IPv6 Address Cookies

Mitigating Spoofed Attacks in the Next Generation Internet

A Thesis Presented by

Timothy D. Morgan

to

the College of Computer and Information Science

In partial fulfillment of the requirements for the degree of

Master of Science

in the field of

Computer Science

Northeastern University

Boston, Massachusetts

May, 2006 CE

IPv6 Address Cookies

MITIGATING SPOOFED ATTACKS IN THE NEXT GENERATION INTERNET

by

Timothy D. Morgan

tmorgan @ ccs.neu.edu

Dedication

This work is dedicated to my loving wife, Joan. Her persistent support throughout this work will be difficult to repay.

Acknowledgements

I am honored to have received help from so many people in this research endeavor. My thesis advisor, Dr. Guevara Noubir, has been a great motivator and an inspirational mentor. I thank my parents for their unwavering encouragement in all things educational.

I would also like to thank the following people for helping me with the research, experimentation, and writing that went into this paper (in alphabetical order): Dr. Javed Aslam, David Blank-Edelman, Bill Finizia, Jason Morgan, Dr. Riccardo Pucella, and Dr. Ravi Sundaram.

Abstract

It has long been known to researchers that address spoofing on the Internet is a serious problem. While a great deal of effort has been put into finding theoretical and practical solutions, spoofed attacks are still globally endemic. They represent a simple nuisance to many, but a business-halting bane to others. Enter IPv6. IPv6 is the next generation of the Internet protocol designed to alleviate the existing global address shortage and improve the scalability and extensibility of the aging IPv4 protocol. This new protocol provides an enormous 128-bit address space, which should provide enough addresses for several decades, if not centuries, of Internet expansion. In this paper, we propose methods which utilize the large IPv6 address space to mitigate spoofed attacks.

Contents

1 Introduction

It has long been known to researchers that address spoofing on the Internet is a serious problem. While a great deal of effort has been put into finding theoretical and practical solutions, spoofed attacks are still globally endemic. They represent a simple nuisance to many, but a business-halting bane to others.

Enter IPv6. IPv6 is the next generation of the Internet protocol designed to alleviate the existing global address shortage and improve the scalability and extensibility of the aging IPv4 protocol. This new protocol provides a huge 128-bit address space which should provide enough addresses for a great deal of Internet expansion. Individual Internet users can easily obtain their own 80-bit block of addresses if they currently have a single IPv4 address. This means it will be possible for any user to effectively hide a system in such an address space without ever being found. That is, if no DNS records point to them, and they don't respond to broadcast requests, it would be effectively impossible to find a host through brute force probes.

To put this into perspective, let us suppose an attacker can scan addresses of a network at 2^{32} (more than four billion) packets/second (which is probably a stretch for today's fastest routers [45]). It would take on average, 2^{47} seconds (or 4.5 million years) to find a single live address in a range of 2^{80} addresses. For this reason, the only way to find systems that do not pick an obvious IPv6 address would be to query some name resolution service (such as the DNS). In this paper, we propose methods which utilize the large IPv6 address space to mitigate spoofed attacks by forcing clients to always use the DNS prior to submitting service requests.

The remainder of this chapter contains the problem statement and some definitions of commonly used terms. Chapter 2 describes and classifies the popular types of attacks used on the current IPv4 Internet. Chapter 3 explores the impact IPv6 will have on scanning and spoofing attacks. Chapter 4 gives an overview of the novel system proposed for combating spoofed attacks in IPv6. Chapter 5 provides analysis of attacks against the proposed system, as well as how it can be used to mitigate current attacks. Chapter 6 proposes some solutions for the engineering challenges faced with implementing the system and describes a prototype software implementation used in testing. Chapter 7 describes experiments performed with the prototype software implementation, and provides results of these tests. Finally, chapter 8 offers some concluding remarks.

1.1 Problem Statement

Spoofed denial of service attacks have plagued the Internet for a number of years, and show no signs of abating. Research into mitigation techniques has apparently not led to a financially viable solution, and new attacks have been discovered in the wild without being widely anticipated. With the advent of a new Internet protocol, the world stands in a unique position to get ahead of this criminal activity by fundamentally changing the way the Internet works and being prepared for the attacks to come. Whether by intention or chance, the vastly increased Internet address space in IPv6 has done so. We seek to use this fundamental shift in resource availability to develop a novel, lower cost solution to mitigating spoofed attacks.

1.2 Definitions

Network and System Security

Botnet Slang for robot network. Also referred to as a network of zombies. These are a set of compromised machines, controlled illegally by an attacker, and are typically used for conducting denial of service attacks and storage of contraband [36].

DoS Denial of service. An attack that typically involves the flooding of a victim computer with large numbers of requests. The intent is to deny service to legitimate users of the system.

DDoS Distributed DoS. An attack using many different attacking systems, which are almost always members of a botnet.

Zero-day A vulnerability or exploit that has not yet been announced to the public or software vendors. Thus, it has been public for 0 days. Development of such exploits allows malicious attackers to compromise system security before any patch or workaround is available [83].

Zombie A compromised computer system often used in spoofed attacks and other illegal activity.

Cryptography

CFB Ciphertext feedback. A mode of encryption that allows a block cipher to act as a stream cipher. In this mode, ciphertext is fed back into the block cipher's input, thereby allowing the plaintext to influence the output of future blocks [66].

Confusion An essential property of cryptographic primitives which is often implemented through the use of a substitution operation (called the S-box). The purpose of confusion is to make the statistical relationship between the plaintext, key, and ciphertext a very complex one [67].

Diffusion A second important property of cryptographic primitives which deals with how evenly specific output values depend on every input value. Diffusion serves to distribute the information of inputs throughout the ciphertext, thereby reducing the statistical significance of specific features of a plaintext or key. (See [67] for more information.)

HMAC A hashed message authentication code. An adaptation of a secure one-way hashing function which uses a secret shared key. HMACs may be used to provide integrity and authentication for messages of variable length. (See [43] for more information.)

IPsec Internet protocol security. A layer three protocol designed by the IETF for authenticating and encrypting communications on the Internet.

OFB Output feedback. A mode of encryption that allows a block cipher to act as a stream cipher. In this mode, the output of the block cipher is fed back into the block cipher's input prior to being combined with the plaintext. This means the plaintext encrypted in one round of this mode will not influence future rounds of encryption [66].

OTP One-time password. A form of authentication used to thwart passive eavesdropping of credentials. Passwords are used only once and each new password is typically generated via some synchronized random number generator or series of hashes at both a client and server.

PKI Public Key Infrastructure. An authentication framework generally designed to

authenticate public keys by associating them with aspects of a user's identity. Such associations are often called certificates.

Miscellaneous

6to4 A transitional tunneling technology allowing IPv6 systems to automatically transmit packets to other IPv6-enabled systems over the IPv4 Internet. (For more information, see RFC 3056 [14].)

AAAA The DNS record type which maps domain names to IPv6 addresses. (See RFC 3596 for more information [77].)

Anycast A form of routing where the "closest" or "best" member of a group is sent a packet.

Flow label A field in a packet used to differentiate several flows of traffic between two hosts. IPv6 flow labels are defined in RFC 3697 [60].

IETF The Internet Engineering Task Force is a part of the Internet Society which develops technical Internet standards and best practices.

LRU Least recently used. An algorithm for removing an object from a cache based on when objects were last used.

2 Survey and Discussion of Current Spoofed Attacks

In this section, we will first list and describe a number of specific spoofed attacks and follow with a general method of classification for DoS attacks. While extensive, the list below cannot be considered exhaustive.

2.1 Current and Popular Attacks

2.1.1 Spoofed Scans

Idle Scan

While the most commonly seen spoofed attacks are designed for resource-consumption, there are a few that can be used to gather information about target systems. Perhaps one of the most

effective of these is the *idle scan*. An idle scan is a spoofed port scan which uses an unwitting third party host as a reflector. Many IP implementations use a single sequential counter for the ID field of all outgoing packets. A global counter such as this leaks information to potential attackers about the number of packets being sent by a host, and this can be used to scan other systems.

The specific idle scan attack (described in [3, 30]) requires an attacker to first find a system running a vulnerable IP implementation which currently utilizes few network resources (hence the term "idle"). The attacker then sends a TCP SYN packet to the victim he wishes to scan, with a source address matching that of the idle host. If the victim's port is open, it will respond with a SYN/ACK packet. When the idle host receives a SYN/ACK for a connection it did not start, it will respond with a TCP RST. This will use up one of the IP IDs in its global sequence. On the other hand, if the victim's port is closed, it will respond with a RST evoking no response from the idle host. If an attacker can detect the difference between the two through the idle host, then he can obtain the results of the scan without ever giving away his real IP address. This detection can be accomplished by continuously pinging the idle host while the scan is taking place. Whenever a "hole" is found in the IP ID sequence, one can assert with high probability that the most recent port scanned is open.

IP Source Routing

Another classic technique used in spoofing attacks is the utilization of IP source routing headers (see [57]). By using source routing options, an attacker can bypass naïve firewall rules and carry on spoofed conversations with systems that reverse the source routing options [48]. While quite powerful, these spoofing attacks have the drawback that the attacker's IP address is ultimately sent to the victim, though not in the normal source field. In addition, this vulnerability in the IP protocol has been known for some time, and most firewalls strip or drop packets which contain source routing options. Many operating systems are also configured by default to ignore these options.

2.1.2 Denial of Service Attacks

While spoofed attacks not designed for DoS are technically interesting, they are not all that popular on today's Internet. The majority of spoofed attacks are designed to exhaust a victim's resources, and come in a variety of flavors. For surveys and case studies of the relative

popularity of the following attacks, see [4, 37, 52, 53].

Reflected Attacks

Perhaps one of the more classic of DoS attacks the *smurf attack* [25]. This is one form of an amplification attack, which can be very effective at saturating a victim's connection. This attack utilizes a third party network, which contains hosts that respond to broadcast ping (ICMP echo_request) packets. An attacker spoofs a ping packet to such a broadcast address, with the source address of the victim. When the third party network responds, it will likely produce many more packets than the attacker originally sent, which can amplify the attack by a significant factor. Fortunately, over the last several years, network administrators have become more aware of the dangers of allowing broadcast packets into their networks from the Internet at large. Consequently, finding amplification reflectors has become more difficult (though not impossible) for attackers using smurf.

Similar to the smurf attack is *fraggle*. Instead of using ICMP echo_request packets to induce a response from a reflector network, the fraggle attack uses UDP packets directed at the echo service [58]. While fraggle attacks are as effective as smurf attacks, given a network supporting broadcast and UDP echo, they are also easier for victims to block since fewer people rely on UDP echo for network testing.

Many other protocols are susceptible to reflection attacks. Most do not provide an attacker with quite as much control over response packet sizes or have as large of an amplification effect, but they can still be quite devastating. Some examples of these include TCP handshake reflection and DNS reflection. Many of these are described in detail in a recent Internet draft [34] and in [2, 8], and at least one case study of a TCP handshake reflection attack has been published [31].

While using a reflection attack can be a very powerful way to exhaust a victim's CPU or bandwidth resources, they are also typically very easy to filter if done far enough upstream of the victim. This is because the final packets reaching the victim are generally a response to packets never previously sent (e.g., ping replies where no ping was sent) and they come from a specific array of hosts (the reflectors) which are not spoofing their own addresses. By working with service providers, a defender can generally mitigate such attacks pretty quickly.

Direct Floods

Another form of spoofed attack, which has proven to be very effective against TCP, is the *SYN flood* [23]. This simple attack involves flooding a victim with a large number of spoofed SYN

packets. Victim systems will reply with a SYN/ACK as the second step of the TCP handshake, subsequently storing a record of the transaction for later verification. Normally, clients will finish the handshake with an ACK packet which will establish the connection, but with a spoofed SYN flood, no ACK will ever come. If an attacker sends SYN packets fast enough, then the victim will run out of memory resources before it can time out the spoofed connection requests. Eventually, victim servers will be forced to drop some handshake records, some of which will be those of valid clients. This attack is particularly effective because it is usually impossible to tell the difference between malicious and non-malicious SYN packets when they are initially received.

SYN floods were used with great effectiveness in early February of 2000 to shut down several large Internet companies [71]. Since then such attacks have become less effective, as mitigations such as SYN cookies [12] have been developed and implemented in several operating systems. SYN cookies rely on a cryptographic checksum of connection parameters (which is embedded in SYN/ACK replies) to authenticate connections when the third step of the handshake occurs. This allows servers to forego storing any information about half-open connections, which greatly increases the rate at which SYNs can be handled with limited memory. Unfortunately, not all TCP implementations use SYN cookies, so SYN floods are still effective against several widely-used operating systems[1].

IP fragmentation attacks are another class of direct floods that exhaust memory resources. The Internet protocol provides features for packet fragmentation which are used when transmitting data from one medium to another. If, for instance, one were to send a 9000-byte data frame from one network to a system located on another network with a maximum frame size of 1500 bytes, the packet (typically) would be fragmented into at least 6 separate IP packets. Endpoints (and possibly routers in between) will need to reassemble the fragments prior to using any of the upper-layer information contained in the packet.

There are many forms of fragmentation attacks, some of which are meant for denial of service, and others for the evasion of security controls. An early form of this attack, named *teardrop* [24], used overlapping packets to confuse victims' IP implementations. Some of these implementations would fail catastrophically, causing systems to crash. Today, teardrop and its variants are not very effective in denial of service, as such implementation bugs have mostly been eliminated.

[1] As of this writing, we could not find documentation indicating that Microsoft ® or Apple ® operating systems use SYN cookies.

More recently, the *rose attack* [35] was developed with the goal of memory and CPU resource exhaustion. In its simplest form, two packet fragments are sent to a victim: one reporting itself as the the first fragment from the set, the other claiming to be the last fragment. After that, no more fragments are sent which claim to be a part of that original packet. By sending large numbers of these fragment pairs, some IP fragment reassembly algorithms will allocate large amounts of memory (e.g., all of the memory needed to accommodate fragments not yet received) or CPU resources.

Against a well designed and implemented reassembly algorithm, such attacks should not be a big issue because the majority of traffic on the Internet is no longer fragmented. Thus, legitimate traffic can pass through relatively unhindered by the dropped fragments that would result in such a DoS. However, some protocols such as DNS, may need to use fragments for large result sets, and could be affected. Finally, security devices often have a lot of trouble with fragmentation. Firewalls may need to reassemble fragments before validating proper port numbers and network intrusion detection systems generally need to reassemble packets prior to analysis, since attackers sometimes use fragmentation to hide their tracks [59].

2.2 Attack Classification

There is much discussion in the related literature on the topic of denial of service attack classification [2, 28, 37, 49]. For the purposes of this paper, we focus on two aspects of DoS attacks: the primary resource consumed, and the potential points of failure in the path of the attack. In order to more easily break down the classification, we first introduce some general attack classes:

direct: Attacks considered direct are those where a victim is flooded with single-packet spoofed requests directly from the attacking systems. A simple ping or DNS request flood sent from the attacking systems directly to the target are examples of this kind of attack.

direct stateful: This type of spoofed flood is a special subset of direct attacks. It preys on protocols where the victim expects more packets later which relate to those previously received. Because such protocols require victims to store information between packets, it is sometimes possible to exhaust a victim's memory resources. The classic example is the SYN flood, where victims later expect an ACK to their SYN/ACK responses. IP fragmentation attacks also fall into this category, as well as misconfigured DNS servers

which allow recursive lookups. Any protocol that requires end points or routers/firewalls to store state for each request could potentially be attacked in this way.

reflected: Reflected attacks are those where an attacker does not send packets directly to the victim, but sends packets to third party systems (reflectors) which reply with packets directed at the victim. This can serve to amplify the number or sizes of packets, or to mask the original source of an attack.

reflected echo: Reflected echo attacks are a special case of reflected attack in which an attacker has control over the content (or at least the amount) of data contained within the response packets being generated by third parties. For instance, in smurf and fraggle attacks, reply packets sent by reflectors contain the same data as sent by the attacker. This is important, since attackers have full control over the amount of content being sent and, therefore, the amount of amplification. DNS query/response reflection can have a similar effect, if an attacker first goes through the trouble of finding DNS servers which reply with large records.

The preceding definitions should encompass nearly every type of resource-consumption-based denial of service attack currently known. The following table contains a breakdown on how each group could be used to target different resources and systems:

TARGET	END POINT	FIREWALL	ROUTERS
MEMORY	direct stateful	direct stateful	
CPU	direct reflected	direct reflected	direct reflected
BANDWIDTH		reflected echo	reflected echo

Table 1: SPOOFED DOS ATTACK CLASSIFICATION

In the table above, “End Point” refers to the destination address of the floods; “Firewall” refers to one or more packet filtering devices, which store state about connections for the purposes of filtering; and “Routers” refers to all routers between the End Point and the attacker’s systems, which do not store state about connections. It is generally assumed that End Points have a much faster link between them and their gateway than the gateway does to the Internet. Therefore it should not be considered a choke-point for communications to the outside world. Also note that, while it is possible for direct and reflected attacks to consume all available bandwidth on a link due to the sheer amount of data being moved, it is assumed that packet

rate bottlenecks will arise (e.g., CPU limitations) on some router/firewall before this occurs. (This is probably not a good assumption, but it does not significantly affect the final analysis.)

