capacity of intelligent attackers. While the eventual deployment of S-BGP could address many of these problems, widespread adoption is years away and we will still need tools for understanding prefix routing behavior. Such knowledge is critical for addressing both current and future network anomalies.

In this work we introduced Buddyguard, a buddy system-based monitoring scheme that provides accurate detection of various prefix anomalies and is resilient to targeted countermeasures. Buddyguard is able to detect all manner of prefix hijacks and misconfigurations, and remains highly extensible to new anomalies as they emerge. We rigorously evaluated Buddyguard against known hijacking events and misconfigurations, and demonstrate the accuracy and efficacy of our buddy system design. This work represents a major step forward in the accurate and resilient monitoring of IP prefixes and prefix-level anomalies.

7.3. Additional Lessons Learned

In addition to the main contributions that we set out to research, we made further discoveries as we went.

While analyzing existing defenses and designing our new defenses, we found that adversarial thinking is extremely important. Many of the previous works proposed solutions that would defend against current methods of attack, without considering how intelligent attackers would change their methods to circumvent new defense systems. Without incorporating adversarial thinking into the design, attackers will always be able to be a step ahead of the defenders. We saw that even security enhancements for BGP opened up avenues for new kinds of IP prefix hijacking attacks. Researchers need to put themselves in an attacker's mindset, and try to punch holes in their own works. It is a difficult but necessary part of a truly secure system. Defense researchers must think about not only current attacks but possible future attacks, in order to create a defense system that will last and not simply become obsolete right away.

While exploring existing IP spoofing defenses we came to the conclusion that router based solutions are in general preferred to end-host based systems. They allow greater protection with fewer deployed locations, and they are the only way to catch packets before they reach a victim network where it is often too late to mitigate the damage. We further discovered how important it is for such systems to only rely on characteristics that attackers cannot influence. Many systems propose using packet marking mechanisms, but such mechanisms open up the defense system itself to attack. If an attacker can influence or directly change some characteristic of the packet, that characteristic should not be relied upon in a defense system. Our exploration also taught us that incremental deployability is of utmost importance. Some existing systems would work beautifully and simply, if only everyone on the Internet would begin deploying them at the same time. In the real world that will never happen, and so deployment incentives and efficacy with incremental deployment must be considered with any new protocol or distributed defense system development.

One interesting product of the SAVE research was the pushback mechanism. While we did not initially set out to create a system that could be used for pushback, it naturally evolved as we created the necessary mechanisms for SAVE. A pushback system can prove extremely useful in defending against Denial of Service (DoS) attacks [137]. This artifact of our SAVE research is a great example of why it is necessary to have a good understanding of not only the specific problem you are trying to solve, but also of other areas related to your research. Contributions in one area often have useful implications in other areas.

While researching IP prefix hijacking, an issue that we ran into involved finding a good threshold for deciding if data was showing normal behavior or hijacking behavior. Attempting to come up with a single threshold value or ratio never lead us to good results. A lower threshold created false positives when monitoring some prefixes, and a higher threshold created false negatives when monitoring other prefixes. Finally we were able to see an improvement by moving to a variable threshold, allowing a different threshold to be set for each monitored prefix. The normal behavior of each prefix is unique, so the threshold fore each prefix should also be unique. This underlines the importance of an abnormality detection system to learn what normal is before trying to detect what is abnormal.

Another "bonus" lesson we found through our hijacking research is that our buddy-based mechanism could be useful for more than just hijacking. The idea of using buddies that show similar behavior to define normal is a powerful idea that we think can be further explored. Anomaly detection generally falls into two types: defining what is abnormal and detecting abnormalities, or defining what is normal and detecting when something deviates from normal. It is often easier for researchers to define what is abnormal, but it is nearly impossible to come up with every possible abnormality—something is always missing. Defining normal has proven to be generally more difficult, but with the idea of using similar behaved buddies to help define normal it becomes an attainable goal.

