LIPS: Lightweight Internet Permit System for Stopping Unwanted Packets

Changho Choi[1], Yingfei Dong[2], and Zhi-Li Zhang[1, ⋆]

[1] Dept. of Computer Science, Univ. of Minnesota, Minneapolis, MN 55455
choi,zhzhang@cs.umn.edu
[2] Dept. of Electrical Engineering, Univ. of Hawaii, 2540 Dole St., Honolulu, HI 96822
TEL:01-808-956-3448, FAX: 01-808-956-3427
yingfei@hawaii.edu

Abstract. In this paper, we propose a *Lightweight Internet Permit System (LIPS)* that provides a lightweight, scalable packet authentication mechanism for ensuring traffic-origin accountability. LIPS is a simple extension of IP, in which each packet carries an *access permit* issued by its destination host or gateway, and the destination verifies the access permit to determine if a packet is accepted or dropped. We will first present the design and the prototype implementation of LIPS on Linux 2.4 kernel. We then use analysis, simulations, and experiments to show how LIPS can effectively prevent protected critical servers and links from being flooded by unwanted packets with negligible overheads. We propose LIPS as an domain-to-domain approach to stop unwanted attacks, without requiring broad changes in backbone networks as other approaches. Therefore, LIPS is incrementally deployable in a large scale on common platforms with minor software patches.

Keywords: Network Security, IP Spoofing, Denial of Service, Unwanted Packets.

1 Introduction

One of the key security issues in the current Internet is that a source IP address can be easily spoofed and manipulated, and *unwanted* packets can intrude an unwary host with ease (despite firewalls), which is often tricked into unintentional "accomplice" (e.g., in the case of viruses and worms), spreading attacks to many other vulnerable hosts. To combat this problem, many organizations choose to "close off" their networks via mechanisms such as VPNs or employ firewalls to block certain types of packets (e.g., based on IP addresses, ports, or packet payload), regardless of senders and their intent. Clearly, such solutions are fairly limited in their scope or effectiveness as email viruses and worms

⋆ This work was supported in part by the National Science Foundation (NSF)under the grants ITR-0085824 and CNS-0435444.

can routinely penetrate firewalls. Furthermore, they are rather *rigid*, sometimes breaking existing applications and potentially impeding creation and deployment of new services and applications. There is still much debate in the networking research community regarding how to secure and fortify the current Internet while without jeopardizing its open architecture and end-to-end design principle.

In this paper, we propose a novel *lightweight Internet permit system (LIPS)* to provide traffic accountability through fast packet authentication for stopping unwanted packets. By unwanted packets, we mean packets *not* intended for "normal" communications between hosts, such as packets with spoofed IP addresses, generated in port scanning or worm spreading. Such packets account for, or are forerunners of, most of unauthorized accesses, intrusion, disruption, denial-of-service (DoS) attacks and other cyber threats in today's Internet. LIPS is designed as an efficient traffic authentication mechanism to filter out most of these illegitimate packets, with minor changes to current systems and negligible overheads. We implement LIPS as a small patch to the IP layer at a LIPS-aware host. When a source wants to communicate with a destination, it first requests and obtains (if granted) an *access permit* from the destination. It then inserts a destination access permit into each packet sent to the destination. Only packets with proper access permits will be accepted at the destination. This simple architecture provides a scalable and flexible framework for establishing *traffic accountability* among networks and hosts, and for securing Internet resources *without* sacrificing their open and dynamic nature. Furthermore, LIPS also simplifies and facilitates the early detection of, and timely protection from, network intrusion and attacks by requiring valid access permits before any data packets can be accepted and processed. Hence by incorporating *active monitoring* and *rapid response* mechanisms into LIPS, we can build an effective and scalable "first-line" defense to protect Internet resources from unwanted traffic.

Related Works. Many security mechanisms have been proposed to control the damage of (DDoS) attacks and trace back attacking sources. Pushback [7] treats DDoS as a congestion-control problem and requires each router to detect and preferentially drop packets that probably belong to an attack. Upstream routers are also notified to drop such packets in order that the routers resources are used to route legitimate traffic. The network capability scheme [1] inserts special tokens into packets and use routers to check these tokens along forwarding paths for restricting unwanted packets. While these routers authenticate packets and maintain *per-flow states*, destination hosts also keep *per-flow states* for authentication using hash chains. Furthermore, overlay approaches (SOS and Mayday) [5] use a *wide-area* overlay infrastructure with a *large number* of intermediate nodes to filter out attacking traffic. IP Easy-Pass [8] aims to protect real-time priority traffic Denial-of-QoS attacks at an ISP edge router by maintaining *per-flow states*. In addition, the visa protocols [3] use encryption and data signatures to authenticate a flow of packets. They require a shared key to be established between access control servers on a *per-source-destination* basis. Similarly, IPsec and VPNs establish shared keys to secure end-to-end communications with known overheads [4]. Several traceback schemes were proposed to track down attacking sources, such as IP traceback. These schemes usually require to modify intermediate routers along packet forwarding paths and are mostly for post-attack analysis. To deal with IP-spoofing, ingress filters can not stop attackers to spoof in valid address space. However, the computational overhead of public key schemes limits host performance and makes them difficult to scale to large systems.