2.3 Current Defense Strategies

Ingress/Egress Filtering

Perhaps the most straight-forward strategy for mitigating spoofing attacks is to filter packets at their source. If all ISPs and businesses dropped any outbound packets which advertised a source address they do not own, many spoofing attacks would be made much more difficult. In particular, massively distributed DoS attacks using virus-infected zombies would not be as feasible. The IETF has made recommendations on how to do this [29]. Unfortunately, many (perhaps most) Internet service providers have failed to do so. This lack of vigilance can primarily be attributed to the fact that such filtering increases load on routers and would require providers to buy additional equipment. In addition, it is likely difficult for ISPs to see such filtering as an immediate benefit to their business, since they would still be susceptible to spoofed attacks if other ISPs did not implement the same filtering.

Upstream Mitigation

As a victim of a DoS attack, often the only option to stop the attack is to have the upstream ISP filter the attack temporarily. Because many attacks are easy to spot, based on the type of packet or source addresses (in the case of reflected attacks), ISPs can generally stop most attack traffic directed at a victim. Unfortunately this is typically a very manual, labor-intensive task. Also, such filters can rarely be left in place after the attack has stopped, because some valid traffic may be blocked by them.

Defending against direct stateful attacks poses a particularly difficult problem for networks trying to filter the flood, because these attacks are often more difficult to distinguish from normal requests. Generally, when buffers storing state for a given protocol (i.e., SYN buffers, fragment buffers) are exhausted, the best strategy is to randomly drop entries instead of the more commonly used LRU policy. This is because attackers who can flood a victim at a specific rate can always push out any valid stateful requests before they are serviced when an LRU strategy is employed. Unfortunately, even random drop policies do not provide much protection against a powerful attacker. Some protocols (such as TCP) are conducive to the addition of cryptographic cookies which can effectively mitigate state-based attacks, because defending

systems can avoid storing any information about initial handshake packets. On the other hand, for many protocols which are vulnerable to state-based attacks, protections like cookies cannot be shoe-horned into the protocol after the fact. Additionally, while the use of cookies can prevent state-based attacks, CPU resources can still be targeted.

Traceback

While mitigating DoS attacks may yield an immediate fix to the side-effect of this problem, it does not address the core of it: the attacker who is attempting to disrupt communications and deny access to the victim. If attackers were presented with a strong deterrent, a reasonably high probability of being caught and prosecuted, then there would likely be many fewer such attacks. One way of catching a perpetrator starts by tracing the attack back to the originating attack system(s). The traceback problem has been discussed a great deal in the literature, and a large number of solutions have been proposed [46, 62, 68, 70, 74]. Unfortunately, many of these systems either require changes to the Internet protocol, are specific to upper-layer protocols, or only provide partial traceback information. A good summary of several of these techniques (as well as other defensive strategies) can be found in [2].

Client Puzzles

An approach for attack mitigation which is closely related to cryptographic cookies is the use of client puzzles [7, 10, 26, 39]. The general technique is to instrument protocols with simple mathematical problems, which clients must solve before being given access to a system's resource. In such a protocol, clients who are behaving badly are given incrementally more difficult problems to solve for each request, which has the effect of significantly slowing attacks. This approach is attractive, because it can be designed to require very low overhead on the part of the server (so long as generation and verification operations are fast) and degrades gracefully even in the face of occasional false positives. On the down side, client puzzles invariably require significant changes to existing protocols (if not new protocols all together) and this approach is not any better suited to dealing with spoofed DoS attacks than cookies, since attackers do not intend on carrying on the conversation to complete the puzzle anyway.

Overlay Networks

There have been many proposals to use overlay networks for the mitigation and traceback of DoS attacks [41, 55, 70, 74, 85]. This approach to the problem is almost certainly to provide convenient backward compatibility with old protocols while newer, safer ones are in use. Sev-

eral appear to be effective solutions, but almost always require some critical mass of routers to participate in the new protocol. Simpler mitigations to several specific DoS attacks have already been proposed and implemented in software, but convincing administrators of the need to deploy them has held up the process. It is difficult to see how a more complicated solution, like an overlay network protocol, could be deployed in the short term unless some other major benefit accompanied the protocol.

3 Next Generation Internet and the New DoS Landscape

3.1 IPv6 Overview

IPv6, formerly called the IPng, is designed to be an incremental upgrade to the current Internet protocol, IPv4. The chief motivating factor of moving to another protocol is the artificial restriction on IPv4 address space. While network address translation technologies have staved off an address shortage disaster temporarily, they often create difficulties with certain upper-layer protocols and will not continue to operate efficiently forever. IPv6's 128-bit address space is much larger than IPv4's meager 32-bit space, and it is difficult to envision a world in which 2^{128} addresses would not be sufficient for every humans' computers.

In addition to the larger address space, IPv6 sports a simpler header for faster routing and more easily integrated IPsec along with new features such as anycast addressing, interface auto-configuration and flow labeling [33]. Many of these features exist in IPv4, but are notoriously difficult to use without configuring all systems on a network specifically to use them. IPv6's flexible header format allows many extensions to be used while also being more portable.

Besides having a much larger address space, many address-wasting conventions of IPv4 have been done away with in IPv6. Network base addresses are not reserved for any special purpose and by default, networks do not need to be configured with a broadcast address. The reserved localhost addresses no longer eat an entire block of 2^{24} addresses. NAT/NAPT are no longer needed which means RFC1918-style private ranges are not either. Finally, with the ample address space available, the organizations in charge of distributing address ranges will do so in a uniform and well-planned manner designed to minimize the size of global routing tables [38].

3.2 The Impact on Routing and Spoofing

With the advent of IPv6's simpler address allocations/routing tables and the option of integrated IPsec, there is a real opportunity to limit the impact of spoofing attacks. Simpler routing tables means it may be easier for network administrators to stop obvious spoofing attacks coming from their networks. Additionally, if end-users deploy IPsec authentication headers, packet source addresses could be validated. Unfortunately, it is unlikely that the public key infrastructure will be set up any time soon to accommodate authentication of arbitrary clients.

As for egress/ingress filtering, there are limitations with it as well. Transition technologies, such as 6to4 auto-tunnelling, will be used to migrate from IPv4 to IPv6 [14]. These mechanisms will be in place for a significant period of time, at least until the majority of the Internet fully supports the new Internet protocol. While they are still being used, attackers can easily spoof packets without the fear of filtering so long as they can route IPv4 packets and their ISP does not employ IPv4 ingress filtering [63]. Therefore, it seems switching to IPv6 will not create any significant difficulties for attackers trying to conduct spoofed attacks in the near term. Other mitigations to spoofed attacks will still be relevant and a necessity for many users.

3.3 Network Scanning a Massive Address Space

One interesting side effect of the deployment of IPv6 will be the potentially increased difficulty of discovery scanning [15]. With each end-user network receiving many orders of magnitude more addresses than the entire public Internet uses now, scanning for live hosts could become a much more difficult task. Some have commented that this larger address space will make high-speed network worms less commonplace, though the debate on that topic is far from reaching a conclusion [11, 40]. In any case, it is likely that the DNS or other naming services will be used much more frequently even within smaller networks, since addresses will be harder to remember. Also, network inventory and auditing software will likely turn to passive traffic analysis for the purpose of host discovery. These observations are what initially inspired the work presented in this paper. Surely there must be a way that administrators can use this large address space to defend their networks.

4 Overview of Address Cookies

4.1 Inspiration and Approach

The use of random or cryptographic cookies has been around for some time, typically for the purpose of tracking how information is used [69, 78]. In the domain of networking, several protocols have been proposed in which cookies are used to defend against denial of service attacks [1, 73]. More recently, systems have been developed which use email address cookies to track spammers [79]. The original inspiration for IPv6 address cookies came directly from the success of SYN cookies. SYN cookies are effective because they require clients to provide the cryptographic cookie back at a later time in order to complete a TCP connection. The only way a client could obtain the cookie would be to provide some source address belonging to that client in the original SYN.

This is the same premise being used in address cookies. When clients request an IP address for a domain, they are given a cryptographic cookie as the address. If they spoof this initial request, they will be denied any further service since they will not have access to the cookie. The main advantages over SYN cookies is that address cookies work at a lower layer (and thus are not limited to TCP alone) and the size of an address cookie can be much larger, allowing for more complex features and additional security. The disadvantage of using address cookies with the DNS, is that the system requesting the record is not commonly the same as the one requesting services later. Therefore, (relatively) strong authentication of clients between these two protocols is not possible as it is between the SYN and the ACK of a TCP handshake.

4.2 Previous Work

Some related work on cryptographically generated IPv6 addresses appeared in [56] and [51], in the area of mobile IPv6. Generalized schemes for this were later standardized in CGA [6], which is used by SEND [5] to authenticate ICMPv6 neighbor discovery messages. In general, these are methods for generating an IPv6 address through the hash of a public key, thereby tying the key to the address of the system. This approach to authenticating addresses is a great idea, and would be very useful in private networks, but it does not seem likely that a universally trusted PKI will be in place anytime soon to eliminate spoofing attacks over the Internet at large.

TARP is another system, proposed by Gleitz and Bellovin, which proposes an interesting use for the large IPv6 address space [32]. In this system, additional addresses are allocated dynamically by clients in order to alleviate the difficulties associated with allowing complex, multi-port protocols through a stateful firewall. Such a system may be useful in conjunction with address cookies to reduce the storage requirements on the firewall.

Work in the IPv4 space has also yielded similar systems. For instance, DAM [16] is an architecture for DoS mitigation which is very similar to that proposed in this paper, but requires much more storage and communication overhead between components. Sundaram and Milliken have applied for a US patent [75] covering the use of randomized, resolver-specific responses to name resolution queries for the purpose of tracking and blocking attackers. Unfortunately, due to the limited IPv4 address space being used, attackers would be able to brute force the generated IP addresses in relatively little time through direct or spoofed idle scans.

The system proposed in this paper obviously shares characteristics with many of these systems, but none seem to have gone far enough down this path of research to fully apply the idea of cryptographically generated IPv6 addresses to spoofed attack mitigation.

4.3 Protocol Goals

It is hypothesized that a system using address cookies can make the following four guarantees about client requests:

1. *Unpredictability*

   Clients cannot easily guess valid address cookies (destination addresses) even with knowledge of several past valid cookies.

2. *Expiration*

   Valid address cookies remain valid for new requests only as long as the DNS TTL has not expired.

3. *Record Association*

   A defending firewall can *strongly*[2] associate an address cookie with the specific DNS record requested earlier.

[2]Here, we mean that the firewall will be able to determine the precise DNS record the resolver requested to obtain a cookie.

4. *Resolver Association*

A defending firewall can *weakly*[3] associate an address cookie with the DNS client's address.

4.4 Proposed Protocol and Architecture

The following protocol is designed to satisfy all of the above guarantee. Let us assume a defending network hosts a firewall, a DNS server, a web server[4], and (optionally) a logging server. In addition, the defending network's ISP manages an upstream router, which may also participate in the protocol. While these systems may be organized in several different ways, let us assume for now they are set up more or less as indicated in Figure 1.

In order to communicate with the web server, a client would need to follow these steps:

1. Client requests AAAA record from some third-party DNS recursive/caching server (DNS cache);
2. The DNS cache sends a request for AAAA record to the defending authoritative DNS server;
3. DNS server responds with a AAAA record containing an IPv6 address for the web server;
4. DNS cache responds to the client with AAAA record; and,
5. Client begins sending requests to the web server.

Of course a client could be playing the role of the cache server, but for generality, we assume they are different systems. Figure 2 helps to illustrate this interaction.

In order to require the client to first query DNS for the AAAA record, the DNS server and web server/firewall must agree on a randomization scheme for the addresses that will be considered valid and the firewall must validate these addresses upon receiving a request. One could simply have the DNS server pick a random address for each request, and communicate the IP addresses returned to the firewall, which will then set up rules to allow packets to those

[3]Since the entire IPv6 address of a resolver can't be fully embedded in an 80-bit cookie, some resolver-specific value will be embedded in the cookies for DNS client. Multiple resolvers may be assigned the same identifier value, making it a weak identifier.

[4]This system is applicable to any server which responds to requests from the public. The concrete example of a web server merely helps set the context and makes discussion simpler.

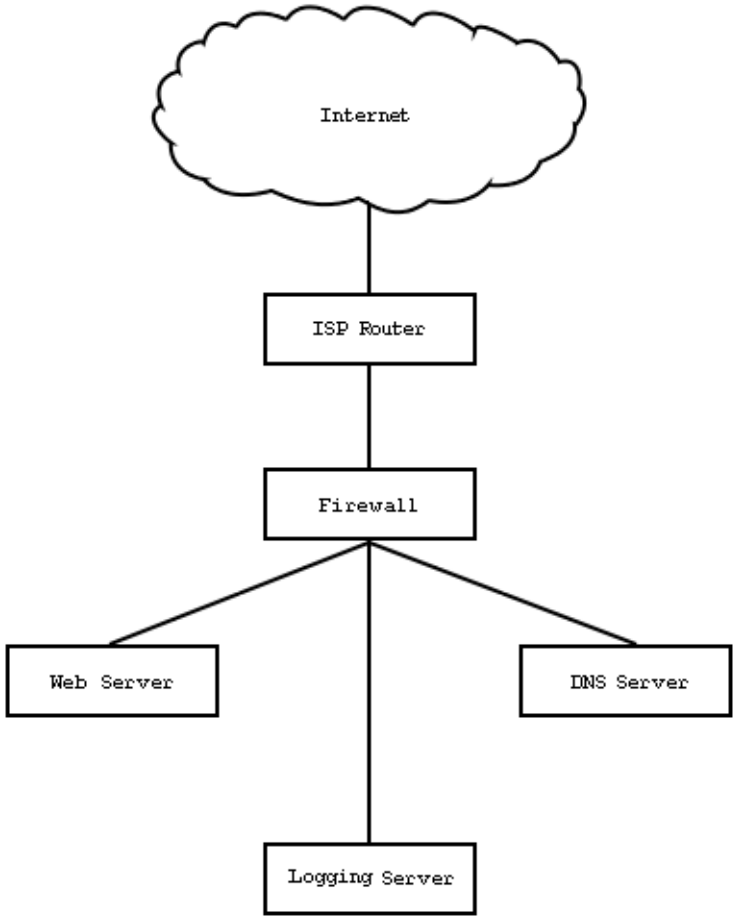

Figure 1: IPv6 ADDRESS COOKIES ARCHITECTURE

specific destinations. However, this would require a large amount of network overhead and storage. Instead, we propose the use of cryptographic cookies to randomize and authenticate these addresses. First, let us first define a few terms:

***Address Prefix (AP)*:**

The 48-bit address prefix for an 80-bit IPv6 range.

***Address Cookie (AC)*:**

A generated 80-bit value returned by the DNS server as part of an IPv6 address.

***Key (K)*:**

A secret key of at least 80 bits in length.

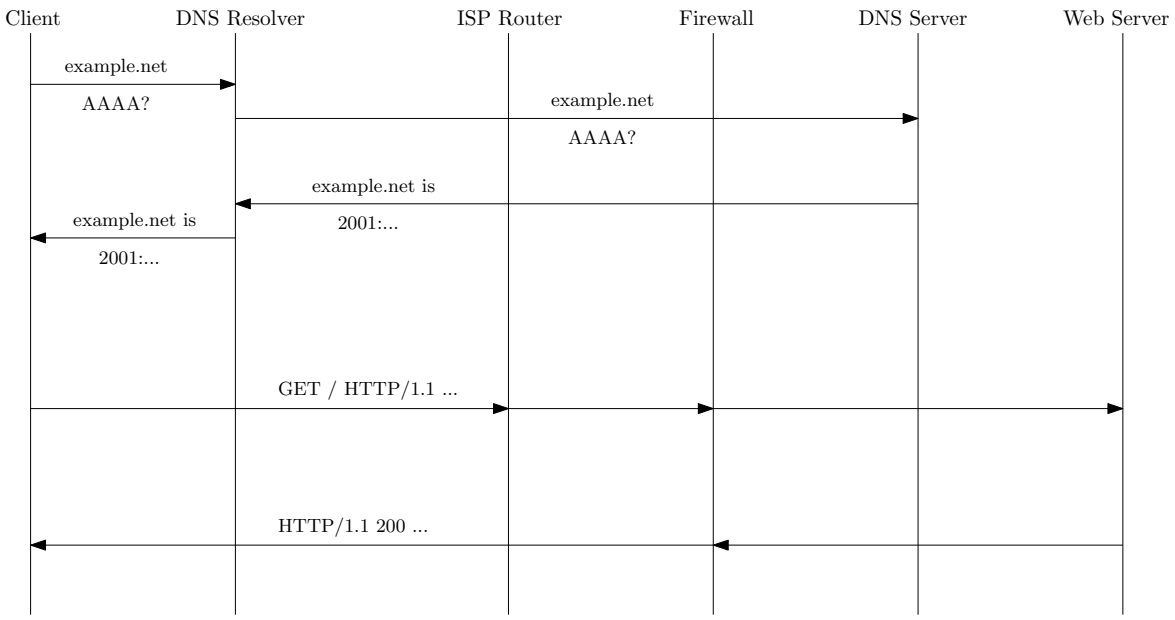

Figure 2: SAMPLE DNS QUERY AND HTTP REQUEST

***Time increment (T)*:**

Some fixed amount of time in seconds. (Probably in the range of 64 to 256).

***Time-To-Live (TTL)*:**

The configurable amount of time DNS records are set to live for. Must be a multiple of T.

***expiration*:**

A 24-bit timestamp representing the expiration time of the associated DNS record. This stamp is in the units of T, and is calculated by: $expiration = \lceil \frac{current\ time + TTL}{T} \rceil$

***Record ID (RID)*:**

An 8-bit identifier indicating which DNS record was requested.

***Source ID (SID)*:**

The 24-bit result of a secret hash function applied to the requesting DNS cache's IP address (or at least a portion of it). The secret key for this function need only be known to the defending DNS server.