REFERENCES CITED

[1] MIT Advanced Network Architecture Group, "ANA Spoofer Project," 2007, http://spoofer.csail.mit.edu/.

[2] T. Wan and P. C. van Oorschot, "Analysis of BGP prefix origins during Google's May 2005 outage," in *Proceedings of IPDPS*, 2006.

[3] RIPE NCC, "YouTube hijacking: A RIPE NCC RIS case study," http://www.ripe.net/news/study-youtube-hijacking.html.

[4] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *ACM Computer Communications Review (CCR)*, vol. 31, no. 3, pp. 38–47, July 2001. [Online]. Available: citeseer.ist.psu.edu/paxson01analysis.html

[5] D. Jackson, "DNS amplification variation used in recent DDoS attacks," February 2009, http://www.secureworks.com/research/threats/dns-amplification/. [Online]. Available: http://www.secureworks.com/research/threats/dns-amplification/

[6] J. Touch, "Defending TCP against spoofing attacks," RFC 4953, 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4953.txt

[7] US-CERT, "Multiple DNS implementations vulnerable to cache poisoning," July 2008, vulnerability Note VU 800113. [Online]. Available: http://www.kb.cert.org/vuls/id/800113

[8] C. Morrow, "BLS FastAccess internal tech needed," January 2006, http://www.merit.edu/mail.archives/nanog/2006-01/msg00220.html. [Online]. Available: http://www.merit.edu/mail.archives/nanog/2006-01/msg00220.html

[9] R. Beverly, A. Berger, Y. Hyun, and k claffy, "Understanding the efficacy of deployed Internet source address validation filtering," in *Proceedings of the ACM Internet Measurement Conference*, November 2009.

[10] A. Bremler-Barr and H. Levy, "Spoofing prevention method," in *Proceedings of IEEE INFOCOM*, 2005.

[11] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets," in *ACM SIGCOMM*, 2001, pp. 15–26.

[12] R. Beverly and S. Bauer, "The Spoofer Project: Inferring the extent of source address filtering on the Internet," in *Proc. USENIX SRUTI*, 2005, pp. 53–59.

[13] T. Underwood, “Con-Ed steals the ’Net - Renesys blog,” 2006. [Online]. Available: http://renesys.com/blog/2006/01/coned-steals-the-net.shtml

[14] H. Ballani, P. Francis, and X. Zhang, “A study of prefix hijacking and interception in the Internet,” 2007, pp. 265–276. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282411

[15] ”BGPmon”, “BGP prefix hijack by AS16735,” November 2008, http://bgpmon.net/blog/?p=80?

[16] T. Underwood, “Internet-wide catastrophe-last year,” 2005. [Online]. Available: http://www.renesys.com/blog/2005/12/internetwide_nearcatastrophela.shtml

[17] ”BGPmon”, “Chinese ISP hijacked 10% of the internet,” 2010, http://bgpmon.net/blog/?p=282.

[18] T. Ehrenkranz and J. Li, “On the state of IP spoofing defense,” *ACM Transactions on Internet Technology*, vol. 9, no. 2, May 2009.

[19] T. Ehrenkranz, J. Li, and P. McDaniel, “Realizing a source authentic Internet,” in *Proceedings of the Conference on Security and Privacy in Communications Networks*, Sep. 2010.

[20] P. Elliott, T. Ehrenkranz, and J. Li, “A buddy system for fast and reliable detection of IP prefix anomalies,” University of Oregon, Tech. Rep. CIS-TR-2012-02’, month =.

[21] D. M. Piscitello, “Anatomy of a DNS DDoS amplification attack,” 2006, http://www.watchguard.com/infocenter/editorial/41649.asp.

[22] E. Messmer, “Report says identity thieves working hand in hand with ‘bot herders’,” *Network World*, March 2007. [Online]. Available: http://www.networkworld.com/news/2007/031907-identity-thieves-bot-herders.html

[23] K. Martin, “Stop the bots,” *Security Focus*, April 2006. [Online]. Available: http://www.securityfocus.com/columnists/398

[24] J. H. Saltzer, D. P. Reed, and D. D. Clark, “End-to-end arguments in system design,” *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277–288, 1984.