In summary, these approaches generally either incur high computational overheads and/or heavy key management costs, or require modification to intermediate routers or broad infrastructure support. On the contrary, the proposed LIPS does not use encryption or digital signatures, hence the overheads of encryption and key management are minimized. Furthermore, LIPS is an end-to-end/edge-to-edge approach that does not require any support from *intermediate* networks. Therefore, it is easier to be deployed incrementally to a large scale.

The remainder of this paper is organized as follows. In Section 2, we present the basic concepts and constructs of LIPS, illustrate how it works and why it is useful, and discuss the related work. In Section 3, we present the design and implementation of LIPS. In Section 4, we evaluate the performance of LIPS via simulations and experiments. We conclude this paper in Section 5.

2 Basics of LIPS Architecture

The idea of LIPS is simple: every LIPS packet carries an *access permit* issued by its destination, and this permit is verified at the destination to determine whether the packet is accepted or dropped. Hence for a source to send packets to a destination, it must obtain a valid access permit first. This simple mechanism enables the destination to easily eliminate illegitimate/spoofed packets and *control who has access to it.* As only data packets with valid access permits will be accepted and processed by applications running on a destination host. Thus a malicious host cannot simply inject unwanted traffic to harm a destination host without first requesting an access permit and identifying itself to the destination. In the following, we first introduce the basic components and operations of LIPS, and illustrate how access permits are generated, exchanged, and verified. We then discuss the advantages and limitations of LIPS at the end.

LIPS Packet. LIPS is a simple extension of the IP protocol. We convert an IP packet into a LIPS packet by inserting a LIPS header into the payload field of the IP packet and changing the protocol type to 138 in the IP header[1], as shown in Fig.1. (We choose 138 as the protocol type of LIPS in our prototype implementation.) The format of a LIPS packet header is given in Fig.2, which includes four control fields, a *destination access permit (DAP)*, and a *source access permit*

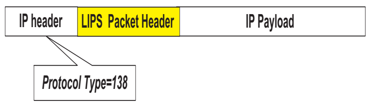


Fig. 1. A LIPS Packet

[1] To avoid the potential segmentation issue, we first perform a path MTU discovery and then set a proper MTU for the connection.

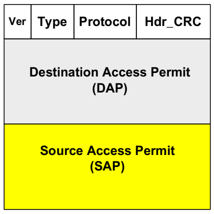


Fig. 2. LIPS Packet Header

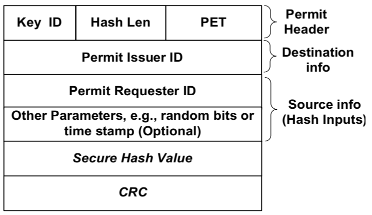


Fig. 3. LIPS Permit

(SAP). A DAP is issued by a destination to a source. It is carried in packets from the source to the destination and is verified at the destination. A SAP is issued by the source to the destination for packets on the return path. The *Ver* field holds a LIPS version number. The *Type* field specifies the type of a LIPS packet, such as a permit request, a permit reply, or a LIPS data packet. Since we replace the IP protocol type in the IP header to 138 when we translate an IP packet into a LIPS packet, we use the *Protocol* field to hold the protocol type in an original IP packet such that we can restore it after the LIPS packet is accepted at a destination. The *Hdr_CRC* field is a simple CRC for a LIPS packet header.

Access Permit. An access permit is constructed using *keyed* message authentication code (MAC) [6] at a LIPS-aware host. This MAC is generated through a secure hash function with two inputs: a *plain* hash message (chosen by a permit issuer and carried in an access permit in plain text) and a *secret* hash key held by the issuer.