***checksum*:**

A 24-bit cryptographic checksum of the *expiration*, *RID*, and *SID* values for a given cookie. In other words: $checksum = hash(expiration \mid RID \mid SID)$

The *AP*, *K*, *T*, and *TTL* values are shared between the DNS server, firewall, and upstream router. Note that, based on Figure 1, the DNS server may need to be accessible to the outside world via a non-randomized address, lying outside of *AP*, so resolvers can find it via referrals. Let the DNS server generate an 80-bit address cookie by:

$$AC = K\{expiration \mid RID \mid SID \mid checksum\}$$

Upon responding to a request with the address $(AP \mid AC)$, the DNS server need not store any information about the cookie in memory, nor does it need to communicate anything directly to the defending firewall about each lookup.

The purpose of the checksum is to make it difficult to guess random ciphertexts that happen to decrypt to valid cookies. This does not completely mitigate such an attack, but along with other sanity checks, sufficient security is provided. See section 5.1.4 for a detailed discussion of such attacks.

The purpose of the *SID* field is to assign an unpredictable identifier to requesting DNS caches. This is designed to satisfy resolver association (guarantee #4), and its utility is discussed in later sections.

The purpose of the *RID* field is to communicate to the firewall which DNS record the cookie was returned with. This allows the firewall administrator to define an *access control policy* based directly on the domain name that was requested. For instance, the DNS server, hosting several sub-domains, could have a domain mapping configured like so:

www.example.org ⇒ 0
mail.example.org ⇒ 1
ftp.example.org ⇒ 2
...

Up to 256 domains could be supported in this way. When a client subsequently connects to a service on the web server, it will send packets to the encrypted address cookie generated by the DNS server. The firewall, upon receiving these packets, will check a cookie using Algorithm 1. Once a packet is allowed, it will be passed on to the web server, possibly with the destination address rewritten.

Algorithm 1 FIREWALL COOKIE VALIDATION

```
If packet is acceptable part of established connection:
    ALLOW
Otherwise, decrypt the cookie and:
    If (expiration > current time)
        and (expiration < current time + TTL)
        and (RID contains defined value)
        and (firewall policy allows port/protocol for this RID)
        and (checksum is valid):
            Add connection to list of established connections
            ALLOW
    Otherwise:
        DENY
```

The determination of allowable ports and protocols is done by looking up the value of the *RID* field in a policy table stored on the firewall. Based on how domain names are mapped to the integers above, and administrator might want to map those numbers to the policies such as:

0 ⇒ Allowed: 80/TCP (HTTP), 443/TCP (HTTP/SSL)

1 ⇒ Allowed: 25/TCP (SMTP), 993/TCP (IMAP/SSL), 995/TCP (POP3/SSL)

2 ⇒ Allowed: 21/TCP (FTP)

...

This allows for a very flexible firewall policy which ties domain names directly to a set of rules while relying on a very small amount of information embedded in the cookie. This *RID* field could also dictate how packets are rewritten when NAT is in use.

The role of the ISP's upstream router is relatively simple. Its job (if it is participating) is primarily to validate cookie expirations, in order to take some load off of the defending firewall. See Algorithm 2 for the specific validation performed. While seemingly redundant, this validation is necessary to mitigate certain bandwidth and CPU exhaustion attacks. More on the utility of the ISP's router is described in later sections.

The logging server is an optional component and does not participate directly in the protocol. Instead, its purpose is to receive information from both the DNS server and the firewall allowing for offline analysis of attacks by administrators, as well as semi-realtime response to prolonged attacks. More information on the advantages of having a logging server can be found in section 5.2.

Algorithm 2 ROUTER COOKIE VALIDATION

Decrypt cookie
If (*expiration* > current time) and (*expiration* < current time + *TTL*):

    ALLOW

Otherwise:

    DENY

The proposed system meets the guarantees described in section 4.3. An in-depth analysis of how attacks against these guarantees are (or can be) thwarted can be found in section 5.1.

5 Analysis

5.1 Attacks Against Cookie Architecture

Here, we describe a list of possible attacks against each guarantee stated in section 4.3 for the proposed system.

5.1.1 Attacks on Guarantee #1, Unpredictability

Offline Cracking of Cookies

The most straight-forward attack against any encrypted cookie, would be to brute force crack the key. Once the key is discovered, new cookies could be generated offline, and used until the key were changed. In order to crack a key, there must be some method of verification involved. In this case, there is ample data in the plaintext of the cookie for an attacker to verify with high probability that a guessed key is a good candidate. For instance, the decrypted cookie must have a matching checksum, and the expiration must be reasonably close to the current time. Once a strong candidate key were discovered (which passes these tests on multiple cookies), one could generate new cookies with it and verify they work by sending requests to the firewall.

Defending Against Offline Cookie Cracking

Obviously, the best defense against brute force attacks is to create a large equiprobable key space. A very large key can be used with the encryption algorithms discussed in section 6.1.

In particular, Blowfish [65] permits enormous keys (up to 448 bits) without a performance penalty. Also, algorithms which use multiple rounds of block encryption, can use a different key with each round and frequent key rotation would also help mitigate offline attacks such as these, if necessary. These protections would easily thwart brute force offline cracking.

Online Guessing of Cookies

Online brute forcing of valid cookies may be possible if the set of all valid cookies with in the 80-bit cookie space is large enough and guesses can be sent at a high enough rate. To accomplish this, an attacker could merely generate random cookies and send them to the firewall. With some small probability, the random plaintext that the firewall obtains after decrypting will pass all necessary validation checks, and the request will be allowed.

An attacker will need some way to verify that a specific random cookie was valid. There are two main ways of accomplishing this. The obvious one is to send the request with a source IP address owned by the attacker. Once the request is answered, the attacker will know it was a valid cookie. This carries the disadvantage of giving away the attacker's address. However, there are other more covert ways of validating cookies. For instance, one could use an idle scan (see section 2.1 and [3, 30]) for verification, much like they are typically used to semi-anonymously scan for open ports. This specific attack is normally slower than sending/receiving packets directly, but may be sped up by using many idle hosts.

Defending Against Online Guessing

When validating cookies, the firewall checks several fields in the decrypted cookie. Obviously, the checksum is verified, but also, the *RID* field is checked as well as the expiration timestamp. The checksum for random cookies will be valid once in every 2^{24} attempts which, on its own, is not considered sufficient to defend against this attack.

Fortunately, randomly generated expiration (resulting from the randomly generated ciphertext) would not always be valid. It could have already expired, or the expiration could be set far enough into the future as to be considered invalid. Here, we present a concrete example in order to help explain the additional defense expiration timestamp checking provides against this attack. If an administrator decided to use a TTL of 65536 seconds (around 18.2 hours), and chose a value for T of 256 (or about $4\frac{1}{4}$minutes), then one could reject any cookies which have already expired, or expire after the TTL. This means the valid window is $\frac{65536}{256} = 256 = 2^8$ in size. The remaining bits in the expiration timestamp must match, and thus provide an ad-

ditional 16 bits worth of security against cookie guessing. Obviously choosing a larger *T*, or a smaller TTL will increase the security of such a mechanism up to at most 24 bits worth of security, but may negatively impact performance and granularity of TTLs.

An additional check is performed against the *RID* field in the proposed validation algorithm, which will typically provide some small amount of additional security. However, for the sake of argument, let us assume for now that our administrator is using all 256 record IDs, thus eliminating the added security of this check.

With around 40 bits worth of protection against random cookies, it is very unlikely that an attacker will find this kind of attack worth her while. If one were able to send 2^{17} random cookie packets per second to the firewall (which is well beyond the typical realistic rate limit of 100 megabit Ethernet), then it would take on average, 48.5 days to obtain a valid cookie. While not outside the realm of possibility, the gains of such an attack are limited since the cookie would be useless after the embedded expiration timestamp. In addition, such a random cookie would contain an unpredictable *RID,* preventing the attacker from choosing the services she gains access to. Administrators who remain concerned with this level of security could limit their use of *RID* values to 2, and set *T* equal to the TTL, yielding 55 bits worth of protection. Against the same attack rate, it would take on average 4358 years to obtain a valid cookie randomly.

5.1.2 Attacks on Guarantee #2, Expiration

Clock Roll-over and Cookie Reuse

There is an attack against Guarantee #2, which lies in the limited field width of the expiration timestamp. Every $(T \cdot 2^{24})$ seconds this timestamp will roll over, causing cookies generated previously to be re-usable during their limited window. If $T = 1\ sec$, this would occur about every 194 days, which is not terribly worrisome, but it does break this guarantee in the strictest sense. Larger values for *T* will provide additional protection, but will also reduce *TTL* granularity.

Defending Against Clock Roll-over Reuse

An easy workaround to this problem is to be sure to rotate the symmetric key each time the expiration field rolls over. This should be done periodically anyway, and doing so at least once

every clock roll-over would not be difficult to implement. Since old cookies would not properly decrypt with the new key, they would not be reusable.

5.1.3 Attacks on Guarantee #3, Record Association

So long as Guarantee #1 is satisfied, there does not seem to be any viable attack strategy for messing with, or divulging *RID*s as they are sent in the encrypted cookie to the firewall.

5.1.4 Attacks on Guarantee #4, Resolver Association

Offline *SID* Victimization

Since the *SID* field is based on a portion of the DNS resolver's IPv6 address, an attacker has some amount of control over it. One attack against this field is based on knowledge of the hashing algorithm. If an attacker knew the specific hash algorithm being used by the defenders, and the attacker owned a large enough address block, then it might be possible to generate collisions with other DNS cache addresses in an offline attack.

For example, if an attacker knew he wanted to obtain the same *SID* as some other, non-malicious DNS cache, he could repeatedly hash random addresses from his block until he found a collision. Then, simply sending a request from that IP would get him a cookie with the same *SID* as the other DNS cache. In this way, an attacker could effectively obtain the same level of trust that any other cache has been assigned by the defenders.

Defending Against Offline *SID* Victimization

Defeating this kind of offline collision attack is not difficult. The defenders can use a secure secret hash function or an HMAC to prevent the attacker from generating *SID*s offline. Since the *SID* values themselves are never exposed to an attacker, it should not even be possible for an attacker to brute force the key to such a hash function.

Multiple Source *SID* Hoarding

Another attack against Guarantee #4 is to hoard cookies with many different *SID*s. If a single attacker could obtain many valid cookies each with a different *SID*, then another (perhaps a DDoS, or other spoofed) attack could be launched using all of these, making it difficult to tie all of those attacks to a single entity or to block in a reasonable amount of time.

There are three basic ways an attacker could obtain cookies with a diverse set of *SID*s. The easiest would be to obtain a large block of IPv6 addresses, and then request a cookie from each of these. Secondly, an attacker could locate many third-party DNS servers, which provide recursive name resolution to the public, and request records through them. The final method would be to use a large distributed botnet where each node requests one record from the defending DNS cache, sharing this cookie with its owner or the rest of the botnet.

Defending Against Multiple Source *SID* Hoarding

The first strategy for defending against massive *SID* hoarding is to use the proposed address assignment scheme for the IPv6 Internet to our advantage. RFC 3177 [38] describes how addresses will be allocated to entities:

> - Home network subscribers, connecting through on-demand or always-on connections should receive a /48.
> - Small and large enterprises should receive a /48.
> - Very large subscribers could receive a /47 or slightly shorter prefix, or multiple /48's.

Therefore, it is a relatively safe assumption that a defending DNS server can hash just the leading 48 bits of a requesting IPv6 address to generate *SID*s. It would be very difficult for an attacker to obtain allocations of many 48-bit prefixes, and eliminates this first method of hoarding.

The use of third party DNS resolvers is a more difficult problem to solve. A recent survey of public DNS servers indicates that as many as 75% provide recursive resolution to arbitrary clients [54]. This means there will probably be no shortage of servers to use for the purpose of hoarding cookies. The good news is, there are ever stronger incentives for administrators to change this open resolution policy. For one, providing recursive resolution to arbitrary clients makes DNS cache poisoning attacks much easier [72]. Additionally, there is evidence that recent DDoS attacks have utilized recursive resolution with third party DNS servers to aid in attack amplification [13, 80, 82]. Hopefully in the near future, DNS administrators will become more aware of these problems, and fewer such servers will be available. In the near-term, other strategies to mitigate a large number of *SID*s will be necessary. Section 5.2.4 describes one approach for dealing with this problem through *SID* re-mapping.

The final method of *SID* hoarding, the use of a large number of compromised zombies, is also difficult to solve directly. So long as such botnets exist, mitigation algorithms will need to account for this. Fortunately, a good *SID* re-mapping algorithm should be able to deal with this situation.

5.1.5 Denial of Service Attacks

While this architecture is designed to help prevent some DoS attacks, it should not introduce any memory or CPU requirements which could lead to new DoS vulnerabilities. Some specific attacks and against the proposed system and some mitigations are described here.

DoS Attacks on DNS Server's Cookie Generation

One kind of denial of service attack possible on the cookie system would be to attempt to exhaust processing resources on the DNS server. While generating one cookie is technically an $O(1)$ operation with respect to CPU resources, it may require a large, constant amount of computation. An attacker could, even from a single host, send a large number of DNS queries, each of which requires two hashing operations and the encryption of the 80-bit block, which could cause service degradation rather easily. An attacker could spoof such request packets from random sources, effectively thwarting naïve caching schemes, thus remaining untraceable.

Defending Against Cookie Generation DoS Attacks

One approach to mitigating this attack would be to cache certain cookie values, such as the *SID* hashes to speed up computation. However, this probably would not work against a spoofed DDoS attack, so it may not be worth the effort.

A second, more effective mitigation, would be to use a single referral cookie for every query. The referral response would contain a name server address, which is a time-based cookie and is the same for all requesters. Basically, this would be the same as the normal cookie, but it would contain an *RID* which instructs the firewall to direct the traffic back to to the DNS server, and the *SID* would be a constant (not based on the source address). This would allow the DNS server to respond to all first-round queries very quickly and would require the resolvers to obtain this first cookie before getting the AAAA record. Non-spoofing record requesters would then be able to continue to request the AAAA record again, and the response to this second query would contain a normal cookie. Spoofing requesters would not get the first referral cookie back and would have a hard time forcing the DNS server to perform a full cookie generation for each request. The only flaw in this system is that in a DDoS attack, a single resolver could avoid spoofing its address, communicate the referral cookie to all other attacking systems, and then all of them could use it to attack with spoofed addresses. Future work may resolve this by using alternate referral cookie formats, which are resolver-specific but still easy to

generate/cache.

A third, perhaps more effective, solution to this problem would be to have the upstream ISP host a DNS server that generates such referrals. This adds yet another server to the mix that needs to be configured with the cookie keys, but this would push many DoS attacks against the DNS components upstream. For instance, if a DNS server located at the ISP handled all cookie generation, or just cookie-based referrals to the real DNS server, then no public services would need to be hosted on static addresses at the defender's main network.

DoS Attacks on the Firewall's Cookie Verification

Another, similar, denial of service attack is possible on the firewall side. Since the firewall needs to perform almost as much computation to verify cookies, it may be possible to exhaust processor resources by sending a large number of bogus cookies. This attack could also be performed anonymously by spoofing source addresses.

Defending Against Cookie Verification DoS Attacks

In order to fully verify an address cookie, the firewall needs to perform a full decryption, a checksum, an expiration check, and an *RID* check. An obvious way to optimize this for the average case would be to verify the *RID* and expirations are valid before running the checksum. In a worst-case scenario, suppose all 256 *RID*s are used (which means no cookies can be blocked based on that check) and all record TTLs are set to 7 days. The valid expiration window can be set to 7 days, or about $2^{19.21}$ seconds. Since our expiration stamps are in units of 256 seconds, this is the same as $2^{11.21}$ expiration clock cycles. With a 24-bit clock, the chance of a random stamp falling within this window is about $2^{-12.79}$. In other words, about one in every 7101 random cookies will need to have their checksums verified. On the other hand, in perhaps a more typical configuration where 16 *RID*s are used with a TTL of 24 hours, the chance of a random cookie needing to have its checksum verified is one in 794,121. For this reason, it does not appear the checksum operation is the computation of concern in this scenario.

Instead, a defending firewall will be spending most of its time decrypting the random cookies. Some minor speed improvements may be possible if a firewall can partially verify an address cookie without completely decrypting the contents. However, this greatly depends on the method of encryption. If the 3-round 64-bit encryption method were employed, and the contents of a cookie were arranged just right, then 16 bits worth of data could be checked (for instance, containing the high-order bits of the expiration stamp) after just two decryption

rounds. This would yield a 33% speedup in the exclusion of the majority of random cookie addresses. Of course, faster encryption methods would also speed this up, such as the reduced round variants or an elastic cipher. (See section 6.1 for more details on encryption methods.) Finally, using hardware implementations of 64-bit ciphers (DES implementations are common) would greatly alleviate this problem, as even cheap hardware solutions have produced speedups on the order of 60 times for CBC modes [47].

Another approach may help, which involves the caching of valid cookies. If a firewall maintains a cache of recently verified address cookies, it could quickly check this cache when new requests come in. This would speed up validation for clients who have recently been authenticated, and are still within their expirations. Then, if a firewall detects that it cannot keep up with the load of incoming cookies, it could randomly drop some portion of the new, non-cached cookies that need to be decrypted. This mode of failure is better than dropping a random selection of any incoming packet and would allow established connections and recently validated clients to continue interacting with the defending network. New valid clients would initially have no priority over the attackers and may have difficulty getting validated initially. Luckily, once a single cookie of theirs gets verified and cached, they would then be fast-tracked in later verifications, which would ultimately benefit them.