[25] S. Kent and K. Seo, “Security architecture for the Internet Protocol,” RFC 4301, IETF, 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4301.txt

[26] Fyodor, “Remote OS detection,” 2006, http://nmap.org/book/osdetect.html.

[27] M. Zalewski, “Passive OS fingerprinting tool,” 2006, http://lcamtuf.coredump.cx/p0f.shtml.

[28] R. Beverly, "A robust classifier for passive TCP/IP fingerprinting," in *Proceedings of the Passive and Active Measurement Conference*, 2004, pp. 158–167.

[29] G. Taleck, "Ambiguity resolution via passive OS fingerprinting." in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*, 2003, pp. 192–206.

[30] M. Zalewski, "Strange attractors and TCP/IP sequence number analysis," 2001, http://lcamtuf.coredump.cx/oldtcp/.

[31] ——, "Strange attractors and TCP/IP sequence number analysis — one year later," 2002, http://lcamtuf.coredump.cx/newtcp/.

[32] D. J. Bernstein, "SYN cookies," 1996, http://cr.yp.to/syncookies.html.

[33] T. Aura and P. Nikander, "Stateless connections," in *ICICS*, ser. Lecture Notes in Computer Science, Y. Han, T. Okamoto, and S. Qing, Eds., vol. 1334. Springer, 1997, pp. 87–97.

[34] W. chang Feng, E. C. Kaiser, W. chi Feng, and A. Luu, "Design and implementation of network puzzles," in *Proceedings of IEEE INFOCOM*, 2005, pp. 2372–2382.

[35] C. Jin, H. Wang, and K. G. Shin, "Hop-count filtering: An effective defense against spoofed DDoS traffic," in *Proceedings of the Conference on Computer and Communications Security*, 2003, pp. 30–41.

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

[37] S. J. Templeton and K. E. Levitt, "Detecting spoofed packets," in *Proceedings of the DARPA Information Survivability Conference and Exposition*, vol. 1, 2003, pp. 164–175.

[38] F. Baker, "Requirements for IP Version 4 routers," RFC 1812, IETF, 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1812.txt

[39] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," RFC 2827, IETF, 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2827.txt

[40] T. Killalea, "Recommended Internet service provider security services and procedures," RFC 3013, IETF, 2000. [Online]. Available: http://www.ietf.org/rfc/rfc3013.txt

[41] F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks," RFC 3704, IETF, 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3704.txt

[42] Y. He, M. Faloutsos, and S. V. Krishnamurthy, "Quantifying the routing asymmetry in the Internet at the AS level," in *Proceedings of IEEE GLOBECOM*, 2004.

[43] ——, "On routing asymmetry in the Internet," in *Proceedings of IEEE GLOBECOM*, 2005.

[44] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin, "Working around BGP: An incremental approach to improving security and accuracy of interdomain routing," in *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, February 2003.

[45] Cisco Systems Inc., "TCP intercept," 1998. [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/product/software/ios112/intercpt.htm

[46] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and adoptable source authentication," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*, 2008.

[47] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, November 1976.

[48] J. Li, J. Mirkovic, M. Wang, P. L. Reiher, and L. Zhang, "SAVE: Source address validity enforcement protocol," in *Proceedings of IEEE INFOCOM*, New York, June 2002, pp. 1557–66.

[49] Z. Duan, X. Yuan, and J. Chandrashekar, "Constructing inter-domain packet filters to control IP spoofing based on BGP updates," in *Proceedings of IEEE INFOCOM*, 2006.

[50] H. Lee, M. Kwon, G. Hasker, and A. Perrig, "BASE: An incrementally deployable mechanism for viable IP spoofing prevention," in *Proceedings of the ACM Symposium on Information, Computer, and Communication Security*, 2007.