As shown in Fig.3, an access permit includes five parts: a *permit header*, a *permit issuer's ID*, a *permit requester's ID* (plus optional parameters), a *secure hash value*, and a *CRC checksum* of the secure hash value. The permit header contains an index (*Key ID*) of a secret key used for this permit at its issuer, a hash length (*Hash Len*) that specifies the length of the secure hash value in this permit, and a permit expire time (*PET*) that defines the effective duration of this permit. The length of secure hash value can be adjusted from 64 bits to 128 bits depending on the permit issuer's security requirements. The source information (denoted as M) includes, e.g., the source IP address and several optional security parameters (e.g., a random number used to deal with permit-replay attacks). It is used by the issuer as the input plain hash message to a secure hash function to compute a message digest, $H(M, K_t)$, where $H()$ is a secure hash function, e.g., HMAC-MD5, and K_t is a secret key of the permit issuer at time t. For ease of exposition, we use the source IP address as M in the following presentation. Given that the hash length in the permit header is l, the secure hash value of a permit is the first l bits of the message digest. For example, we can choose the first 64 bits of a 128-bit HMAC-MD5 digest as a secure hash value. This hash value will be used for validating the permit. Note that the hash value is *specific* to the requester and valid only for a certain period of time, as K_t is changed over time. Without knowing the secret key, it is very difficult to forge a permit.

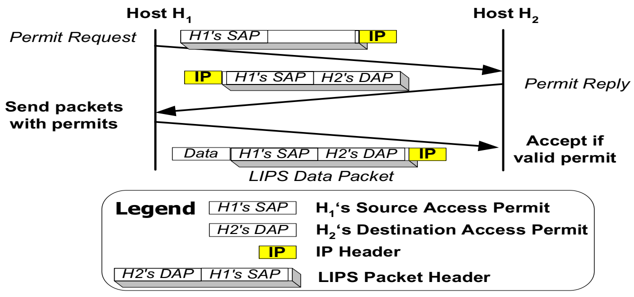


Fig. 4. Message Exchanges in Setting up an Access Permit

Exchange of Access Permits between a Source and a Destination. We use a simple example to illustrate how access permits are set up between two LIPS-aware hosts. As shown in Fig.4, when a source host H_1 wants to communicate with a destination host H_2, H_1 first sends a *permit request* to H_2. This request carries H_1's SAP in the request. The SAP contains a secure hash value generated based on a *secret* hash key of H_1 and a plain hash message about H_2 (e.g., H_2's IP address). *Note that not all hosts will be allowed to access* H_2. A *security policy* at H_2 is checked to determine if H_2 accepts this permit request[2]. If it does, it generates an access permit (H_2's DAP for H_1), containing a secure hash value generated based on a secret hash key of H_2 and a plain hash message about H_1 (e.g., H_1's IP address). Then H_2 sends the permit (as the SAP) in a *permit reply* message back to H_1, using H_1's SAP (attached in the permit request from H_1) as the DAP.

H_1 will only accept a permit reply that carries a valid DAP, namely, a SAP that it issues in an earlier permit request. This is done by computing a secure hash value using the plain hash message carried in the DAP and a secret key pointed by the key index. If this hash value matches the secure hash value carried in the DAP, H_1 accepts this reply and caches the SAP of the packet (i.e., H_2's DAP) into a *permit cache*. For the subsequent data packets sent to H_2, H_1 puts H_2's DAP and its own SAP into the LIPS headers of these packets. When H_2 receives a LIPS packet from H_1, it verifies the DAP of the packet and accepts it only if the DAP is valid.

Table 1. Permit Cache

Destination IP address	Flag	Destination Access Permit
172.10.10.1	1	xxxxxxxx
129.128.128.1	3	NULL

Permit Cache. Each LIPS host maintains a permit cache with a format shown as Table. 1. A cache entry contains a destination IP address, a *Flag*, and a destination access permit. For incremental deployment, the cache not only holds permits

[2] An example policy may be only accepting permit requests from a local domain, e.g., accepting packets from (secure) proxy servers when communicating with hosts in other domains. This will automatically eliminate port scanning and worm spreading packets from other domains, i.e., an attacker outside a domain cannot discover vulnerabilities via scanning and a worm cannot propagate across domains through random probing. Damages are localized.

for LIPS-aware hosts, but also tracks non-LIPS hosts (allowed by security policies), using the destination IP address as its primary index. The flag is used to distinguish the state of a cache entry: flag = 0, indicating that the entry is in initialization, namely, a permit request has been sent to the destination but the reply has not been received yet; flag = 1, a valid permit for the destination is available; flag = 2, the destination permit has expired; and flag = 3, the destination does not supports LIPS.

Key Management. Each LIPS host maintains a secret key pool of, say, 256 keys. Each key is uniquely identified by a key index. When a host generates an access permit, it randomly chooses a key from its key pool and records the key index in the *Key ID* field of a permit header. When a host verifies an access permit, it retrieves a key using the *Key ID* of the permit header. Note that in LIPS a key pool is *not shared* with any other hosts, and *no key exchange* among hosts is required, contrary to the complex key establishment procedures in other approaches. Hence the overhead of LIPS key management is minimized.