DoS Through Memory Exhaustion

One of the most effective resource exhaustion techniques is memory exhaustion. This is the reason SYN cookies were developed, and such attacks should be taken into account with any new security system. Fortunately the IPv6 address cookie scheme, as presented, is immune to such attacks since neither the DNS server nor the defending firewall need to store any information about cookies they generate and verify. While caching mechanisms will improve performance and help defend against other kinds of DoS attacks, these can easily be implemented with a fixed memory.

5.1.6 Summary of Attacks Against IPv6 Address Cookies

In summary, the most worrisome attack against the core architecture and protocol are the different forms of *SID* hoarding, as mentioned in section 5.1.4. If not properly mitigated, these attacks may render several of the applications of address cookies ineffective. Also of note are the many possible DoS attacks against the address cookie architecture. While it is possible to

mitigate them, a solidly implemented and tested software is essential, as it is with any security software.

5.2 Applications of Proposed Architecture

Provided that the guarantees specified previously hold up, we may base more advanced mitigation algorithms on them. The following strategies are designed to use these guarantees to defend against typical spoofed attacks (not attacks specific to this system).

5.2.1 Hidden Services

The most basic feature that IPv6 address cookies provide, given the proposed design, is a form of authentication reminiscent of port knocking techniques [44]. Port knocking, is a method of changing network access rules via a covert channel. It has many variants, but in its basic form it consists of a client and firewall which agree on a certain sequence of unused ports. When a client sends requests to the firewall with the correct sequence of destination ports, the firewall opens up access to a service for that client. The firewall never responds to the requests in any special way, other than opening up the predefined service. The purpose of this covert communication is to hide the availability of potentially vulnerable services. For instance, if only systems administrators are allowed to use SSH to a given server, then using port-knocking to hide it will protect it from online password brute-forcing attempts and zero-day exploits. This technique is generally only for services that are not offered to the general public. Recently, there have been many variants of port knocking proposed and developed. More modern versions even employ one-time password schemes to prevent passive replay attacks [84].

It should noted that, in general, port knocking systems are not intended to provide strong security. They merely serve as clever way to hide a service from scripted attacks. A dedicated attacker with the ability to sniff packets of a user connecting to a protected service would be able to overcome the majority of port knocking schemes.

With IPv6 address cookies, we can hide services in a transparent way. For instance, if an administrator wished to hide a specific service, only to be accessed over the Internet by employees, then he could distribute a secret domain name to his coworkers, perhaps of the form: SECRET.example.com. Of course, SECRET would be some strong password. He could configure his DNS server to map this domain to an *RID* used only by that domain. Then on the firewall,

a port filter policy could be set where only that *RID* is allowed to connect to the hidden service. So long as the DNS server is configured not to leak that domain name, then there would be no way for attackers to obtain a cookie with that *RID*, and no way to connect to that service. It would also be possible for the administrator to map multiple secret domain names to the same *RID*, and give each of his coworkers a different secret. A practically infinite number of these mappings can exist in the DNS server without the need to use up any more *RID*s. This would allow some secrets to be expired when certain employees no longer require access.

The main advantage of this approach over most port knocking systems, is that it is completely transparent to the client. No special port-scanning software is needed in order to perform the knock. All that is required is knowledge of a specific domain name. As it stands, replay attacks against this system are certainly a problem, but such secret domain names could be augmented with one-time passwords just as other systems have been.

5.2.2 Static Flood Mitigation Based on *SID* History

Since the *SID* field of an address cookie provides a defender with some information (though certainly imperfect) about the original source of requests, this can be used to mitigate spoofed denial of service attacks. While many *SID* values will be used by both attackers and valid clients, it is highly unlikely that the distribution of attacks will match the distribution of valid clients. In other words, if we were to calculate the ratio of attack rate to valid request rate for every *SID*, it is unlikely that this ratio would be the same across all *SID*s. Thus, a defender can mitigate some portion of attacks by choosing certain *SID*s to block, which have a history of mostly malicious requests. For instance, if an attacker is sending a huge number of spoofed SYN packets through cookies with a single *SID*, it may make sense to block that *SID* even though some valid clients will be blocked as well. If the majority of clients (outside of that *SID* group) would benefit from this blockage, then perhaps the resulting number of valid requests serviced would be higher than with doing nothing at all. The difficult part is to quickly determine what the optimum block strategy would be, given an arbitrary distribution of attacks and valid requests. Here, we describe such a strategy which is optimal given some assumptions.

The assumptions made by the following algorithm are as follows:

1. For a specific type of request, the maximum servicing rate is known and is constant. Above such a threshold, requests of this type will be dropped randomly due to limited

resources. We refer to this maximum rate as ***M***.

2. The rate of requests can be measured for each *SID*. In addition, when malicious requests are received, they can be identified as attacks some short time later through a reliable heuristic.

3. An attack against the defender's network is currently in progress, and will be for the foreseeable future. This attack causes the total rate of requests to be greater than ***M***. During the attack, the distribution of malicious requests across all *SID*s will not change significantly, nor will the distribution of valid requests. Thus, the attack is static in nature.

4. The defending administrator gives no preference or priority to any valid requesters. Her goal is merely to maximize the number of valid requests getting serviced, regardless of who they are coming from.

Now, for the sake of generality, let us assume that the defender will assign a probability to each *SID*. For every request received, it will be accepted with the probability assigned to its *SID*. Let ***N*** represent the number of possible *SID*s and $\vec{P}$ be the vector of ***N*** probabilities maintained by the defender for this purpose. In order to maximize the rate of valid requests serviced, we first must be able to compute this rate based on the parameters we have so far defined. Let $S(\vec{P})$ be the rate of valid requests serviced, where p_i represents the probability associated with the i-th *SID*, and $1 \leq i \leq N$. Also, let us define the v_i and a_i as the rate of valid requests and malicious (attack) requests, respectively, being sent via the *i*-th *SID*. $S(\vec{P})$ can be simply computed by:

$$S(\vec{P}) = F(\vec{P}) \cdot R(\vec{P})$$

where $F(\vec{P})$ is the rate of valid requests, regardless of *SID*, which make it past the probability filter. $R(\vec{P})$ is the probability that any given request will be successfully serviced once it reaches the server. (Recall assumption #1 states that if the number of requests reaching the server is greater than ***M***, then some will be dropped randomly.)

Obviously,

$$F(\vec{P}) = \sum_{i=1}^{n} v_i \cdot p_i$$

and

$$R(\vec{P}) = \frac{M}{\sum_{i=1}^{n}(a_i + v_i) \cdot p_i}$$

Therefore the expanded form of $S(\vec{P})$ is:

$$S(\vec{P}) = [\sum_{i=1}^{n} v_i \cdot p_i] \cdot \frac{M}{\sum_{i=1}^{n}(a_i + v_i) \cdot p_i}$$

Once again, as a defender, we would like to know what values to select for each p_i in order to maximize $S(\vec{P})$. To find these, we can take N partial derivatives with respect to the p_i's. The form of these derivatives, for some p_j (where $1 \leq j \leq N$) is:

$$\frac{\partial S(\vec{P})}{\partial p_j} = \frac{M \cdot (v_j \cdot [\sum_{i=1}^{n}(a_i + v_i) \cdot p_i] - [a_j + v_j][\sum_{i=1}^{n} v_i \cdot p_i])}{[\sum_{i=1}^{n}(a_i + v_i) \cdot p_i]^2}$$

$$\frac{\partial S(\vec{P})}{\partial p_j} = \frac{M \cdot ([v_j \cdot \sum_{i=1}^{n} a_i \cdot p_i] + [v_j \cdot \sum_{i=1}^{n} v_i \cdot p_i] - [a_j \cdot \sum_{i=1}^{n} v_i \cdot p_i] - [v_j \cdot \sum_{i=1}^{n} v_i \cdot p_i])}{[\sum_{i=1}^{n}(a_i + v_i) \cdot p_i]^2}$$

$$\frac{\partial S(\vec{P})}{\partial p_j} = \frac{M \cdot \sum_{i=1}^{n}(v_j \cdot a_i - v_i \cdot a_j) \cdot p_i}{[\sum_{i=1}^{n}(a_i + v_i) \cdot p_i]^2}$$

Notice, in the numerator's summation, that for the case where $i = j$, the term will go to 0. Therefore, we can rewrite this as:

$$\frac{\partial S(\vec{P})}{\partial p_j} = M \cdot [\sum_{i=1, i \neq j}^{n} (v_j \cdot a_i - v_i \cdot a_j) \cdot p_i] \cdot [\sum_{i=1}^{n}(a_i + v_i) \cdot p_i]^{-2}$$

Hence, the partial derivative depends upon p_j only in the denominator, and therefore the function will be positive or negative, for all values of p_j in that range, depending solely on the summation in the numerator.

If, for a specific j:

$$\sum_{i=1, i \neq j}^{n} (v_j \cdot a_i - v_i \cdot a_j) \cdot p_i < 0$$

then

$$\frac{\partial S(\vec{P})}{\partial p_j} < 0$$

This is helpful, because we know that if a partial derivative with respect to p_j is always negative, then $S(\vec{P})$ will be monotonically decreasing with respect to p_j. If an administrator needed to select only one p_j (e.g., $\vec{P}$ had only one element), then the choice would be obvious.

If:

$$\frac{\partial S(\vec{P})}{\partial p_j} < 0$$

then set $p_j = 0$, otherwise set $p_j = 1$. Unfortunately, many probabilities must be selected, and each selection affects the numerators of the other partial derivatives. However, what if we found that for some particular j, every element in the numerator summation of $\frac{\partial S(\vec{P})}{\partial p_j}$ were negative or zero? This would imply that no matter what values we select for its p_i's, the resulting slope would be negative. Therefore, we could set this p_j to 0 with confidence that later selections of p_i's would not affect the sign of $\frac{\partial S(\vec{P})}{\partial p_j}$. In addition, we can prove that one such j always exists.

It is obvious that:

$$\exists j \;\; s.t. \; \forall i : \;\; \frac{v_j}{a_j} \leq \frac{v_i}{a_i}$$

Meaning, out of all $\frac{v_i}{a_i}$ ratios, there exists one which is smaller than or equal to all the rest. From this we conclude:

$$\exists j \;\; s.t. \; \forall i : \;\; (v_j \cdot a_i - v_i \cdot a_j) \leq 0$$

which means if we just select the j with the smallest $\frac{v_j}{a_j}$ ratio, we can always set $p_j = 0$. By induction, it is easy to see that once our first p_j is set to 0, it drops out of all the other partial derivatives, and we will have a new j which now has the smallest ratio. One would use Algorithm 3 to select an optimal $\vec{P}$.

Algorithm 3 STATIC SID BLOCK POLICY GENERATION

Let P be an array of length N
Let $T \leftarrow \sum_{i=1}^{n}(a_i + v_i)$
Sort list of SIDs by $\frac{v_i}{a_i}$ ratio, ascending
For $i \leftarrow 1$ to N:

    $P[i] \leftarrow 1.0$

For $j \leftarrow 1$ to N:

    If $T \leq M$:
        QUIT
    Else if $(T - M) < (v_j + a_j)$:
        $P[j] \leftarrow 1 - \frac{T-M}{v_j+a_j}$
    Else:
        $P[j] \leftarrow 0$
    $T \leftarrow T - (1 - P[j]) \cdot (v_j + a_j)$

In essence, this algorithm selects the *SID* with the most malicious ratio of attacks, and blocks it first. If after blocking it, the resulting number of requests is still greater than M, it continues onto the next one. Once it reaches the *SID* whose blocking will result in $T < M$, it will block the minimum amount of that *SID* in order to make $T = M$.

This method of blocking could be very effective against attacks directed at the web server, but it is basically pointless if an attack is merely meant to overload the CPU of the firewall, or to choke up the bandwidth of links to the firewall. This is why the upstream ISP router component is important. This router would need to at least validate cookies to slow random cookie attacks, but it also would need to honor an *SID* filtering policy, based on the algorithm just described, in order to be effective. Fortunately, even with this addition, the blocking algorithm for the upstream would remain stateless and simple. If the TTL is valid, and the *SID* policy does not block it (probabilistically), then it would be allowed. The *SID* block policy on the upstream router would likely be updated by the firewall through some out-of-band messages sent whenever the firewall updates its own policy.

5.2.3 Dynamic Flood Mitigation Based on *SID* History

While the algorithm that resulted from the analysis above is certainly nice and simple, a big assumption was made. Real life is never static, and the Internet is even less so. How one

detects and measures attack rates and request rates from valid users is very important for the overall success of such an algorithm. At the very least, the problem of tracking attacking *SID*s after they have been blocked needs to be addressed. For instance, when a block policy is generated and put in place, how does one determine when an attacker has ceased the attack? If 100% of requests from a specific *SID* are blocked, then there would be no way to determine the ratio of valid requests to attacks. For this reason, it is important to allow a certain percentage of attacking *SID*s through even though this will slightly degrade the performance in the static attack scenario. One possible algorithm for a more dynamic block strategy can be found in Algorithm 4.

Algorithm 4 DYNAMIC SID BLOCK POLICY GENERATION

Let P_{new} be an array of length N
Let P be an array of length N
Let M_{test} be a small constant based on M (on the order of $\frac{M}{10000}$)
Let $T \leftarrow \sum_{i=1}^{n}(a_i + v_i)$
Sort list of SIDs by $\frac{v_i}{a_i}$ ratio, ascending
For $i \leftarrow 1$ to N:

    $P_{new}[i] \leftarrow 1.0$

For $j \leftarrow 1$ to N:

    $R \leftarrow (\frac{v_j + a_j}{P[j]})$
    If $T \leq M$:
        QUIT
    Else if $(T - M) < R$:
        $P_{new}[j] \leftarrow 1 - \frac{T-M}{R}$
    Else:
        $P_{new}[j] \leftarrow \frac{M_{test}}{R}$
    $T \leftarrow T - (1 - P[j]) \cdot R$

$P \leftarrow P_{new}$

This adjustment is much better, as it calculates the overall rate of an attack based on the collected data and the current block rate (stored in P). In addition, it allows a small portion of the attack to come through for each *SID* so that changes in the attack rate and other request rates can be monitored. Obviously, the algorithm would be slightly more complex if an upstream router were configured with a block policy as well, but the same basic technique could be used. Since this algorithm sorts *SID*s only by their ratio of valid requests to attacks, blocking the majority of requests from a few *SID*s does not affect the result of future policy calculations.

The only tricky part here is picking a good M_{test}. It is suggested that something on the order of $\frac{M}{10000}$ be used, but this is mostly arbitrary. The risk with using any specific constant in the denominator could cause the system to be overwhelmed if an attacker could send a flood using that many *SID*s at a rate of M_{test}. A better method of the selection for M_{test} has been left to future research.

Even with this improved algorithm, there are several questions remaining. How would one track changing attack and valid request rates over time? How often should one update the block policy? These questions remain open. It seems that the tracking of request rates would be best accomplished with commonly used exponential weighting formulas. However, it seems that no matter what time-weighted formula one uses, there will always be some form of modulation attack that could at least partially defeat the defense. Imagine a clever attacker who first attacks under a single *SID*. Just when the firewall begins to learn the correct ratio of valid requests to attacks on that one *SID*, the attacker switches to a new *SID*. Repeat ad nauseum. Over time, the average failure rate for valid clients would certainly be higher than with a static attack, because of the time it takes to learn the new attack rate. This modulation attack is so far the most significant threat to the system. One possible defense is proposed the next section.

5.2.4 Semi-realtime *SID* Remapping

One problem with *SID*-based filtering is the fact that it is possible to have an entirely innocent DNS resolver be blacklisted due to *SID* hash collisions. Figure 3 illustrates a possible scenario in the mapping of clients to resolvers and resolvers to *SID* hash values. Attacking clients and the resolvers they are using are black. Non-malicious clients are white, as are resolvers which do not have any attackers using them. Finally, *SID*s which map to one or more attacking resolvers are grey.

While it does not seem possible to decipher the true attacker's identities through IPv6 address cookie alone, it should be possible to determine exactly which DNS resolvers are being used. By using simple *SID* remapping, it is possible to weed out the resolvers which are being used for attacks from those who just happen to hold the same *SID* value.

A simple way to accomplish this would be to use the logging server located in the defender's network to make the remapping decisions. This system would receive logs from both the DNS server and the firewall. The DNS server would provide information on the association between

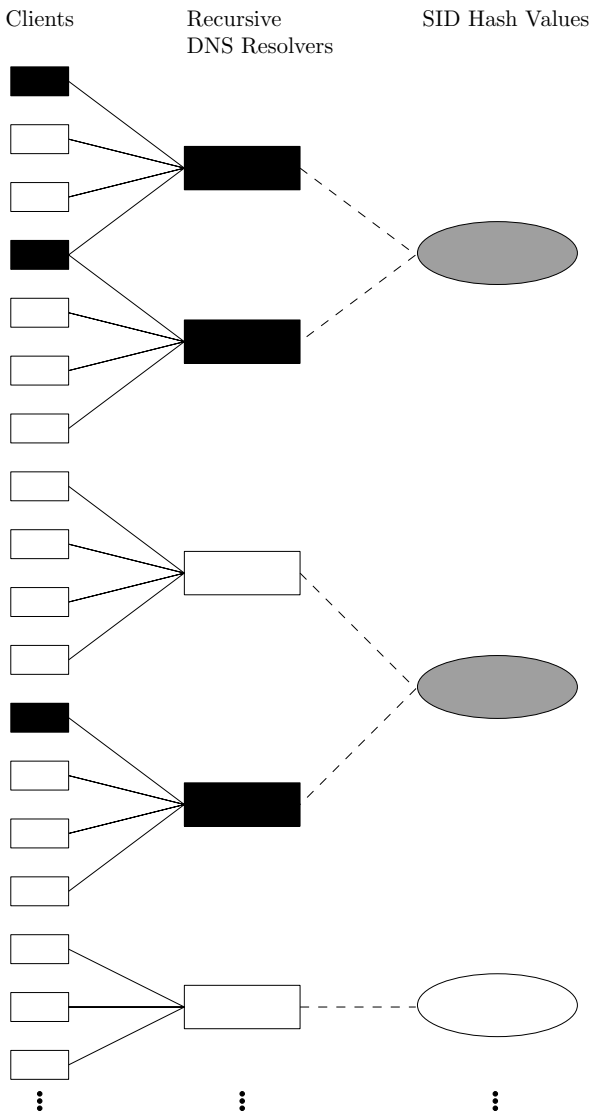

Figure 3: SID Mapping Scenarios

*SID*s and resolver IP addresses. The firewall would provide the list of all *SID*s recently seen, and their attack and valid request rates. With this information, the offline log processor would be able to determine where *SID* collisions exist. In order to weed these out, the log processor could send commands to the DNS server to make special exceptions for certain DNS resolvers. The next time the resolver asks for a cookie, the *SID* returned would be based on some static value in a lookup table, instead of being based on the source address.