[51] A. Yaar, A. Perrig, and D. Song, "Pi: A path identification mechanism to defend against DDoS attack," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2003, pp. 93–107.

[52] ——, "StackPi: New packet marking and filtering mechanisms for DDoS and IP spoofing defense," *IEEE Journal of Selected Areas in Communications*, vol. 24, no. 10, pp. 1853–1863, October 2006.

[53] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer, "Single-packet IP traceback," *IEEE/ACM Trans. on Networking*, vol. 10, no. 6, pp. 721–734, 2002.

[54] S. Savage, D. Wetherall, A. R. Karlin, and T. E. Anderson, "Practical network support for IP traceback," in *ACM SIGCOMM*, Stockholm, Sweden, 2000, pp. 295–306.

[55] M. Adler, "Trade-offs in probabilistic packet marking for IP traceback," *Journal of the ACM*, vol. 52, no. 2, pp. 217–244, 2005.

[56] D. Dean, M. K. Franklin, and A. Stubblefield, "An algebraic approach to IP traceback," *ACM Transactions on Information and System Security*, vol. 5, no. 2, pp. 119–137, 2002.

[57] T. Ehrenkranz and J. Li, "An incrementally deployable protocol for learning the valid incoming direction of IP packets," Tech. Rep.

[58] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), IETF, 2006.

[59] J. Moy, "OSPF Version 2," RFC 2328 (Standard), IETF, 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt

[60] D. Oran, "OSI IS-IS Intra-domain Routing Protocol," RFC 1142 (Informational), IETF, 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1142.txt

[61] B. Albrightson, J. Garcia-Luna-Aceves, and J. Boyle, "EIGRP—A fast routing protocol based on distance vectors," in *Networld/Interop*, May 1994.

[62] J. Wu, G. Ren, and X. Li, "Source address validation: Architecture and protocol design," in *Proceedings of ICNP*, 2007.

[63] J. Li, J. Mirkovic, T. Ehrenkranz, M. Wang, P. Reiher, and L. Zhang, "Learning the valid incoming direction of IP packets," *Computer Networks*, vol. 52, no. 2, pp. 399–417, February 2008.

[64] J. Mirkovic and E. Kissel, "Comparative evaluation of spoofing defenses," *IEEE Transactions on Dependable and Secure Computing*, vol. 99, no. PrePrints, 2009.

[65] S. Kent, C. Lynn, J. Mikkelson, and K. Seo, "Secure border gateway protocol (S-BGP) — real world performance and deployment issues," in *Proceedings of the Network and Distributed System Security Symposium*, 2000.

[66] R. Views, "University of oregon route views project," 2000.

[67] H.-Y. Tyan, A. Sobeih, and J. C. Hou, "Towards composable and extensible network simulation," in *Proceedings of the International Parallel and Distributed Processing Symposium*, 2005, pp. 225–232.

[68] P. Mahadevan, C. Hubble, D. V. Krioukov, B. Huffaker, and A. Vahdat, "Orbis: rescaling degree correlations to generate annotated Internet topologies," in *ACM SIGCOMM*, 2007, pp. 325–336.

[69] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," in *ACM SIGCOMM*, 1997, pp. 115–126.

[70] C. Labovitz, G. Malan, and F. Jahanian, "Origins of Internet routing instability," in *Proceedings of IEEE INFOCOM*, March 1999.

[71] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs, "Locating Internet routing instabilities," in *ACM SIGCOMM*, August 2004.

[72] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, "BGP routing dynamics revisited," 2007, to appear.

[73] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proceedings of ACM SIGCOMM*, August 2002. [Online]. Available: citeseer.nj.nec.com/mahajan02understanding.html

[74] J. Fartar, "C&W routing instability," http://www.merit.edu/mail.archives/nanog/2001-04/msg00209.html.

[75] S. Misel, "Wow, AS7007!" http://www.merit.edu/mail.archives/nanog/1997-04/msg00340.html.