Why LIPS. We conclude this section by illustrating how LIPS can be used as a first-line defensive and preventive mechanism to protect hosts from unwanted traffic. First, LIPS introduces *traffic-origin accountability* into the Internet and enables destinations with the ability to deny access and stop unwanted traffic from untrusted hosts. Second and perhaps more importantly, *LIPS facilitates and simplifies the tasks of detecting unauthorized intrusion and attacks* by forcing malicious hosts to first requesting access permits and identifying themselves to the intended targets before launching an offense, as we illustrate below. Clearly, since a source must obtain a valid access permit before it can send a packet to a destination, illegitimate/spoofed packets will be automatically filtered out. Cyber attacks such as worms generally rely on port-scanning to identify vulnerable hosts, tricking them to execute the malcode carried in the payload, and thereby compromising them as stepping stones or unintentional accomplices to further spread attacks. Similarly, DoS attacks rely on target hosts to expend valuable resources by unnecessarily processing bogus service requests. Since a malicious host cannot simply inject unwanted traffic to harm a LIPS-protected destination without first requesting a permit and identifying itself to the destination[3], we can better defend such attacks by simply *detecting anomalies in permit request traffic.* For example, a sudden/unusual surge of permit requests to one or more hosts in a protected network signifies suspicious activities. In particular, when implemented in the gateway mode (introduced in the next section), a source zone can detect attacks originating from malicious or worm/virus-affected hosts within its zone and quarantine them by denying (host-specific) request permits to their target destinations. Hence combined with network intrusion mechanisms, LIPS can form an effective first line of defense against cyber attacks by stopping unwanted

[3] An attack may flood a destination, but its packets will be simply dropped and can not harm applications as today's worms. We will discuss the prevention of flooding attack in Section 4.

traffic. In summary, LIPS is designed to localize spoofing and associated attacks, restrict worm spreading, stop random probing and reflection attacks, assist IDSs in significantly reducing their load and providing cross-domain feedbacks, and protect important servers and their incoming links.

3 LIPS Design and Implementation

For incremental deployment and scalability, we design LIPS operating in two modes. The basic LIPS works in a *host mode*, in which a LIPS-aware host directly communicates with another LIPS-aware host as introduced in the previous section. The LIPS host mode is used as an incremental approach to deploy LIPS when a few LIPS-aware hosts directly communicate with each other in a small scale, and it is also used for communications within a zone under the gateway mode when LIPS is deployed in a large scale. We refer readers to [2] for the details of LISP host mode. Here we mostly introduce the LISP *gateway mode*.

We organize LIPS-aware hosts into *secure zones* based on their network administrative domains or zones. We use *zone access permits* to authenticate *inter-zone* packets, and use *host access permits* (as in the host mode) to authenticate *intra-zone* packets. Each zone has a *permit server (PS)* to manage inter-zone permits and a *security gateway (SG)* to validate inter-zone packets based on inter-zone permits. Once an inter-zone permit is established between a pair of zones, the subsequent communications between them will take advantage of this permit and avoid repeatedly setting up inter-zone permits. As a result, we not only reduce permit setup delays but also significantly reduce inter-zone permit exchange traffic. Furthermore, we propose a unique and simple *permit-mutation* method to transform zone permits and host permits back and forth such that *not only security gateways do not need to keep per-flow states but also zone permits are not revealed to hosts.* Another advantage of permit-mutation is to localize damage caused by potential attacks as discussed later. In each zone, LIPS-aware hosts still directly communicate with each other as in the LIPS host mode.

Permit Server, Intra-zone and Inter-zone Permit Setup Protocol. As show in Fig.5, host H_1 in zone Z_1 wants to access host H_2 (e.g., a protected application server) in zone Z_2. PS_1 is the permit server of zone Z_1, and PS_2