For instance, in Figure 3, the second *SID* is associated with one resolver which is being used for attacks, and another resolver that is not. The offline log processor wouldn't know which is the attacker (or if both are), but could perform an experiment to find out. By asking the DNS server to re-map one of these resolvers' IPs to some *SID* that has no attackers behind it, it could be determined which of the two resolvers are being used for attacks, based on later log results. This of course could be generalized for a large number of resolvers being associated with one *SID*. Over enough time, the *SID*s could be re-mapped sufficiently such that no benign resolvers would fall in the same *SID* with a malicious resolver.

Once the set of all malicious resolvers is found, the offline log processing agent could begin mapping malicious resolvers into the same *SID* for more efficient filtering at the firewall and upstream ISP. This is actually a very important mitigation against *SID* modulation attacks that are not easily addressed with dynamic *SID* filtering alone. For example, if the log processing agent identified a modulation attack over a long period of time, the DNS resolvers being used in the attack could simply be mapped to a single *SID*, eliminating the problem for the firewall. This kind of blacklist remapping must be used with caution, however. Each remapping causes the firewall to have to relearn its distribution of attacks some small amount, and mapping too many resolvers with a single *SID* degrades the firewall's ability to differentiate between requests if attacks change in scope. More analysis of the costs and benefits is certainly needed before a safe and efficient algorithm for remapping can be developed.

5.2.5 Slowing Attacks Through Repeated Referrals

Another simple defense strategy lies in the use of a reactive DNS server. Since clients are now required to use the DNS in order to use any other services, one can apply several delay tactics to slow attacks. For instance, if it is decided through specific heuristics that a certain DNS resolver is "evil" through association with the clients using it, or by other means, then the DNS server could intentionally delay responses to it for some period of time, marginally

slowing any attacks that use cookie addresses. A more effective strategy against such DNS resolvers, would be to return repeated referrals to other cookie addresses. By creating a chain of referrals to other cookie addresses (which are ultimately mapped back to the same DNS server), a DNS server can delay queries even further. While this may not seem like a strong mitigation, it could slow an attack by a substantial percentage in the instance of DDoS attack where all clients are requesting records separately and TTLs are low. An even more extreme mitigation strategy would be to respond to the most malicious of DNS resolvers with bogus addresses (e.g., the loopback address). This is risky, since all clients who use that resolver would be impacted. However, with a small enough TTL this may be an acceptable risk.

Obviously, these strategies have the potential of opening the DNS server itself up to DoS attacks, and any such algorithms should be mindful of the current load on the DNS server. Fortunately, it is trivial to distribute the DNS request load across many servers, so in large networks this should not be an major issue. An additional limitation of this approach is that it will not mitigate attackers who simply flood a network with packets to invalid cookie addresses. Therefore, these mitigations are useless against many attacks unless upstream filtering is used at the ISP to validate cookies.

5.2.6 Offline Traceback

The last benefit of IPv6 address cookies we have identified is related to the traceback problem. If one were to use *SID* remapping to find the list of DNS resolvers used by an attacker, tracing the attack back to one or more of the attacking systems would be much easier, assuming a large portion of these resolvers are not compromised themselves. Defending administrators could contact the administrators of the resolvers, and ask them for logs or even ask them to disable anonymous recursive resolution, if that were enabled. While this does not completely solve the traceback problem, it does almost as well as many proposed algorithms without requiring any modification to core routing protocols. This could be very helpful, for instance, against smaller DoS attacks and other spoofed scans where a forensic investigation were being conducted.

6 Design and Implementation

6.1 Cryptographic Considerations

While the cookie format proposed above is encrypted, its purpose is not to provide secrecy for the plaintext. Secrecy may play some role in protecting cookies against certain attacks, but the real purpose of encryption is to provide enhanced integrity. In order to authenticate cookies later, and be reasonably sure an attacker cannot manipulate a cookie offline, some amount of redundancy is needed. This is primarily provided by the checksum, but it is unfortunately short due to the restrictions on cookie length. To prevent attackers from manipulating the checksum and/or the other data fields independently, we need to introduce some unpredictable diffusion into the cookie. This is most easily accomplished through a block cipher which encrypts the entire 80-bit cookie as a single block. Unfortunately, there are not any readily available 80-bit block ciphers in the public domain, and this presents an engineering challenge.

6.1.1 OFB and CFB Modes

The first option considered as a solution for this problem are the OFB and CFB cipher modes. One can quickly rule out OFB mode, since it does not provide any diffusion of the plaintext. There are however, problems with using CFB mode as well. First is the issue of an initialization vector (IV). Normally one needs to use a new randomized IV for every plaintext encrypted, but in this situation, we do not have any bits to spare for transmitting an IV in the cookie. Instead, we could choose secret rotating IVs, but this is difficult given the unpredictable way in which cookies will be used after they are issued, and may leave them open to attack. Even without this hurdle, there are limitations to the amount of diffusion CFB mode can provide. As a cipher in CFB mode progresses through the ciphertext, it diffuses the information from the plaintext into subsequent encryptions. Therefore diffusion is provided only in one direction: later encryptions are affected by plaintext of earlier encryptions, but earlier ciphertext would not be affected by modifications to later ciphertext blocks. This limitation would require us to do multiple passes in CFB mode (perhaps forward and backward), in order to achieve full diffusion. While possible, it does not seem like the easiest or best performing solution, so other approaches were considered instead.

6.1.2 Elastic Ciphers

A survey of papers on variable length block ciphers turned up some interesting related research. Ritter proposed a framework for a variable length block cipher which purportedly achieves good diffusion [61]. However, the literature does not specify a concrete implementation, and the apparent lack of peer review (even though the work was published in 1998 or earlier) does not bode well for the security of the approach.

More recent work by Cook, et. al. [17, 18, 19, 20, 21, 22] proposes a scheme where existing block ciphers are modified to stretch the block length by an arbitrary number of bits up to double the width of the original cipher. In these *elastic* ciphers, additional rounds are added in propoportion to the amount the block is stretched, giving them nice speed properties. This approach also makes proving aspects of security a bit easier, since the strength of the original cipher can be assumed, and proofs can be based on that security. Elastic ciphers are very new, however, and thus a lack of peer review remains an issue. Nevertheless, a proposed application of elastic cipher techniques to Blowfish [65] is included in Appendix A. Its security is in no way guaranteed as it is merely included to demonstrate how a fast (and patent-free) 64-bit block cipher could be stretched to 80 bits for the purposes of cookie encryption.

6.1.3 Triple Block Encryption

A simpler approach to the problem of encrypting an 80-bit block, involves multiple encryptions of the plaintext. Supposing we use a 64-bit block cipher, we could achieve almost complete diffusion of an 80-bit block by encrypting the first 64 bits in place, followed by the last 64 bits (which of course would include the last 48 bits of the output of the first encryption), and finally a third encryption of the first 64 bits would be repeated (which includes the first 48 bits of the second encryption). The diagram in Figure 4 helps to illustrate this process.

At first blush, one might wonder why three encryptions are necessary, instead of just two. The reason for this, is if only two encryptions are done, then the last 16 bits of the plaintext would not influence the value of the first 16 bits of the ciphertext. This could (depending on the contents of the plaintext) open up the encrypted block to attack through the modification of the last 64 bits. So, the third encryption remixes the output of the second block (which is influenced by the last 16 bits of plaintext) with the first 16 bits of ciphertext. Arguably, the diffusion achieved in this scheme is not completely symmetric as some input bits may influence

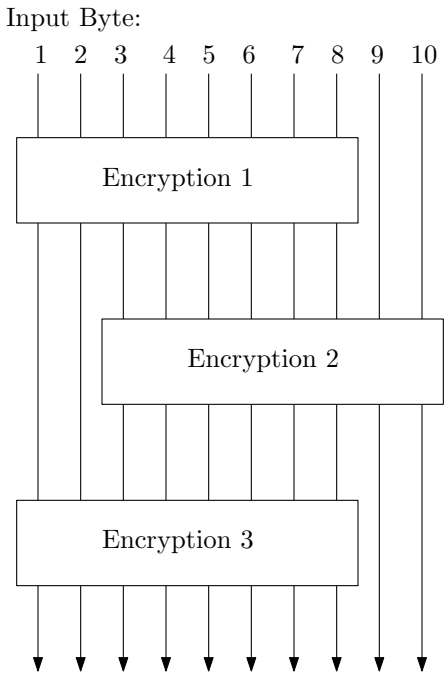

Figure 4: TRIPLE BLOCK ENCRYPTION

some final output bits more than others. However, given the amount of remixing on the second and third encryptions, it doesn't seem like an easily exploitable flaw. Those concerned with this may be able to increase the rounds of encryption (perhaps up to five in total) in order to achieve more uniform diffusion.

This method of encryption is certainly easier to implement than either the CFB or elastic approaches. While 64-bit block ciphers are no longer popular among cryptographers, several remain resilient to publicly known attacks. However, three encryptions is a bit slower than what could be achieved through an elastic cipher by as much as a factor of 2.

6.1.4 Double Block Encryption

While it is necessary to use at least three encryptions to achieve full diffusion on arbitrary 80-bit plaintexts, one can get by with less given the proposed cookie format. Recall that the last 24 bits of the cookie is a cryptographic hash of the first three fields. If one were to change values in any of these first fields, chances are that the checksum would also need to change. The checksum is, by definition, redundant information and it can be relied on for diffuse into the encryption blocks.

For instance, if one were to modify the previous encryption scheme by removing "Encryption 3" from Figure 4, then it would be possible for an attacker to modify the first two bytes of a cookie without it affecting the last two bytes of the resulting plaintext. (This would result in the first 8 bytes of the decrypted plaintext being essentially random.) Since the last two bytes are a checksum of the earlier bytes, the cookie would be invalid with a high probability. Taking into account the resulting timestamp would also be random; the chance of such a modified cookie being valid no more likely than in the three encryption case.

6.1.5 Single Block Encryption

By analyzing attacks against the specific cookie format even further, it becomes apparent that one could get away with just a single 64-bit encryption. If we were to remove both "Encryption 2" and "Encryption 3" from Figure 4, we would leave 16 bits of the checksum unencrypted. If we then change the 24-bit checksum into an HMAC, it would not leak any information about the data being authenticated, and the cookie would remain difficult to brute force.

For certain filtering schemes, there are attacks available that may be exploited with this encryption scheme if verification is not carefully implemented. Recall the random cookie flooding attack described in section 5.1.5, where an attacker sends large numbers of spoofed requests to random destination addresses. If a single round of encryption is used, an attacker could bypass the basic checks performed by the ISP router on the TTL of a cookie, if a valid cookie were modified randomly. Specifically, an attacker could first obtain a single valid cookie, and then systematically alter the last two bytes (containing 16 bits of the HMAC) and send these cookies in a spoofed attack. Because the information stored in the first 64 bits would still be valid, the TTL check performed by the ISP router would be bypassed. Then, when cookies reach the firewall, it would first check the TTL and the *RID*, which would both be valid, and then be forced to check the HMAC, which is a more expensive operation. In order to defend against this attack with this cookie format, one would need to employ the *SID* filtering mechanisms described in section 5.2.2 on the upstream router. Because the attacker would not be able to manipulate *SID* values in this attack, certain *SID*s could be blacklisted at the router and the attack would be properly stopped.

6.2 Software Implementation

6.2.1 Platform Selection

For the purposes of developing a proof of concept system, it was important to find software packages which were easy to augment. Oak DNS server [27] (version 1.2) was selected because it is written in a high-level scripting language (Python) which reduced development time significantly. As for firewall software, we settled on using Click Modular [42] as a platform to build two modules: a stateful, address translating firewall, and a stateless cookie-validating router. While Click Modular is written mostly in C++, we found it very easy to extend with minimal impact on surrounding code, and flexible as a testing platform. In addition, Click Modular is surprisingly fast, considering the amount of flexibility it imparts on users and programmers.

The additions to Oak were very simple. A new configuration syntax was added to the configuration file format, and an external configuration file is parsed which stores the secret keys used in cookie generation. No DNS DoS mitigation code was added to Oak, as it did not seem useful to test such attacks against such a slow language. In the end, less than 200 lines of code were added to Oak.

Far more work was put into the Click modules (called elements) which handle cookie validation and some more advanced filtering techniques, such as *SID* real-time blocking. Another, smaller, Click element was created to play the role of the filtering ISP router. This simple element merely decrypts cookies and validates their expiration fields before passing them on. Overall, the Click code totals around 1700 lines of C and C++ source. While not a trivial addition of code, this demonstrates how an IPv6 address cookies implementation would not be a huge undertaking, or be a major risk to perimeter security due to code complexity.

6.2.2 Features Implemented

In the prototype implementation, address cookies were encrypted using the two-round block encryption method (see section 6.1.4) and the checksum and *SID* were generated using SHA-1. Much faster secure hashing algorithms exist which would work well for this application, but this additional overhead had a negligible impact on the specific experiments being run. A known-good cookie cache is maintained by the firewall module, which is designed to help mitigate bad cookie floods. However, testing against this type of attack has not yet been conducted.

The main mitigation algorithm implemented in the firewall was dynamic *SID* filtering, as described in section 5.2. In order to make address cookies transparent to servers, the firewall module translates destination addresses of incoming connections after cookie validation. It stores state for each connection in a primitive way, in order to speed up translation and to provide a simple heuristic for tracking TCP SYN floods. The firewall module tracks rates for the various types of connection attempts and dynamically responds to SYN floods by partially blocking the most "evil" of *SID*s. No mitigation algorithms were implemented in the DNS server, as the proposed techniques were developed later in the research process.

6.2.3 Engineering Challenges

During the course of development, a number of implementation challenges arose. Foremost, the method of tracking request rates over time needed to be solved. Also, a heuristic for the detection of a specific attack was needed in order to perform proper *SID* filtering. In order to track packet rates over time, a standard exponential moving average formula was used:

$$M_n = \alpha \cdot M_{n-1} + (1 - \alpha) \cdot m_n$$

where n is the index of the current packet, M_n is the newly calculated average, and m_n is the data of interest for this packet.

Using this averaging formula, the current global request rate is tracked, in addition to request rates for each *SID*. Rates are calculated by keeping a running average of differences between arrival times of packets. Simply taking the inverse of this average delta provides one with a rate. Tracking the packet arrival delta, rather than the instantaneous rates directly, is much easier to do when system clocks are not very granular. Over time, the average request rate tracks accurately and responsively through the exponential moving average formula. This technique is very similar to the one used by TCP implementations to track average round-trip time. Other variants of moving averages were tried, but experimentation proved this method to be the most accurate for basic tests. Different approaches are almost certainly possible, however.

The second major engineering challenge that had to be overcome was the detection of malicious requests. Since the goal of this research is to develop strategies for reacting to attacks, and not

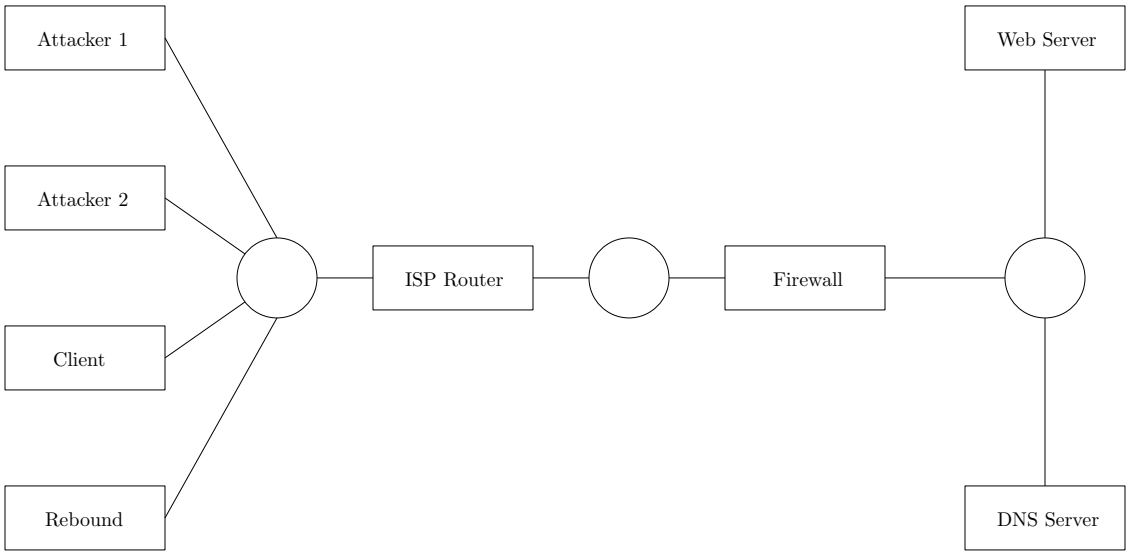

Figure 5: TEST NETWORK

to develop heuristics for attack detection, a simplistic approach was used for the detection of SYN floods. Because the firewall already tracks the state of connection, it was not difficult to add attack detection based on this. For each *SID*, three rates were recorded: blocked request rate, allowed request rate, and valid request rate. When a handshake completes successfully, it is added as a valid request. Valid requests are a subset of all allowed requests, and the allowed attack rate can be extrapolated from those. While the algorithm described in section 5.2.3 does not require one to track the blocked request rate, we found it more stable to do this, rather than trying to extrapolate this rate from the small amount of traffic making it past high-percentage *SID* filters.