[76] J. Cowie, A. Ogielski, B. Premore, and Y. Yuan, "Internet worms and global routing instabilities," in *Proc. of SPIE International symposium on Convergence of IT and Communication*, July 2002.

[77] J. Cowie, A. Ogielski, B. Premore, E. Smith, and T. Underwood, "Impact of the 2003 blackouts on Internet communications," http://www.renesys.com/news/2003-11-21/Renesys_BlackoutReport.pdf, November 2003.

[78] M. Lad, X. Zhao, B. Zhang, D. Massey, and L. Zhang, "An analysis of BGP update burst during Slammer worm attack," in *Proceedings of 5th International Workshop on Distributed Computing*, December 2003.

[79] J. Li, D. Dou, Z. Wu, S. Kim, and V. Agarwal, "An Internet routing forensics framework for discovering rules of abnormal BGP events," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 55–66, October 2005.

[80] S. Teoh, K. Ma, S. Wu, D. Massey, X. Zhao, D. Pei, L. Wang, L. Zhang, and R. Bush, “Visual-based anomaly detection for BGP origin AS change (OASC) events,” in *Proceedings of the Distributed Systems, Operations, and Management Workshop*, 2003.

[81] K. Zhang, A. Yen, X. Zhao, D. Massey, F. S. Wu, and L. Zhang, “On detection of anomalous routing dynamics in BGP,” in *Proceedings of the International IFIP-TC6 Networking Conference*, 2004, pp. 259–270.

[82] J. Li, D. Dou, S. Kim, H. Qin, and Y. Wang, “On knowledge-based classification of abnormal BGP events,” in *Proceedings of the International Conference on Information Systems Security*, December 2007, pp. 267–271 (short paper).

[83] D. Dou, J. Li, H. Qin, S. Kim, and S. Zhong, “Understanding and utilizing the hierarchy of abnormal BGP events,” in *SIAM International Conference on Data Mining*, Minneapolis, Minnesota, April 2007, pp. 457–462 (short paper).

[84] Z. Wu, E. S. Purpus, and J. Li, “BGP behavior analysis during the August 2003 blackout,” in *The 9th IFIP/IEEE International Symposium on Integrated Network Management*, Nice, France, May 2005, 4 pages (short paper).

[85] RIPE NCC, “RIPE Routing Information Service,” http://www.ris.ripe.net/.

[86] R. V. Oliveira, M. Lad, and L. Zhang, “Cyclops,” http://cyclops.cs.ucla.edu/.

[87] B. A. Prakash, N. Valler, D. Andersen, M. Faloutsos, and C. Faloutsos, “BGP-lens: Patterns and anomalies in Internet routing updates,” in *Proc. of ACM SIGKDD*, 2009.

[88] K. T. Latt, Y. Ohara, S. Uda, and Y. Shinoda, “Analysis of ip prefix hijacking and traffic interception,” *International Journal of Computer Science and Network Security*, vol. 10, no. 7, pp. 22–31, 2010.

[89] M. Lad, R. Oliveira, B. Zhang, and L. Zhang, “Understanding resiliency of Internet topology against prefix hijack attacks,” in *Proceedings of the International Conference on Dependable Systems and Networks.* Washington, DC, USA: IEEE Computer Society, 2007, pp. 368–377.

[90] R. V. Oliveira, M. Lad, and L. Zhang, “A routing leak with spaghetti sauce,” October 2009, http://cyclops.cs.ucla.edu/blog/?p=91.

[91] ——, “Cyclops detects global routing leak by AS13214,” May 2009, http://cyclops.cs.ucla.edu/blog/?p=78.

[92] ——, “Evolva Telecom (AS30890) under attack,” February 2009, http://cyclops.cs.ucla.edu/blog/?p=35.

[93] Z. Zhang, Y. Zhang, Y. C. Hu, and Z. M. Mao, "Practical defenses against BGP prefix hijacking," in *Proceedings of CoNEXT*. ACM, 2007, pp. 1–12.