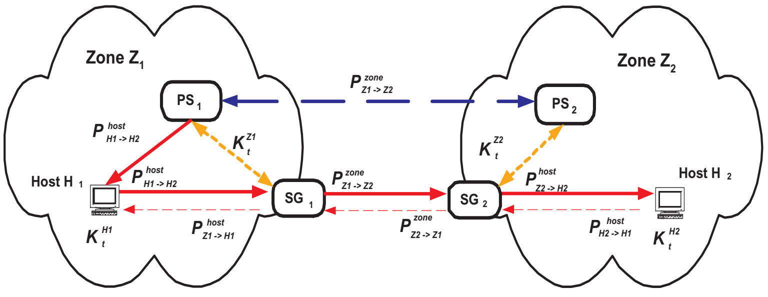


Fig. 5. Illustration of LIPS Gateway Mode

is the permit server of zone Z_2. Zone Z_1 and zone Z_2 are protected by security gateways SG_1 and SG_2, respectively, which authenticate both ingress and egress traffic originating from and destining to trusted hosts in these zones. To obtain a permit to access remote host H_2, H_1 authenticates itself to its local permit server PS_1 (e.g., a local authentication scheme such as Kerberos). PS_1 assists H_1 to obtain an access permit to H_2. Under a two-tiered model, we divide the packet forwarding path from a local host H_1 to a remote host H_2 into three segments: from H_1 to SG_1, from SG_1 to SG_2, and from SG_2 to H_2. Correspondingly, we use three access permits at each of these segments for packet authentication: an intra-zone host access permit $P^{host}_{H_1 \rightarrow H_2}$, an inter-zone permit $P^{zone}_{Z_1 \rightarrow Z_2}$, and another intra-zone host access permit $P^{host}_{Z_2 \rightarrow H_2}$. We introduce the setup protocols for these permits in the following.

Each PS is assigned a *zone ID*. In this prototype design, we simply choose the IP address of a PS as its zone ID since inter-zone permit requests and replies will be exchanged between PSs. We use this zone ID to generate a zone access permit as follows. In response to a permit request from a trusted host, the local PS passes the request to the corresponding (authoritative) PS in the remote secure zone[4], together with its zone ID and other necessary credentials. If the access is allowed, the remote PS will generate a *zone access permit* (or *zone permit* in short) based on the local PS's zone ID. Hence the access permit is *source-zone specific*. The remote PS returns the zone permit to the local PS together with its *own* zone ID. Instead of directly passing the zone permit to the requesting host, the local PS creates a new *host access permit* (or *host permit* in short) by adding some "random" value generated based on the source and destination IP addresses as explained in the following. This *mutation* of a zone permit into a host permit makes the host permit *specific to both source host and destination host*, thereby rendering it difficult to be spoofed by other hosts.

Zone Access Permits. Zone access permits are generated in the same fashion as host access permits but use a zone ID as a plain hash message. For a packet, let use IP_1 to denote its source IP address of host H_1 in zone Z_1, and use IP_2 to denote its destination IP address of host H_2 in zone Z_2. Let IP_{PS_1} be the IP address of a requesting PS (as a zone ID), and $K_t^{Z_2}$ be a *secret key* maintained by the queried PS_2 at time t. Then the secure hash value of the zone permit is $P^{zone}_{Z_1 \rightarrow Z_2} = H(IP_{PS_1}, K_t^{Z_2})$, where $H()$ is a secure hash function, and the CRC checksum of the permit is computed on $P^{zone}_{Z_1 \rightarrow Z_2}$. As explained in the following, the CRC checksum is used to verify the validity of the permit after the *permit de-mutation* for outbound packets. Note that the generated permit is *specific* to the requesting zone, and is valid only for a certain period of time, as $K_t^{Z_2}$ changes over time. Without knowing $K_t^{Z_2}$, it is very difficult to forge a zone permit.

[4] For a PS to find an authoritative PS of a domain, we add a simple resource record at the DNS of a domain such that a PS can find another PS through a simple DNS query, based on a simple name convention. We assume that DNSsec will solve security issues related to current DNS and so we will not discuss DNS security in this paper.

Mutation of a Zone Permit to a Host Permit. Given the zone permit $P^{zone}_{Z_1 \rightarrow Z_2}$, the requesting PS *mutates* it into a host permit $P^{host}_{H_1 \rightarrow H_2}$ using the IP address of the requesting (source) host, IP_1, and the IP address of the queried (destination) host IP_2. Let $K^{Z_1}_t$ be a *secret key* maintained at the requesting PS at time t. We construct a host permit, $P^{host}_{H_1 \rightarrow H_2} = P^{zone}_{Z_1 \rightarrow Z_2} \oplus H(IP_1, IP_2, K^{Z_1}_t)$. Note that the host permit $P^{host}_{H_1 \rightarrow H_2}$ is only valid for the source H_1 to access the destination H_2 for a certain period of time. Again, without knowing the secret key $K^{Z_1}_t$, it is also very difficult to forge a host permit. The host permit is essentially the same as the zone permit, with the secure hash value $P^{zone}_{Z_1 \rightarrow Z_2}$ replaced by $P^{host}_{H_1 \rightarrow H_2}$. Note that the CRC checksum is *not* re-computed.