7 Experimental Results

7.1 Test Environment

In order to test the software implementation in a realistic environment, a number of systems had to be built. To make this task easier, a VMWare® ESX server system was used to create a set of virtual machines which were used for testing. Figure 5 contains a diagram of the network configuration.

Each of the eight virtual machines were running Debian GNU/Linux with a custom-configured

kernel, version 2.6.15. The circles shown in the diagram represent virtual switches. The hardware running all of these virtual machines was a quad-CPU Intel® Xeon™ (3.16GHz with HyperThreading) server with 16GB of main memory. All systems were configured to handle normal IPv6 traffic, but the router and firewall were not set up to route IPv6, since Click Modular handled that.

The purpose of the attacking systems was to send SYN floods to the web server. The client system was used as a test machine to empirically detect the rate of TCP handshake failures from non-malicious clients. The rebound host was placed on the network to simulate an Internet-like environment. This system was configured to claim a block of 256 IP addresses. When the attackers send their spoofed requests, they use randomized source IPs from a larger range, which include this block of IPs owned by the rebound system. In this way, some percentage of responses from the web server were reset by the rebound host, with the rest simply being dropped. The router shown in the diagram was intended to act the part of an ISP's router. The firewall is of course the host which implements the *SID* filtering algorithm. Finally, the DNS server is the system which generates the address cookies, and the web server is the system which was attacked with SYN floods.

7.2 Experiments Performed

The first experiment conducted was designed to test the effectiveness of the TCP SYN cookies implementation in the Linux kernel. Unfortunately, the Linux kernel does not yet support SYN cookies for IPv6, which means the result of this test is difficult to compare with the later tests. However, it does give a baseline for relative performance expectations.

For this test alone, native Linux routing code was used to move IPv4 traffic over the router and firewall. The relevant TCP options configured in the kernel were left to their defaults: the number of SYN/ACK retries was set at 5, and the number of half-open connections was set to a hard limit of 1024. Performance data were collected when SYN cookies were off and on under a number of different attack rates. For each test run, an attack was initiated from one of the attack machines at a specific rate. Once the attack had stabilized (i.e., the SYN backlog on the web server reached equilibrium), a script was run on the client system to test connectivity. A TCP connection was initiated approximately every 0.75 seconds with a 0.75 second timeout. If the connection could be established in that time period, a short HTTP conversation ensued and

success was reported. Otherwise, a failure was logged. These tests were repeated for a period of five minutes for each data point.

The second experiment tested the performance of IPv6 address cookies in a best-case scenario. This was run in a very similar way to the SYN cookies experiment, as there was one attacker, one testing client, and each attack rate was tested with and without the use of address cookies. Click Modular was used for routing the IPv6 traffic, and the firewall was configured to use its *SID* filtering algorithm when cookies were enabled. The same timings were used to test failure rates and the TCP settings on the web server were also the same. The α parameter chosen for exponential moving average (to measure traffic rates) was set to 0.999. This α value was used because it produced consistent results in preliminary tests while not requiring overly long to adapt to changing request rates. The firewall was also configured with a maximum rate limit of 32 connections per second, since tests without cookies indicated this was approximately the maximum flood rate the web server could handle without dropping new legitimate connections. The expectation of this experiment is that the address cookies should perform similarly to SYN cookies because all attacks were coming from one *SID* and all connectivity test from another.

The third experiment was designed to test address cookie performance under more complex and realistic attack scenarios. In this test, two attacking *SID*s and two testing *SID*s were used. There were four different test runs. In the first run, the attacking *SID*s were distinct from the testing *SID*s. In the second run, there was one attacking *SID*, one *SID* whose traffic consisted of tests and attacks, and a third *SID* where the traffic was just tests. The third test run had just two *SID*s, both of which had an attacker and a tester. The fourth and final run had two attackers and two testers, but address cookies were not used. The α parameter and connection rate limit were the same as in the second experiment, but tests were run for just over three minutes, instead of five. In this experiment, it was expected that the first test run perform as well as address cookies had in the second experiment. The second run should perform somewhere in the middle between the best-case and worst-case. Finally, the third and fourth test runs should be similar, since the *SID* filtering algorithm cannot protect against identical *SID* ratios.

The fourth experiment tested the ability of the *SID* filtering algorithm to react to changing attacks. Two attacking *SID*s were used along with a single testing *SID*. A single attacker alternated its attack between the two *SID*s at different frequencies. All attacks were run at a rate of 512 packets per second. Three test runs were conducted, each with a different value

for the α smoothing parameter. The values used were 0.999, 0.995, and 0.990. Tests were run once again for a period of five minutes. While no formal analysis on modulated attacks has yet been done, the expectation here is that the attacker will have a single frequency which is his "sweet spot". At this frequency the attacker will switch the attack to the other *SID* at just the moment when the firewall finally catches up with the new attack rate. At any other frequency, the attack should be less effective. Each α parameter should have a different sweet spot, as this determines the firewall's responsiveness to change.

7.3 Data and Analysis of Results

The first experiment turned out much as expected. Using TCP SYN cookies over IPv4 allows the web server to hold up much better under attack, as we can see from the results in Figure 6.

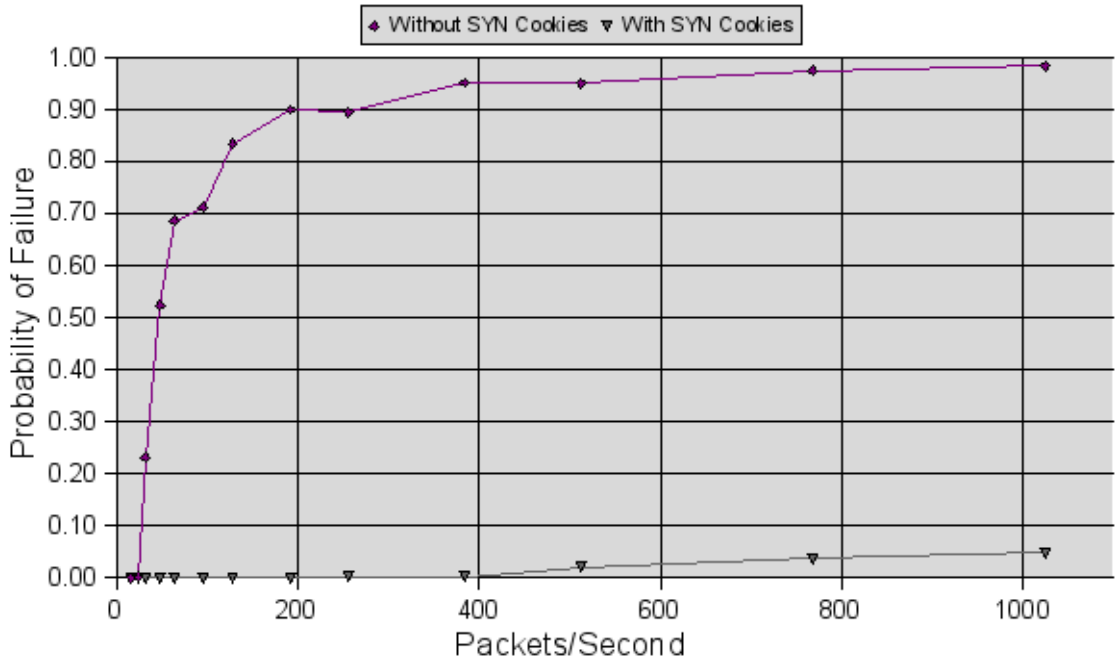

Figure 6: IPv4 TCP SYN COOKIES PERFORMANCE

With SYN cookies enabled, the web server did not start dropping legitimate connections until attack rates of 300-400 packets/second, whereas no protection at the same rates resulted in over 90% loss. One can easily see why SYN cookies have dramatically mitigated SYN floods in today's Internet. The packet loss that appears a bit later in the SYN cookie graph has not been explained. It may be that generating cryptographic cookies has high enough overhead

as to result in some dropped SYNs at that rate, or that some packet loss is occurring on the VMWare virtual switches.

One oddity worth mentioning was observed while testing under IPv4. While the proc filesystem reported the maximum number of half-open connections was 1024, the observed number under a flood was around 768 when SYN cookies were off. However, when SYN cookies were turned on, the observed number went up to 1024. Once again, the proc filesystem reported 1024 for both cases which may be the result of a bug, or a misunderstanding of how figures are reported. Under the later IPv6, the number of observed connections and the half-open limit coincided as expected.

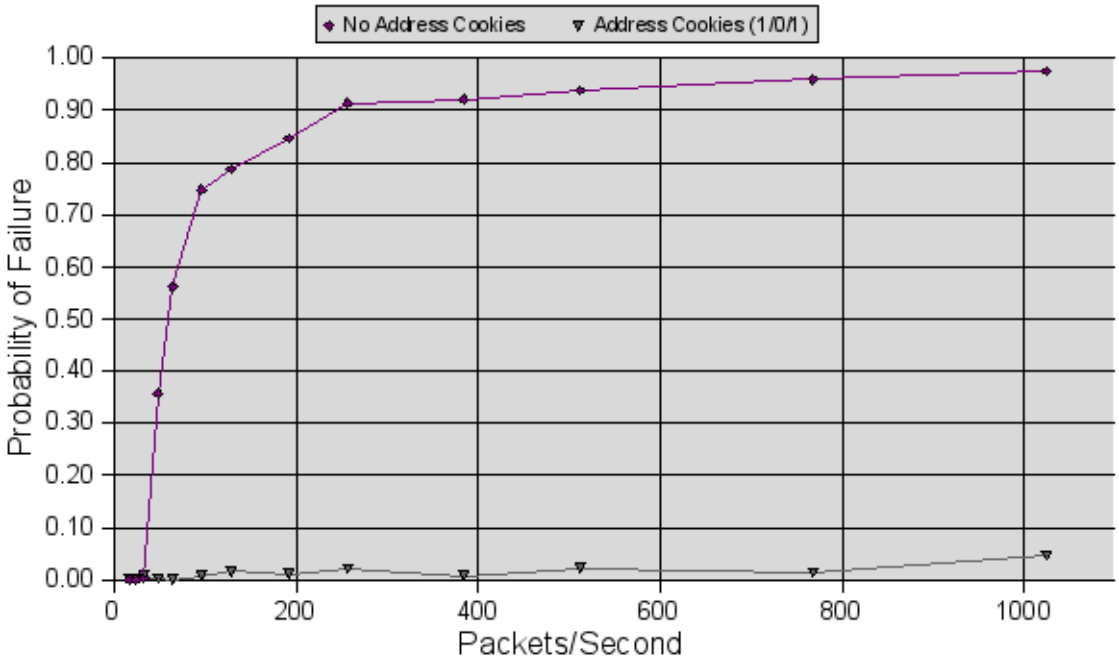

Figure 7: IPv6 Address Cookies - One Attacker, One Client

The second experiment also went much as expected (results in Figure 7). IPv6 address cookies performed similarly to SYN cookies in this best-case scenario. These results are slightly less consistent, but that is to be expected given the dynamic nature of the *SID* blocking algorithm. Looking at this graph alone, one might be lead to believe that IPv6 address cookies perform as well as SYN cookies. However, remember that this is the best-case, where attackers and valid clients do not share any of the same *SID*s.

The third experiment, which involved the mixing of clients and attackers in the same *SID*s, showed a significant performance degradation in the higher packet rates. In this graph (Figure

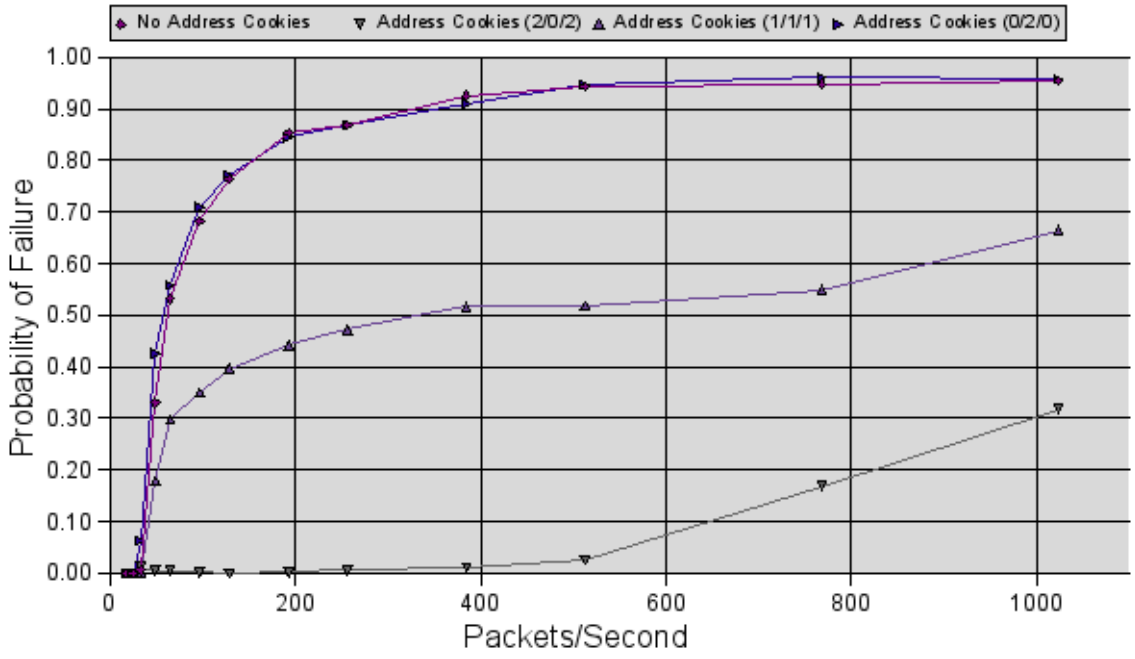

Figure 8: IPv6 ADDRESS COOKIES - TWO ATTACKERS, TWO CLIENTS

8), the best-case scenario (bottom line) is where two attackers and two clients were deployed, all on seperate *SIDs*. The attack rates shown reflect the total of attack rates from both attackers. For some reason, after a rate 512 packets/second, the failure rate begins to rise sharply. This is very different from the previous test, and one would expect them to be similar. The middle line also shows a more moderate rise of failures toward the end. There are several possible explanations for this. First off, the additional hosts participating in the test would have generated additional neighbor discovery messages, further clogging the network for the same attack rates. While these discovery messages do not add a lot of additional traffic, it may have been just enough to push an existing VMWare virtual switch bottleneck to further failures. Notice, that the first two graphs were not immune to failures at the last few data points; they just had fewer. Alternatively, the failures could be a result of some unintended oscillation in the *SID* block policy which would need to be corrected with improved measurement algorithms. It was observed that small inaccuracies in measured attack rates can result in relatively high percentages of failures due to inadequate or overzealous blocking.

The good news about the third experiment is that the rest of the data came out according to theory. The case where one client *SID* was mixed with an attacker came out right in the middle. The firewall was forced to block this mixed *SID* in order to save the web server from

being overwhelmed. This results in a failure rate that is right in the middle of the best case and worst case scenarios, as one would expect. The other piece of good news is that the worst case scenario, where all *SID*s have the same ratio of valid clients to attackers, came out with failure rates almost identical to the situation where no protections were deployed at all. This implies that even in the worst case, an administrator would not be any worse off by deploying address cookies with *SID* filtering. In most cases, there would almost certainly be some level of improved resistance to attack (at least against static attacks).

The results from the final experiment (see Figure 9) are not impressive, but they did come out as one would expect. In these modulated attacks, the current *SID* filtering implementation has a hard time keeping up with the changing attack flows.