[94] T. Qiu, L. Ji, D. Pei, J. Wang, J. Xu, and H. Ballani, "Locating prefix hijackers using LOCK," in *Proceedings of the USENIX Security Symposium*, 2009.

[95] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (S-BGP)," vol. 18, no. 4, pp. 582–592, 2000. [Online]. Available: citeseer.ist.psu.edu/kent00secure.html

[96] W. Aiello, J. Ioannidis, and P. McDaniel, "Origin authentication in interdomain routing." New York, NY, USA: ACM, 2003, pp. 165–178.

[97] K. Butler, P. McDaniel, and W. Aiello, "Optimizing BGP security by exploiting path stability." New York, NY, USA: ACM, 2006, pp. 298–310.

[98] S. Qiu, F. Monrose, A. Terzis, and P. McDaniel, "Efficient techniques for detecting false origin advertisements in inter-domain routing," in *Proceedings of the Workshop on Secure Network Protocols*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 12–19.

[99] P. McDaniel, W. Aiello, K. Butler, and J. Ioannidis, "Origin authentication in interdomain routing," vol. 50, no. 16, pp. 2953–2980, 2006.

[100] P. van Oorschot, T. Wan, and E. Kranakis, "On interdomain routing security and pretty secure BGP (psBGP)," *ACM Transactions on Information and System Security*, vol. 10, no. 3, p. 11, 2007.

[101] J. Karlin, S. Forrest, and J. Rexford, "Pretty good bgp: Improving bgp by cautiously adopting routes," in *ICNP*, 2006, pp. 290–299.

[102] ——, "Autonomous security for autonomous systems," *Computer Networks*, vol. 52, no. 15, pp. 2908–2923, 2008.

[103] Internet Alert Registry, "Internet alert registry." [Online]. Available: http://www.cs.unm.edu/~karlinjf/IAR/

[104] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. Wu, and L. Zhang, "Detection of invalid routing announcement in the Internet," in *Proceedings of International Conference on Dependable Systems and Networks (DSN'02)*, 2002. [Online]. Available: citeseer.nj.nec.com/596080.html

[105] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Topology-based detection of anomalous BGP messages," in *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2003.

[106] RIPE NCC, "RIPE MyASN service," http://ris.ripe.net/myasn.html.

[107] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang, "PHAS: a prefix hijack alert system." Berkeley, CA, USA: USENIX Association, 2006, pp. 11–11.

[108] G. Siganos and M. Faloutsos, "Neighborhood watch for internet routing: Can we improve the robustness of internet routing today?" in *INFOCOM*, 2007, pp. 1271–1279.

[109] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A light-weight distributed scheme for detecting IP prefix hijacks in real-time," 2007, pp. 277–288. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282412

[110] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "ispy: Detecting IP prefix hijacking on my own." New York, NY, USA: ACM, 2008, pp. 327–338.

[111] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz, "Listen and whisper: Security mechanisms for bgp (awarded best student paper!)," in *NSDI*, 2004, pp. 127–140.

[112] X. Hu and Z. M. Mao, "Accurate real-time identification of IP prefix hijacking," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2007, pp. 3–17.

[113] K. Harrenstien and V. White, "NICNAME/WHOIS," RFC 812, IETF, 1982, obsoleted by RFCs 954, 3912. [Online]. Available: http://www.ietf.org/rfc/rfc812.txt

[114] K. Harrenstien, M. Stahl, and E. Feinler, "NICNAME/WHOIS," RFC 954 (Draft Standard), IETF, 1985, obsoleted by RFC 3912. [Online]. Available: http://www.ietf.org/rfc/rfc954.txt

[115] L. Daigle, "WHOIS Protocol Specification," RFC 3912 (Draft Standard), IETF, 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3912.txt

[116] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An information plane for distributed services," in *Proceedings of the Symposium on Operating System Design and Implementation (OSDI)*, November 2006.

[117] A. Broido and k. claffy, "Analysis of RouteViews BGP data: policy atoms," in *Network Resource Data Management Workshop*, 2001.