Host and Gateway Operations. At both source and destination domains, we establish lightweight packet authentication mechanisms for verifying and filtering packets based on host and zone access permits.

Host Operations. In the gateway mode, we install a *host authentication layer (HAL)* at each host. During its initialization, a HAL authenticates itself to its permit server and security gateways, e.g., via a local authentication scheme such as Kerberos. This authentication only occurs once during its initialization. In the meantime, it also issues *host access permits* to its PS and its SG for authenticating permit replies and LIPS data packets from them, e.g., host H_1 issues permit $P^{host}_{Z_1 \rightarrow H_1}$ to PS_1 and SG_1.

The HAL layer at an end host x intercepts each outbound packet and then looks up its permit cache based on the destination IP address of the packet. If a destination access permit is found, it is attached the permit to the packet. In addition, the host will attach its source access permit generated using its local zone ID IP_{PS_i}, $P^{host}_{Z_i \rightarrow x} := H(IP_{PS_i}, K^{H_x}_t)$, where $K^{H_x}_t$ is a secret key kept by the host at time t. This source access permit is used for authenticating packets from the security gateway to the host. For each *incoming* packet, the HAL checks the validity of the destination permit using the *destination zone ID* (carried in the permit) and its own secret key. (It is the *reverse* operation of generating the source access permit in the above). The packet is accepted only if it passes the verification. In this case, the source access permit is *cached* in the permit cache (with a timer appropriately set, in a manner similar to the ARP table used for IP and MAC translation).

Gateway Operations. The *gateway authentication* layer (GAL) is a LIPS realization at a SG, which is a small patch to the IP layer. For *outgoing packets*, the SG is responsible for ensuring that they are authorized to access the protected remote zones and hosts. To verify this, it uses the source IP address IP_1, the destination IP address IP_2, and the destination access permit $P^{host}_{H_1 \rightarrow H_2}$ (carried in the packet) to first compute $X := P^{host}_{H_1 \rightarrow H_2} \oplus H(IP_1, IP_2, K^{Z_1}_t)$, where $K^{Z_1}_t$ is the secret key that the SG shares with the local PS. It then generates the checksum on X. If the computed checksum *does not* match the checksum carried in the destination access permit, the authentication fails and the packet is dropped. Otherwise, the secure hash value in the destination access permit is replaced by

X (note that $X = P^{zone}_{Z_1 \to Z_2}$), and thus the destination access permit is *de-mutated* back to the original zone access permit issued by the destination zone. Furthermore, the (host) source access permit of H_1, together with the source host IP address, is cached in the LIPS permit cache at the gateway. In addition, the gateway will replace the (host) source access permit in the packet with a new (*zone*) source access permit, $P^{zone}_{Z_2 \to Z_1} := H(IP_{PS_2}, K_t^{Z_1})$, where IP_{PS_2} is the IP address of PS_2 as the *destination* zone ID. $P^{zone}_{Z_2 \to Z_1}$ is used to authenticate packets from the destination zone Z_2 on the reverse path.

For packets entering a destination zone, the security gateway is responsible for verifying that they carry proper zone access permits. This is done by checking to see whether the destination permit carried in an incoming packet, $P^{zone}_{Z_1 \to Z_2}$, is valid. If this verification fails, the packet is discarded. Otherwise, the packet is allowed to enter the destination zone. Using the destination IP address IP_2, the gateway looks up its permit cache and replaces the destination zone permit with the corresponding destination host permit. Depending on whether the destination host is a trusted host (e.g., a server) in a *protected* (e.g., secluded) network, or a client host in a less secure environment, the gateway may replace the source *zone* access permit, $P^{zone}_{Z_2 \to Z_1}$, with a mutated source *host* access permit, $P^{host}_{H_2 \to H_1} := P^{zone}_{Z_2 \to Z_1} \oplus H(IP_1, IP_2, K_t^{Z_2})$. In the former case, for scalability this operation is *optional* so that trusted servers and other high-performance hosts in protected networks only need to maintain zone-level access permits. In the latter case, this operation would prevent other untrusted hosts to eavesdrop and forge (zone) access permits. A detailed illustration of operations in LIPS gateway mode can be found in [2].