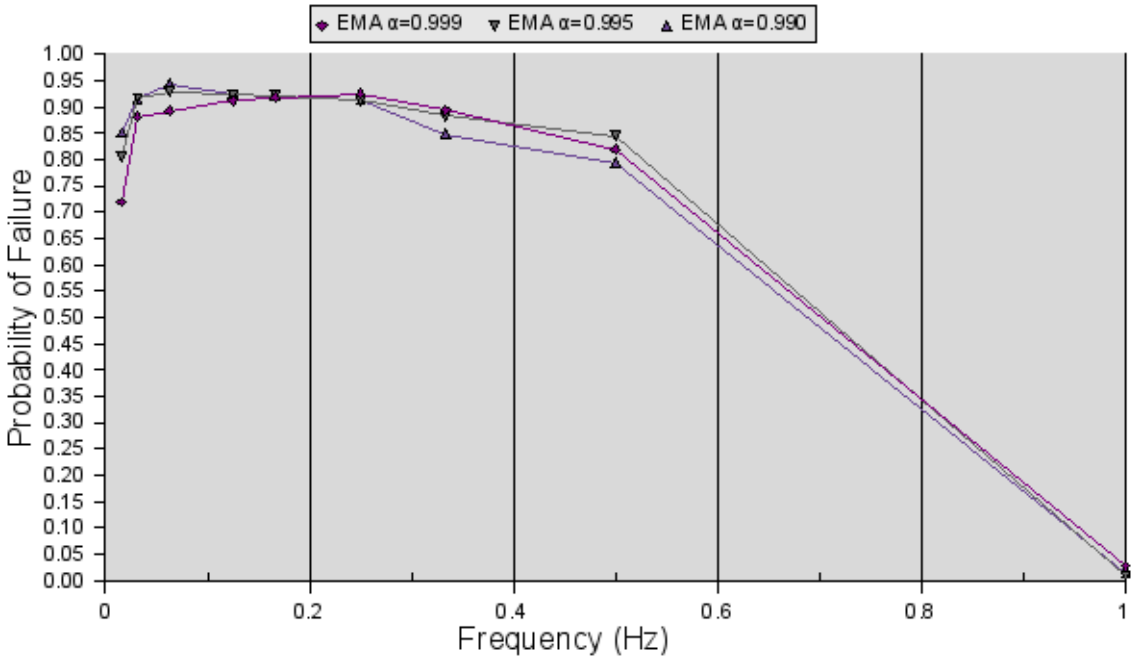

Figure 9: IPv6 ADDRESS COOKIES - MODULATED ATTACKS

Each of the three α values has a similar performance, with the smaller values doing slightly worse. Experimentation showed that a value of 0.999 performed well in other tests, and it continues to do better overall than the others, with a maximum failure rate near 93%. The good news is, at this attack rate (of 512 packets/second), none of these failure rates is greater than that resulting from an attack with no protection. The bad news is that many different modulation frequencies result in very high failure rates, even if they are not all optimal. It is also conceivable that failure rates could be pushed above that of the no-protection case with cer-

tain combinations of mixed *SID*s and modulated attacks. Modulation attacks are definitely the most serious threat to this system, but it may be possible to overcome this with *SID* remapping and more intelligent attack rate measurement methods.

8 Conclusions

Thus far, research into IPv6 address cookies is very promising. Attacks coming from a small number of locations are easily blocked, and as the complexity of an attack increases, the mitigation algorithms tend to degrade gracefully. The most serious threat to *SID* filtering is attack modulation, but it should be possible to fully mitigate this in small to medium-sized attacks. Other features of address cookies can contribute a partial solution to the traceback problem and allow an administrator to hide existence of delicate, authenticated services from unauthorized users. While more research and engineering effort are necessary before such a system can be usable in a real-world environment, the many potential benefits of address cookies make them attractive. This is especially true because address cookies do not require any major changes to networking equipment outside of the beneficiary's network.

Future work on address cookies should include an analysis of tradeoffs for *SID* remapping, the benefits of repeated referrals, and more research into optimal time-weighted averaging formulas. In addition, more testing in a realistic network environment should be conducted to bolster the case for address cookies, and potentially, IPv6 in general.

Appendix A: Stretchy Blowfish (version 0.1)

Background

Recent work by Cook, et. al. [18] in the development of block ciphers with variable block sizes inspired this work on Stretchy Blowfish. Stretchy Blowfish is a modification of Blowfish using this elastic technique to take an existing block cipher and expand its block size to arbitrary widths between the b and $2 \cdot b$, where b is the size (in bits) of the original algorithm's block.

Blowfish is a block cipher proposed by Bruce Schneier in 1993 [65], which has a block size of 64 bits, and key sizes that vary from 32 to 448 bits. No known attacks have been published which severely weaken Blowfish, but due to the relatively small block size, collision attacks are possible when encrypting more than 2^{32} bits of data. Blowfish was selected as an algorithm for extension because it is one of the few remaining 64-bit block ciphers which is unrestricted in use and has not been severely weakened through some public cryptanalytic attack.

Disclaimer: The cipher presented here has not been rigorously analyzed by qualified cryptographers. It is not safe to use in any real world systems at this time. It merely represents an educational exploration in cipher design, and if it is developed to the point of being a trustworthy cipher, it will likely be changed.

Round Adaptations

The Stretchy Blowfish cipher is designed to allow a user to select a block size between 64 and 128 bits, in steps of 8 bits. This 8-bit step limitation in block width was added to simplify implementation and the key schedule algorithms.

The process for building an elastic cipher based on a standard block cipher is as follows:

- The original width of the cipher, *b* in bits, is expanded by *y* bits.
- After each round of the cipher, which operates only on the *b* bits, these *y* bits are swapped into the block of *b* bits, and *y* bits from the block are swapped out. Each round, *y* bits sit out the round. If the cipher is based on a Feistel network, then one round in the elastic cipher is two rounds of the original cipher. In other words, the bits are only swapped out every other round.
- The number of rounds in the elastic cipher, *r'*, is calculated by: $r' = \left\lceil r \cdot \frac{b+y}{b} \right\rceil$, where *r* is the number of rounds in the original cipher. This guarantees that each bit in the elastic

cipher is operated on the same number of times it was in the original cipher. If working with a Feistel cipher, the number of rounds in the original cipher is considered as $\frac{1}{2}$ of the original, and *r'* is calculated based on that.

- In each round, the *y* added bits are XORed with some key material.
- If the original cipher does not have a whitening step at the beginning and end, these are added, along with an optional permutation.

The specific application of this approach for Stretchy Blowfish will now be described. Since Blowfish is a Feistel cipher, we see it as having 8 symmetric rounds, each having 2 of the originally defined rounds. The new number of symmetric rounds is calculated by:

$$r' = \left\lceil 8 \cdot \frac{64 + y}{64} \right\rceil + 1$$

or, since *y* is always a multiple of 8:

$$r' = 9 + \frac{y}{8}$$

And the following table describes the number of rounds based on block size:

Block Size	# of Sym. Rounds	# of Asym. Rounds
64	9	18
72	10	20
80	11	22
88	12	24
96	13	26
104	14	28
112	15	30
120	16	32
128	17	34

Table 2: STRETCHY BLOWFISH - ADDITIONAL ROUNDS

The purpose of the extra symmetric round is to eliminate the need for the initial whitening and mixing. In [18], it is mentioned that the first round of an elastic cipher is vulnerable to a differential attack, since the extra *y* bits do not participate in the first round. The authors suggest either a fast key-dependent permutation be done at the beginning, or an additional

round be added. Since Blowfish's round function is very fast, it makes more sense to add a round for simplicity's sake. In this case, a full symmetric round is being added, which should be sufficient to take care of the needed permutation as well as the whitening.

First, let us define some variables:

There exist four S-boxes, all generated randomly in the key-schedule (see below), which map 8-bit inputs to 32-bit outputs. Let *F* be a function from 32 bits to 32 bits, which is identical to Blowfish's *F* function based on these S-boxes. In addition, there is a variable-sized array of key material used as round keys. The total amount of key material needed, based on block size, is described in the next section. This key material is stored at the beginning of a buffer, B. There are $2 \cdot r' + 2$ 32-bit sub-keys labeled P_1 through $P_{(2 \cdot r' + 2)}$. In addition, there are $r' + 1$ sub-keys of length y labeled K_1 through $K_{r'+1}$. The 32-bit sub-keys are be pulled sequentially from the beginning of *B*, with the variable-width ones being chosen from the portion of *B* directly following the 32-bit keys. In other words, a 32-bit sub-key P_j is located at offset *j* in *B*, and a *y*-bit sub-key K_j would be located at offset $(2 \cdot r' + 2) + y \cdot j$ in *B*.

Let the first 64 bits of the input block be labeled *Z*. Split *Z* bits into two 32-bit halves. The first half will be named *L*, and the second half *R*. The remaining *y* bits will be referred to as *Y*. Also, let $Z[i]$ be the *y* bits within *Z* starting at bit offset $(8 \cdot i)$, wrapping around to the beginning of *Z,* if necessary (e.g., if $y = 16$ and $i = 7$, $Z[i]$ would represent the last byte of *Z* concatenated with the first byte of *Z*). Finally, *T* and *k* are temporary variables.

The algorithm for Stretchy Blowfish is:

Algorithm 5 STRETCHY BLOWFISH ENCRYPTION ROUNDS

For $i = 1$ to r':
$R \leftarrow R \oplus F(L \oplus P_{(2 \cdot i - 1)})$

$L \leftarrow L \oplus F(R \oplus P_{(2 \cdot i)})$
$Y \leftarrow Y \oplus K_i$
$k \leftarrow 5 \cdot (i - 1)\ mod\ 8$
$T \leftarrow Z[k]$
$Z[k] \leftarrow Z[k] \oplus Y$
$Y \leftarrow T$

$L \leftarrow L \oplus P_{(2 \cdot r' + 1)}$
$R \leftarrow R \oplus P_{(2 \cdot r' + 2)}$
$Y \leftarrow Y \oplus K_{(r'+1)}$

Note that in the case where the block size is 64, y is 0 and the operations on Y can be skipped. The cipher, in this case, is almost the same as Blowfish except for the additional symmetric round, and the change in key schedule.

New Key Schedule

Presented here is a possible adaptation to the Blowfish key schedule, which can be used to generate enough key material for all of the changes to the rounds. It is necessary to generate $(2 \cdot r' + 2)$ 32-bit keys, and $(r' + 1)$ y-bit keys. Therefore, the total amount of key material (excluding the S-boxes), b_k in bytes, is:

$$b_k = 4 \cdot (2 \cdot r' + 2) + (\frac{y}{8}) \cdot (r' + 1)$$

or, based solely on y:

$$b_k = 80 + y + \frac{5 \cdot y}{4} + (\frac{y}{8})^2$$

$$b_k = \frac{1}{64} \cdot y^2 + \frac{9}{4} \cdot y + 80$$

The table below describes this function for each possible block size:

Block Size	# of Sym. Rounds	b_k
64	9	80
72	10	99
80	11	120
88	12	143
96	13	168
104	14	195
112	15	224
120	16	255
128	17	288

Table 3: STRETCHY BLOWFISH SUB-KEY MATERIAL

While this function grows quickly with y, the largest size is still significantly smaller than the effort required to generate the S-boxes themselves under the original Blowfish.

The first part of the key schedule generation is very similar to that of the original Blowfish. It starts with a buffer, B, of size $(288 + 4096 = 4384)$ bytes which is filled with the hexadecimal digits of π. Also, a temporary 64-bit buffer X is set to 0. The first 32 bits of this buffer are referred to as X_L and the second 32 bits as X_R. The notation $Encrypt64(B, X)$ means to encrypt the block X with the 64-bit version of Stretchy Blowfish, using the first 80 bytes of B as the 32-bit sub-keys, and the last 4096 bytes as the S-boxes. Finally, $B[j]$ represents the 32-bit value stored at the j-th bit offset of B. The algorithm is as follows:

Algorithm 6 STRETCHY BLOWFISH SUB-KEY GENERATION

Sequentially XOR the user key into the beginning of B, wrapping around at 448-bits.
For $i = 1$ to 36:

$X \leftarrow Encrypt64(B, X)$
$B[64 \cdot i] \leftarrow X_R$
$B[64 \cdot i + 32] \leftarrow X_L$

Once the first 288 bytes of B are filled, the random S-boxes also need to be generated. Let $S_{k,l}$ be the set of 32-bit values in B from $B[k]$ to $B[l - 1]$, inclusive. This part deviates a bit more from the original Blowfish algorithm:

Algorithm 7 STRETCHY BLOWFISH S-BOX GENERATION

Let X retain its last value from the sub-key generation algorithm.
$c \leftarrow 2304$
For $i = 1$ to 512:

$X \leftarrow Encrypt64(B, X)$
$k \leftarrow c + 128 \cdot \lfloor \frac{i-1}{128} \rfloor$
$l \leftarrow c + 64 \cdot i$
$j \leftarrow 20$
While ($X_L = X_R$, or $X_L \epsilon S_{k,l}$, or $X_R \epsilon S_{k,l}$):
    $X_R \leftarrow X_R \oplus F(X_L \oplus B[32 \cdot j])$
    $X_L \leftarrow X_L \oplus F(X_R \oplus B[32 \cdot j + 32])$
    $j \leftarrow (j + 2)\ mod\ 72$
$B[l] \leftarrow X_R$
$B[l + 32] \leftarrow X_L$

When this is finished, calculate the value of b_k (the number of bytes needed for the sub-keys based on y). If b_k is less than 288, discard the portion of B after the first b_k bytes up until the beginning of the first S-box. The resulting $b_k + 4096$ bytes can then be used as B for the main

cipher.

Reasoning

Stretchy Blowfish's key schedule algorithm is significantly different than Blowfish's for several reasons. For one, more key material needs to be generated for the additional rounds and for the whitening of y. Beyond this, two other changes were introduced as improvements. One of the more successful (publicly known) attacks on Blowfish was done by Vaudenay [81], which showed that duplicate entries in an S-box leaves the cipher open to certain attacks. To avoid this, Stretchy Blowfish checks for duplicate entries and does additional rounds of encryption if needed. Secondly, an observation by Schmidt [64] that the third and fourth sub-keys are independent of the first 64-bits of user key prompted the change in sub-key assignment. In the original Blowfish's key schedule, X_L was assigned to the lower numbered sub-key after each encryption. In the second encryption, this causes a cancellation which results in less diffusion of the original key data. The suggested fix, which is implemented in Stretchy Blowfish, is to swap the order of sub-key assignment.

While these changes to the key schedule are meant to improve the security of the cipher, they very well may have introduced some serious vulnerability. Because the S-boxes are generated in the key schedule, nearly all the security of the cipher depends on it. There are two potential flaws we have identified which arise because of these changes. First, the S-box collision avoidance checks introduce a variable amount of computation, and in instances where an attacker can measure the key generation time, this could lead to a timing attack. Fortunately, this can be avoided with a careful implementation.

Secondly, a weakness may have been introduced in the way later sub-keys are generated. The original Blowfish's key schedule algorithm acts as a pseudorandom number generator with a very large state. Each time new data is generated, it is fed back into the state for future rounds. Every new block generated is influenced by both the large B buffer, as well as the temporary value of X. However, in the proposed key schedule for Stretchy Blowfish, the blocks generated after the first 80 bytes are no longer used to generate new blocks. Therefore, the random number generator essentially drops from one with over 4 kilobytes of state, to one with just 64 bits of changing state. This continues until the rest of the sub-keys are generated. When the algorithm begins generating the S-boxes, the generator is once again using all of its state. Therefore, in the cases where most of the later sub-keys are used (with large blocks), there may be an attack to exploit this reduced state. Hopefully though, with the improved

diffusion and the fact that all of the users' key is diffused after just 56 bytes, the algorithm will hold up under scrutiny.

Performance

Blowfish's key schedule has always been its Achilles heel in terms of performance and memory requirements, and Stretchy Blowfish is no different. In fact, the additional symmetric round of Stretchy Blowfish used in the key generation process adds a 12.5% computation overhead, and the S-box collision avoidance adds some additional iteration. Fortunately, collisions in S-boxes are somewhat rare, so the additional rounds used for this will not impact performance a great deal. The additional storage needs are not excessive, requiring between 0.2% and 5.2% more memory after the sub-keys and S-boxes are generated.

It seems that the encryption and decryption operations are not heavily impacted by the additional rounds, proportional to the amount of data being encrypted. For instance, in the original Blowfish, there were two rounds of of encryption per byte. In Stretchy Blowfish, there are between 2.125 and 2.25 rounds per byte, depending on the specific block size.

References

[1] Aiello, W., Bellovin, S. M., Blaze, M., Canetti, R., Ioannidis, J., Keromytis, A. D., and Reingold, O. Just Fast Keying: Key Agreement in a Hostile Internet. http://citeseer.ist.psu.edu/aiello04just.html. ACM Transactions on Information and System Security, volume 7, number 2, pages 1-30. May, 2004.

[2] Al-Duwairi, B. N. *Mitigation and Traceback Countermeasures for DDoS Attacks*. http://archives.ece.iastate.edu/archive/00000149/01/thesis.pdf. Completed as part of a doctorate dissertation at Iowa State University. 2005.

[3] Antirez. *New tcp scan method*. http://seclists.org/lists/bugtraq/1998/Dec/0079.html. Posted on Bugtraq: 1998-12-19. Accessed: 2006-01-03.

[4] Arbor Networks, Inc. *Worldwide ISP Security Report*. http://www.arbor.net/downloads/Arbor_Worldwide_ISP_Security_Report.pdf. Arbor Networks, Inc. September, 2005.

[5] Arkko, J. (editor), et. al. *RFC 3971: SEcure Neighbor Discovery (SEND)*. Network Working Group of the IETF. http://www.ietf.org/rfc/rfc3971.txt. March 2005.

[6] Aura, T. *RFC 3972: Cryptographically Generated Addresses (CGA)*. Network Working Group of the IETF. http://www.ietf.org/rfc/rfc3972.txt. March, 2005.

[7] Aura, T., Nikander, P., and Leiwo, J. *DOS-Resistant Authentication with Client Puzzles*. http://www.securitytechnet.com/resource/security/hacking/DoS/aura-nikander-leiwo-protocols00.pdf. Security Protocols: 8th International Workshops. Springer Berlin/Heidelberg, Lecture Notes in Computer Science, vol 2133. 2001.

[8] AusCERT. *AL-1999.004: Denial of Service (DoS) attacks using the Domain Name System (DNS)*. http://www.auscert.org.au/render.html?it=80&template=1. AusCert. August 13, 1999.