[118] H. Yan, R. Oliveira, K. Burnett, D. Matthews, L. Zhang, and D. Massey, "BGPmon: A real-time, scalable, extensible monitoring system," in *Proceedings of Cybersecurity Applications and Technologies Conference for Homeland Security*. IEEE Computer Society, 2009.

[119] T. Griffin and G. Wilfong, "An analysis of BGP convergence properties," August 1999. [Online]. Available: citeseer.nj.nec.com/griffin99analysis.html

[120] CAIDA, "IP prefix-to-AS mapping comparison," 2010. [Online]. Available: http://www.caida.org/research/routing/prefix_as_comparison/

[121] R. Cohen and D. Raz, "The internet dark matter - on the missing links in the as connectivity map," in *INFOCOM*, 2006.

[122] Y. He, G. Siganos, M. Faloutsos, and S. Krishnamurthy, "A systematic framework for unearthing the missing links: measurements and impact." Berkeley, CA, USA: USENIX Association, 2007, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1973430.1973444

[123] ——, "Lord of the links: a framework for discovering missing links in the internet topology," vol. 17, no. 2, pp. 391–404, April 2009. [Online]. Available: http://dx.doi.org/10.1109/TNET.2008.926512

[124] M. Roughan, S. J. Tuke, and O. Maennel, "Bigfoot, sasquatch, the yeti and other missing links: what we don't know about the as graph," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '08. New York, NY, USA: ACM, 2008, pp. 325–330. [Online]. Available: http://doi.acm.org/10.1145/1452520.1452558

[125] K. Butler, T. Farley, P. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions," vol. 98, no. 1, pp. 100–122, January 2010.

[126] S. Goldberg, M. Schapira, P. Hummon, and J. Rexford, "How secure are secure interdomain routing protocols," ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 87–98. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851195

[127] S. M. Bellovin, "Using the domain name system for system break-ins," in *Proceedings of the Fifth Usenix Unix Security Symposium*, Salt Lake City, UT, June 1995, pp. 199–208. [Online]. Available: https://www.cs.columbia.edu/~smb/papers/dnshack.pdf

[128] D. Eastlake 3rd and C. Kaufman, "Domain Name System Security Extensions," RFC 2065, IETF, 1997, obsoleted by RFC 2535. [Online]. Available: http://www.ietf.org/rfc/rfc2065.txt

[129] D. Eastlake 3rd, "Domain Name System Security Extensions," RFC 2535, IETF, 1999, obsoleted by RFCs 4033, 4034, 4035, updated by RFCs 2931, 3007, 3008, 3090, 3226, 3445, 3597, 3655, 3658, 3755, 3757, 3845. [Online]. Available: http://www.ietf.org/rfc/rfc2535.txt

[130] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, “DNS Security Introduction and Requirements,” RFC 4033, IETF, 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4033.txt

[131] ——, “Resource Records for the DNS Security Extensions,” RFC 4034, IETF, 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4034.txt

[132] ——, “Protocol Modifications for the DNS Security Extensions,” RFC 4035, IETF, 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4035.txt

[133] Executive Office Of The President, Office Of Management And Budget, “Securing the federal governments domain name system infrastructure,” August 2008, http://www.whitehouse.gov/sites/default/files/omb/memoranda/fy2008/m08-23.pdf.

[134] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, March 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[135] Open Networking Foundation, “Software-defined networking: The new norm for networks,” April 2012, https://www.opennetworking.org/images/stories/downloads/openflow/wp-sdn-newnorm.pdf. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/openflow/wp-sdn-newnorm.pdf

[136] Y. Zhang, Z. Zhang, Z. Mao, C. Hu, and B. MacDowell Maggs, “On the impact of route monitor selection,” in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement.* ACM, 2007, pp. 215–220.

[137] J. Ioannidis and S. M. Bellovin, “Implementing pushback: Router-based defense against DDoS attacks,” in *Proceedings of the Network and Distributed System Security Symposium*, 2002.