Advantages of LIPS Design and Implementation. The design and implementation of LIPS have the following several salient advantages. As noted earlier, a key feature of LIPS is that *no secret* is shared across network domains, which makes the architecture more scalable and flexible. Packet authentication is performed using only information carried in (the LIPS header of) a packet and *secret keys* held *locally* by security gateways and hosts. Thus packet operations can be done efficiently. Furthermore, our architecture is purely *edge-to-edge* (or "end-domain-to-end-domain"), as it does not require any *intermediate* networks for assistance. It is also *incrementally* deployable: only those hosts that need to be secured have to be patched with simple protocol enhancement, and to be placed "behind" security gateways for authentication and protection. In addition, *no* modification to applications is required. Through separate *zone-level* and *host-level* access permits, we isolate "bad" packets originating in one's own zone from those outside, and limit the abilities of attacks to mostly "man-in-the-middle" *replay* attacks by "sniffing" permits. Permit- or Packet-Replay attacks can be mitigated by including, e.g., sequence no. or random bits, in access permits. By augmenting LIPS with active monitoring and rapid response defense mechanisms, we can quickly detect and throttle such attacks (e.g., by detecting duplicate access permits, and adjusting timed keys). With such mechanisms, replay attacks will have very *localized* effect, with only "sniffed" hosts/domains being affected, due to the host-specific/domain-specific feature of access permits.

4 Performance Evaluation

We have evaluated the basic overhead of the LIPS itself, and examined the effectiveness of LIPS in protecting server resources. In the following, we present these results briefly. Readers can find more details and results in [2].

Overall Overhead of LIPS. We conducted experiments to examine the overall overhead introduced by our LIPS implementation on Linux platforms in data transmission, compared with IP. In these experiments, we used Iperf to send an CBR UDP flow from a host to another via a dedicated 100Mbps link. When the CBR rate is lower than 100Mbps, there are almost no differences between the transmissions with or without LIPS. Table 2 shows the Iperf measurements when the CBR rate is 100Mbps. Even in this stress test, the difference between the transmission bandwidth with LIPS and that with IP is negligible small (3%). In our experiments, we also measured the delays of key LIPS operations including HMAC computation, permit lookup, and mutation. On a common Linux platform with 2.8MHz Pentium, we can authenticate around 640 Mbps [2], far beyond a common user's requirement.

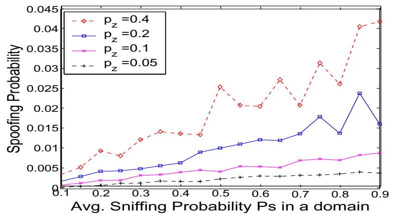
Effective LIPS Protection. We further use analytical models to show how LIPS helps stop DoS attacks from two aspects: the chances for zombies to start DoS floods and the probabilities of successful attacks. We focus on the replay of host permits in LIPS domains because it is rather difficult to gain access to inter-domain links to sniff a domain permit. The real time and host-specific nature of LIPS permits dramatically increases the difficulty to generate attacking traffic. Furthermore, we design a fast response mechanism to quickly stop floods. Therefore, it is extremely difficult to bring down a LIPS-protected target.

We first examine the spoofing chances for a zombie in a LIPS domain. Under LIPS, to sniff host permits for spoofing, a zombie must have access to the path to a destination in real time. We define p_z as the probability that a host is compromised as a zombie in a domain, and p_s as the probability that a zombie can sniff a valid permit to a target in the domain. Assume that a legitimate host communicates with a target server as a Poisson process. As shown in Fig.6(a), given different p_z and p_s, the spoofing probabilities for zombies under LIPS are far lower than 1, the chance under IP.

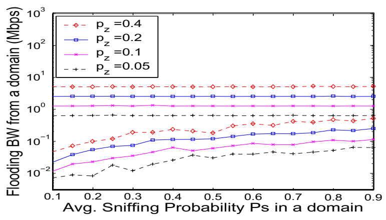
Consequently, LIPS dramatically suppresses zombies' capabilities to launch flooding attacks to a target. Assume we have a domain of 100 hosts; those hosts communicate with a remote server as Poisson processes; the mean flow rate of a legitimate session is 128Kbps.

Table 2. Comparison: with and without LIPS over a dedicated link

	Effective Bandwidth	Loss Rate	Jitter
With LIPS	90.7 Mbps	0.005%	0.025ms
W/O LIPS	93.7 Mbps	0.005%	0.022ms



(a) Spoofing Probabilities in LIPS for a zombie



(b) Comparison of Aggregate Spoofing Bandwidth from a domain: IP vs. LIPS

Fig. 6. LIPS significantly reduces spoofing chances and restricts flooding capability

Fig.6(b) shows that the aggregate flooding bandwidth to the server, which can be generated by zombies in the domain. The top four lines are flooding rates under IP with various p_z, while the bottom four lines are flooding rates under LIPS with the same conditions. Note that the Y-axis is in a log scale. Clearly, it is very difficult for zombies to generate sufficient traffic to flood the server under LIPS; while it is fairly easy under IP. In addition, our study also shows that an attacker needs about 10^4 to 10^5 domains as the above to flood a LIPS-protected 1 Gbps link with over 100% unwanted packets. It is extremely difficult for an attacker to collect such huge amount of resources.