[9] Azar, Y., Broder, A. Z., Karlin, A. R., and Upfal, E. *Balanced Allocations*. http://epubs.siam.org/SICOMP/volume-29/art_28849.html. SICOMP, Volume 29 Issue 1 pages 180-200. Society for Industrial and Applied Mathematics. 1999.

[10] Bencsáth, B., Vajda, I., and Buttyán, L. *A Game Based Analysis of the Client Puzzle Approach to Defend Against DoS Attacks.* http://www.hit.bme.hu/~buttyan/publications/BencsathVB03softcom.pdf. Proceedings of the 2003 International Conference on Software, Telecommunications and Computer Networks (SoftCOM'03). 2003.

[11] Bellovin, S. M., Cheswick, and B., Keromytis, A. D. *Worm Propagation Strategies in an IPv6 Internet.* USENIX ;LOGIN:, Vol. 31, No. 1. http://www.usenix.org/publications/login/2006-02/pdfs/bellovin.pdf. February 2006.

[12] Bernstein, D. J. *SYN Cookies*. http://cr.yp.to/syncookies.html. Accessed: 2005-07-10.

[13] Bridis, T. *Computer Researchers Warn of Net Attacks*. http://news.yahoo.com/s/ap/20060316/ap_on_hi_te/internet_attack. The Associated Press. Mar 16, 2006.

[14] Carpenter, B. and Moore, K. *RFC 3056: Connection of IPv6 Domains via IPv4 Clouds*. http://www.ietf.org/rfc/rfc3056.txt. Network Working Group of the IETF. The Internet Society. February, 2001.

[15] Chown, T. *Internet Draft: IPv6 Implications for TCP/UDP Port Scanning.* http://www.ietf.org/internet-drafts/draft-chown-v6ops-port-scanning-implications-02.txt. IPv6 Operations of the IETF. The Internet Society. October 27, 2005. Expires: April 30, 2006.

[16] Chun, B. G., Fonseca, R. and Mehra, P. *DAM: A DoS Attack Mitigation Infrastructure.* http://www.cs.berkeley.edu/~daw/teaching/cs261-f02/reports/chun.pdf. Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. 2003.

[17] Cook, D. L., Yung, M. and Keromytis, A. D. *Elastic Block Ciphers.* http://www1.cs.columbia.edu/~dcook/pubs/ealg062804.pdf. Cryptology eprint Archive Report 2004/128, May 2004.

[18] Cook, D. L., Yung, M. and Keromytis, A. D. *Elastic Block Ciphers: The Feistel Cipher Case.* http://www1.cs.columbia.edu/~dcook/pubs/misty-TR-021_04.pdf. Computer Science Technical Report CUCS-021-04, Columbia University, May 2004.

[19] Cook, D. L., Yung, M. and Keromytis, A. D. *Elastic AES*. http://www1.cs.columbia.edu/~dcook/pubs/eaes070504.pdf. Cryptology eprint Archive Report 2004/141, June 2004.

[20] Cook, D. L., Yung, M. and Keromytis, A. D. *On Elastic Block Ciphers and Their Differential and Linear Cryptanalyses*. http://mice.cs.columbia.edu/getTechreport.php?techreportID=361. Computer Science Technical Report CUCS 039-05, Columbia University, September 2005.

[21] Cook, D. L., Yung, M. and Keromytis, A. D. *A General Analysis of the Security of Elastic Block Ciphers*. http://mice.cs.columbia.edu/getTechreport.php?techreportID=360. Computer Science Technical Report CUCS 038-05, Columbia University, September 2005.

[22] Cook, D. L., Yung, M. and Keromytis, A. D. *The Pseudorandomness of Elastic Block Ciphers*. http://mice.cs.columbia.edu/getTechreport.php?techreportID=359. Computer Science Technical Report CUCS 037-05, Columbia University, September 2005.

[23] CERT Coordination Center. *CERT® Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks*. http://www.cert.org/advisories/CA-1996-21.html. Last revised: November 29, 2000. Carnegie Mellon University. September 19, 1996.

[24] CERT Coordination Center. *CERT® Advisory CA-1997-28 IP Denial-of-Service Attacks*. http://www.cert.org/advisories/CA-1997-28.html. Carnegie Mellon University. First published: December 16, 1997. Last updated: May 26, 1998.

[25] CERT Coordination Center. *CERT® Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks*. http://www.cert.org/advisories/CA-1998-01.html. Last revised: March 13, 2000. Carnegie Mellon University. 1998.

[26] Dean, D. and Stubblefield, A. *Using Client Puzzles to Protect TLS*. http://www.usenix.org/events/sec01/full_papers/dean/dean.pdf. Proceedings of the 10th USENIX Security Symposium. 2001.

[27] Digital Lumber, Inc. *Oak DNS Server*. http://www.digitallumber.net/oak. 2002.

[28] Douligeris, C. and Mitrokotsa, A. *DDoS attacks and defense mechanisms: classification and state-of-the-art*. http://dx.doi.org/10.1016/j.comnet.2003.10.003. Computer Networks: The International Journal of Computer and Telecommunications Networking, volume 44, issue 5, pages: 643 - 666. April, 2004.

[29] Ferguson, P. and Senie, D. *RFC 2827: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing.* http://www.ietf.org/rfc/rfc2827.txt. Network Working Group of the IETF. The Internet Society. May, 2000.

[30] Fyodor. *Idle Scanning and related IPID games.* http://www.insecure.org/nmap/idlescan.html. Accessed: 2006-01-03.

[31] Gibson, S. *Distributed Reflected Denial of Service: Description and Analysis of a Potent, Increasingly Prevalent, and Worrisome Internet Attack.* http://grc.com/files/drdos.pdf. Gibson Research Corporation. February 22, 2002.

[32] Gleitz, P. and Bellovin, S. M. *Transient Addressing for Related Processes: Improved Firewalling by Using IPV6 and Multiple Addresses per Host.* http://www.usenix.org/events/sec01/full_papers/gleitz/gleitz_html/. Proceedings of the 10th USENIX Security Symposium. 2001.

[33] Hagen, S. *IPv6 Essentials.* O'Reilly & Associates, Inc., Sebastopol, CA. 2002.

[34] Handley, M. (ed.) and Rescorla, E. (ed). *Internet Draft: Internet Denial of Service Considerations.* http://www.ietf.org/internet-drafts/draft-iab-dos-03.txt. Network Working Group of the IETF. September 15, 2005. Expires: March 19, 2006.

[35] Hollis, K. *The Rose Fragmentation Attack Explained.* http://digital.net/~gandalf/Rose_Frag_Attack_Explained.htm. First published: April, 2004. Accessed: 2006-03-06.

[36] The Honeynet Project & Research Alliance. *Know your Enemy: Tracking Botnets.* http://www.honeynet.org/papers/bots/. Last Modified: March 13, 2005.

[37] Hussain, A., Heidemann, J., and C. Papadopoulos. *A Framework for Classifying Denial of Service Attacks.* http://www.isi.edu/~johnh/PAPERS/Hussain03a.html. Proceedings of ACM SIGCOMM 2003.

[38] Internet Engineering Steering Group. *RFC 3177: IAB/IESG Recommendations on IPv6 Address Allocations to Sites*. Network Working Group of the IETF. http://www.ietf.org/rfc/rfc3177.txt. September, 2001.

[39] Juels, A. and Brainard, J. *Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks.* http://www.rsasecurity.com/rsalabs/staff/bios/ajuels/publications/client-puzzles/clientpuzzles.ps. Proceedings of NDSS '99 (Networks and Distributed Security Systems), pages 151-165, 1999.

[40] Kamra, A., Feng, H., Misra, V., and Keromytis, A. D. *The Effect of DNS Delays on Worm Propagation in an IPv6 Internet.* http://www1.cs.columbia.edu/~angelos/Papers/2005/dns-worm.pdf. 2005.

[41] Keromytis, A.D., Misra, V., and Rubenstein, D. *SOS: an architecture for mitigating DDoS attacks.* http://www1.cs.columbia.edu/~angelos/Papers/jsac-sos.pdf. Selected Areas in Communications, IEEE Journal on Volume 22, Issue 1, pages 176 - 188. January, 2004.

[42] Kohler, E., et. al. *Click Modular Router Project.* http://www.read.cs.ucla.edu/click/. 1999-2006.

[43] Krawczyk, H., Bellare, M., and Canetti, R. *RFC 2104: HMAC: Keyed-Hashing for Message Authentication.* http://www.ietf.org/rfc/rfc2104.txt. Network Working Group of the IETF. February 1997.

[44] Krzywinski, M. *Port Knocking: Network Authentication Across Closed Ports.* http://www.portknocking.org/. SysAdmin Magazine 12: 12-17. 2003.

[45] Lawson, Stephen. *Cisco touts core router customers, smaller platform.* http://www.computerworld.com.au/index.php/id;740687715;fp;16;fpid;0. IDG Communications. 2004-07-12. Accessed: 2005-07-10.

[46] Li, J., Mirkovic, J., Wang, M., Reiher, P. and Zhang, L. *SAVE: Source Address Validity Enforcement Protocol.* http://lasr.cs.ucla.edu/save/save_to_infocom.pdf. Proceedings of IEEE INFOCOM 2002.

[47] Ludvig, M. *VIA PadLock - Wicked Fast Encryption.* http://www.linuxjournal.com/article/8042. April 6, 2005.

[48] MacDermid, T. *Out of Band #2 - Loose Source Routing Blues.* http://www.synacklabs.net/OOB/LSR.html. SYN ACK Labs. March 2, 2002. Accessed: 2006-03-10.

[49] Mirkovic, J., Martin, J., and Reiher, P. *A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms*. http://lasr.cs.ucla.edu/ddos/ucla_tech_report_020018.pdf. University of California, Los Angeles, Computer Science Department Technical Report #020018. 2001.

[50] Mockapetris, P. *RFC 1034: Domain Names - Concepts and Facilities*. http://www.ietf.org/rfc/rfc1034.txt. Network Working Group of the IETF. The Internet Society. November, 1987.

[51] Montenegro, G. and Castelluccia, C. *Crypto-based identifiers (CBIDs): Concepts and applications*. http://portal.acm.org/citation.cfm?id=984338. ACM Transactions on Information and System Security (TISSEC), Volume 7 , Issue 1, Pages: 97 - 127. February, 2004.

[52] Moore, D., Shannon, C., Brown, D., Voelker, G. M., and Savage, S. *Inferring Internet Denial-of-Service Activity*. http://www.caida.org/publications/papers/2006/backscatter_dos/backscatter_dos.pdf. IEEE/ACM Transactions on Networking, Jan 2006.

[53] Moore, D., Voelker, G. M., and Savage, S. *Inferring Internet Denial-of-Service Activity*. http://www.caida.org/publications/papers/2001/BackScatter/usenixsecurity01.pdf. Usenix Security Symposium, 2001.

[54] The Measurement Factory. *DNS SURVEY: JUNE 2005*. http://dns.measurement-factory.com/surveys/200506.html. Accessed 2006-02-13.

[55] Ohsita, Y., Ata, S., and Murata, M. *Deployable Overlay Network for Defense Against SYN Flood Attacks*. http://www.nal.ics.es.osaka-u.ac.jp/~y-ohsita/papers/ICCCN2005.pdf. Proceedings of the 14th International Conference on Computer Communications and Networks (ICCCN), 2005. October, 2005.

[56] O'Shea, G. and Roe, M. *Child-proof authentication for MIPv6 (CAM)*. http://portal.acm.org/citation.cfm?id=505668. ACM SIGCOMM Computer Communication Review, Volume 31 , Issue 2, Pages: 4 - 8. April, 2001.

[57] Postel, J. (ed). *RFC 791: Internet Protocol*. http://www.ietf.org/rfc/rfc791.txt. Prepared for DARPA by Information Sciences Institute. September, 1981.

[58] Postel, J. *RFC 862: Echo Protocol.* http://www.ietf.org/rfc/rfc862.txt. Network Working Group of the IETF. May, 1983.

[59] Ptacek, T. H. and Newsham, T. N. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection* http://www.insecure.org/stf/secnet_ids/secnet_ids.html. Secure Networks, Inc. January, 1998.

[60] Rajahalme, J., Conta, A., Carpenter, B., and Deering, S. *RFC 3697: IPv6 Flow Label Specification.* http://www.ietf.org/rfc/rfc3697.txt. Network Working Group of the IETF. The Internet Society. March, 2004.

[61] Ritter, T. *A Variable Size Core for Block Cipher Cryptography: Cipher Blocks of Arbitrary Size with Good Data Diffusion.* http://www.ciphersbyritter.com/VSBCCORE.HTM. Last updated: 1998-03-21. Accessed: 2006-03-11.

[62] Savage, S., Wetherall, D., Karlin, A. R. and Anderson, T. *Practical Network Support for IP Traceback.* http://www.cs.ucsd.edu/~savage/papers/UW-CSE-00-02-01.pdf. Proceedings of the 2000 ACM SIGCOMM Conference, August 2000.

[63] Savola, P. and Patel, C. *RFC 3964: Security Considerations for 6to4.* http://www.ietf.org/rfc/rfc3964.txt. Network Working Group of the IETF. The Internet Society. December, 2004.

[64] Schmidt, D. *On the Key Schedule of Blowfish.* http://eprint.iacr.org/2005/063. Cryptology ePrint Archive, Report 2005/063. 2005.

[65] Schneier, B. *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish).* http://www.schneier.com/paper-blowfish-fse.html. Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.

[66] Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* Second Edition. John Wiley & Sons, Inc. 1996.

[67] Shannon, C.E. *Communication theory of secrecy systems.* http://www.cs.ucla.edu/~jkong/research/security/shannon1949.pdf. Bell System Technical Journal, 28(4):656–715, 1949.

[68] Snoeren, A. C., Partridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Kent, S. T., and Strayer, W. T. *Hash-Based IP Traceback.* http://www.acm.org/sigs/sigcomm/sigcomm2001/p1-snoeren.pdf. Association for Computing Machinery, SIGCOMM'01. August, 2001.

[69] Spitzner, L. *Honeytokens: The Other Honeypot.* http://www.securityfocus.com/infocus/1713. SecurityFocus. July 17, 2003.

[70] Stavrou, A., Cook, D. L., Morein, W. G., Keromytis, A. D., Misra, V., Rubenstein, D. *WebSOS: An Overlay-based System For Protecting Web Servers From Denial of Service Attacks.* http://www1.cs.columbia.edu/~danr/publish/2005/Stavrou:JCN05.pdf. Journal of Communication Networks, Volume 48, Number 5, August 2005.

[71] Stephen, J. *The Changing Face of Distributed Denial of Service Mitigation.* http://www.sans.org/rr/whitepapers/threats/462.php. SANS Information Security Reading Room. August 16, 2001.

[72] Stewart, J. *DNS Cache Poisoning - The Next Generation.* http://www.lurhq.com/cachepoisoning.html. LURHQ Threat Intelligence Group. 2003.

[73] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and Paxson, V. *RFC 2960: Stream Control Transmission Protocol.* http://www.ietf.org/rfc/rfc2960.txt. Network Working Group of the IETF. The Internet Society. October, 2000.

[74] Stone, R. *CenterTrack: An IP Overlay Network for Tracking DoS Floods.* http://www.usenix.org/publications/library/proceedings/sec2000/full_papers/stone/stone.pdf. Proceedings of the 9th USENIX Security Symposium. 2000.

[75] Sundaram, R. and Milliken, W. C. *Systems and Methods for Network Attack Detection and Prevention through Redirection.* US Patent Application No. 10/826,897. Conception Date: March 20, 2003.

[76] Thomer, G. and Poletto, M. *MULTOPS: A Data Structure for Bandwidth Attack Detection.* http://www.usenix.org/events/sec01/full_papers/gil/gil.pdf. Proceedings of the 10th USENIX Security Symposium. 2001.

[77] Thomson, S., Huitema, C., Ksinant, V., and Souissi, M. *RFC 3596: DNS Extensions to Support IP Version 6*. http://www.ietf.org/rfc/rfc3596.txt. Network Working Group of the IETF. The Internet Society. October, 2003.

[78] Thorpe, H. E. *Security Analysis of an Encrypted Authentication Cookie*. http://henry.thorpe.home.att.net/w4995/auth_cookie.ps. April 30, 2001.

[79] Trenholme, S. *The Kiwi Spam-Protection Package: Intro*. http://kiwispam.sourceforge.net/Intro.html. Published: circa October 17, 2001. Accessed: 2006-04-10.

[80] US-CERT. *The Continuing Denial of Service Threat Posed by DNS Recursion*. http://www.us-cert.gov/reading_room/DNS-recursion121605.pdf. US-CERT. 2005.

[81] Vaudenay, S. *On the Weak Keys of Blowfish*. http://citeseer.ist.psu.edu/vaudenay95weak.html. Fast Software Encryption, pages 27-32. 1996.

[82] Vaughn, R. and Evron, G. *DNS Amplification Attacks (Preliminary Release)*. http://www.isotf.org/news/DNS-Amplification-Attacks.pdf. March 17, 2006.

[83] Wikipedia (multiple authors). *Zero Day*. http://en.wikipedia.org/wiki/Zero_day. Version updated: April 14, 2006.

[84] Worth, D. *COK: Cryptographic One-Time Knocking*. http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-worth-up.pdf. A presentation given at Black Hat USA, 2004.

[85] Xuan, D., Chellappan, S., Wang, X., and Wang, S. *Analyzing the secure overlay services architecture under intelligent DDoS attacks*. http://www.cse.ohio-state.edu/~xuan/papers/04_icdcs_xcww.ps. Proceedings of the 24th International Conference on Distributed Computing Systems, pages: 408 - 417. 2004.