Furthermore, we use a simple Stochastic Knapsack framework [5] to model a DoS attack to a protect incoming link of a target. We use C to denote the total amount of incoming bandwidth available. Assume legitimate flows (or attacking flows) have an exponential arrival rate with a mean of λ_l (or λ_a), a bandwidth requirement b_l (or b_a), and an exponential service time with a mean of μ_l (or μ_a). The system admits an arrival whenever bandwidth available. In this model, the probability of a successful DoS attack is the blocking probability corresponding to the legitimate traffic, defined as $P_b = 1 - \frac{\sum_S (\rho_l^{n_l}/n_l!)\cdot(\rho_a^{n_a}/n_a!)}{\sum_{S'} (\rho_l^{n_l}/n_l!)\cdot(\rho_a^{n_a}/n_a!)}$, where S is the set of cases that an arriving legitimate flow can be admitted, and S' is the set of cases that either a legitimate flow or an attacking flow is admitted; in each case, n_l is the number of legitimate flows admitted, and n_a is the number of attacking flows admitted; and offered load $\rho_l = \lambda_l/\mu_l$, $\rho_a = \lambda_a/\mu_a$. Fig.7 shows the blocking probability of legitimate flows as we increase the load of attacking traffic. To block 90% of legitimate traffic, the attacking load has to be 1000 times heavier than the legitimate traffic.

We also design an inter-domain collaboration scheme to stop permit-replay flooding in LIPS domains and take care of zombies across domain [2]. Since zombies have to generate high attacking load to launch successful attacks, it is easy to identify them and then isolate these zombies through a local defense mechanism. Furthermore, we can also identify and deal with zombies through automatic inter-domain collaborations. Fig. 8(a) and Fig. 8(b) show the effectiveness of our fast response scheme: to protect incoming links with various

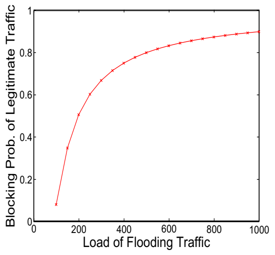
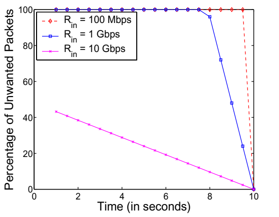
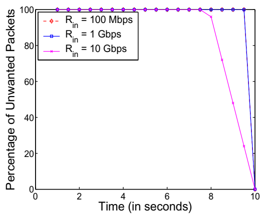


Fig. 7. Blocking Probability of legitimate flows as the attacking traffic load increases



(a) 1,000 replaying sources: 100 sources in each of 10 domains



(b) 10,000 relaying sources: 100 sources in each of 100 domains

Fig. 8. Static Attacks: Delays to shut off all replaying sources

capacities, we can quickly shut off these replaying source in 10 seconds for a large number of replaying sources.

5 Conclusions

LIPS is a simple packet authentication mechanism which provides traffic origin accountability for stopping unwanted traffic. We presented the basic design of LIPS and a prototype implementation on Linux platform. Our analytical, simulation and experimental results show that LIPS is capable of stopping unwanted packets with negligible overheads. Currently, we are incorporating active monitoring and rapid-response defense mechanisms into LIPS for further improving its security and performance.

References

1. T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. *Hotnets 2003*, Nov. 2003.
2. Y. Dong, C. Choi, and Z. l. Zhang. A lightweight permit system for stopping unwanted packets. *Technical Report, http://www.ee.hawaii.edu/~dong/papers/LIPS_report.pdf*, July, 2004.
3. D. Estrin and et. al. Visa protocols for controlling inter-organization datagram flow. *In IEEE Journal on Selected Areas in Communication*, May 1989.
4. G. Hadjichristo, N. Davis IV, and C. Midki. Ipsec overhead in wireline and wireless networks for web and email applications. *In Proc. of IEEE IPCCC*, Apr. 2003.
5. A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. *In Proc. of ACM SIGCOMM*, Aug. 2002.
6. A. Menezes, P. Oorschot, and S. Vanstone. Handbook of applied cryptography. *CRC Press, ISBN: 0-8493-8523-7*, 1996.
7. K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets. *Proc. of ACM SIGCOMM 2001, San Diago, CA.*
8. H. Wang, A.Bose, M.Gendy, and K.Shin. IP Easy-pass: Edge Resource Access Control. *In Proc. of IEEE INFOCOM*, 2004.