Lightweight Opportunistic Tunneling (LOT)

Yossi Gilad and Amir Herzberg

Computer Science Department, Bar Ilan University, Ramat Gan, Israel
{yossig2,amir.herzbea}@gmail.com

Abstract. We present LOT, a lightweight 'plug and play' tunneling protocol installed (only) at edge gateways. Two communicating gateways A and B running LOT would automatically and securely establish efficient tunnel, encapsulating packets sent between them. This allows B to discard packets which use A's network addresses but were not sent via A (i.e. are spoofed) and vice verse.

LOT is practical: it is easy to manage ('plug and play', no coordination between gateways), deployed incrementally and only at edge gateways (no change to core routers or hosts), and has negligible overhead in terms of bandwidth and processing, as we validate by experiments on a prototype implementation. LOT storage requirements are also modest. LOT can be used alone, providing protection against blind (spoofing) attackers, or to opportunistically setup IPsec tunnels, providing protection against Man In The Middle (MITM) attackers.

1 Introduction

IP Spoofing: The vast majority of packets sent on the Internet are not authenticated; namely, attackers are often able to send *spoofed* packets, containing incorrect sender IP address. IP spoofing is widely deployed in a variety of attacks, including Distributed Denial of Service (DDoS) attacks such as SYN clogging [10, 19, 13], network scans [21], spamming (by circumventing port-25 blocking or spamming at higher rates than zombie's connection speed), and other attacks, esp. on connectionless protocols such as SNMP [15, 1].

Currently, IP spoofing is often easy: once a packet with spoofed IP address leaves an ISP, it usually reaches its destination. ISPs should try to prevent IP spoofing by their clients, mainly by ingress filtering [18, 12, 4], blocking spoofed packets received from their clients. However, some ISPs do not perform ingress filtering (well), and an attacker may sometimes control a gateway at an ISP. IP spoofing is usually easier than intercepting IP packets sent to others (eavesdropping), although in certain scenarios, interception is also possible; see e.g. Bellovin's seminal paper [5].

In spite of the recommended best practice of ingress filtering, Pang et al. [20] as well as Beverly and Bauer [7] found that IP spoofing is still quite common. In particular, IP spoofing is often used for indirect DDoS attacks, e.g. DDoS on a victim by sending DNS queries with source address of the victim (to load victim with the longer responses).

M. Backes and P. Ning (Eds.): ESORICS 2009, LNCS 5789, pp. 104–119, 2009.

LOT: We present LOT, a simple, efficient, 'plug-and-play' protocol for establishing secure tunnels between two gateways. LOT requires no coordination between the administrators of the two gateways; instead, once it is installed on both gateways, it automatically sets up the tunnel between them. This tunnel prevents spoofing of sender's IP addresses.

The most obvious use for LOT is between source and destination edge gateways, of small to large networks. LOT may also be used to protect communication between the edge gateway (of a 'small' network, say FOO.COM) and the gateway of the autonomous system connecting FOO.COM to the Internet.

LOT has two main components: an *opportunistic tunnel setup* protocol and an *efficient tunneling* mechanism.

LOT's OPPORTUNISTIC TUNNEL SETUP allows two LOT gateways to identify and realize their ability to setup a tunnel between them. Furthermore, each gateway, e.g. A, identifies the block of IP addresses $Block(A)$ that are connected via A, and also validates that the other gateway, e.g. B, is really on the path from the addresses in $Block(B)$ to A. The challenge is to perform this validation efficiently, without allowing exploitation such as for DoS attacks. Specifically, LOT validates the address block claimed by each gateway by several rounds of cookie exchanges to different, randomly selected addresses in the address block. As shown later, this mechanism provides good probability of detection of false address blocks, very efficiently and without creating new risks of DoS.

LOT's EFFICIENT TUNNEL MECHANISM allows highly-efficient filtering of spoofed packets; we confirmed by experiments that LOT has very low overhead (compared to no tunneling). LOT authenticates the source IP address in packets, by attaching and validating a 'nonce' (random identifier). LOT's tunneling is very lightweight and optimized for efficiency, much like GRE [11, 9]. LOT tunneling has a novel option (cf. to GRE and other existing tunneling protocols), that can allow better performance, esp. for edge networks connected to the Internet via multiple routers (for multihoming, fault-tolerance or performance). Details within.

LOT, like GRE, is secure against a blind (spoofing) attacker, but not against a MITM attacker. Blind attacks are much more common, and many currently deployed mechanisms are secure against blind attackers but not against MITM attackers, e.g. by relying on TCP's three-way handshake. This allows LOT to be much more efficient than cryptographic secure tunnels, that offer security also against MITM attacker, such as IPsec and SSL/TLS [17, 8, 22]; in particular, unlike IPsec and SSL/TLS, LOT does not use the payload as input to computationally-intensive cryptographic operations, and hence has lower computational and storage requirements.

Instead of applying cryptographic authentication to the packets, the endpoints to LOT tunnels merely validate that packets arriving via the tunnel, contain an appropriate *cookie*. The cookie is selected by the receiving end of the LOT tunnel (Bob) and sent to the sending end of the tunnel (Alice); Alice attaches it to each packet it sends to Bob, and Bob filters any packet from Alice which does not contain a valid cookie. LOT cookies provide evidence that the sender previously

received a packet sent to a particular address, while limiting the amount of work by the recipient; this is much like the similar IKE and TCP cookies [16, 6] and the ϕ-filtering mechanism analyzed in [2] . In LOT, the maximal overhead per incoming (spoofed) packet is sending one packet in response, and computing one (efficient) shared-key cryptographic pseudo-random function. A similar effect can be obtained by using IPsec with randomly-chosen SPI values, and without encryption or message authentication .

When sufficient computational resources are available and security against MITM attackers is required or desirable, it is possible to use IPsec or similar tunneling mechanisms instead of LOT's tunneling, while still using LOT's opportunistic tunnel setup mechanism. This would reduce the management effort required to setup IPsec tunnels, esp. the need to coordinate between the two networks connected via the secure tunnel. Opportunistic IKE [23] was also proposed for the same function, however, it has significant overhead for connections to existing systems, that do not implement [23], which may even be exploited as a DDoS vector; and furthermore using [23] required configuring the reverse DNS tree, which requires additional management effort and is not always feasible.

2 LOT Specifications: Goals and Scenarios

In this section we present (informal) specifications for LOT, including LOT's design goals and deployment scenarios.

2.1 LOT Design Goals

LOT has the following design goals:

Prevent IP spoofing: LOT's most basic goal is to prevent a blind (spoofing) adversary on the Internet, Eve, from sending one of LOT's gateways, say GW3, packets from a network behind another LOT gateway, say GW2 (see Figure 1). This protection should work, of course, assuming that GW2 and GW3 have already established (opportunistically) a LOT tunnel between them.

Do no harm: LOT tunnels are designed to improve security, in particular defenses against IP-spoofing and against DDoS attacks. These goals are obviously important; however, it is critical that such improvements will not result in significant losses in efficiency or reliability. In particular, LOT tunnels should be established and operated with minimal overhead, including no or minimal impact on routing; furthermore, clearly the LOT mechanisms should be designed carefully, to make sure LOT itself cannot be abused to perform DoS, spoofing or other attacks.

Incremental, edge-only deployment: Deploying new mechanisms for Internet security can be challenging, esp. when the mechanism involves tunneling, i.e. requires adoption at both ends to provide value. In light of this, it is highly desirable for such new mechanisms to be *incremental*, i.e. provide value even

when adoption is very limited, and gradually increasing as the number of deployments grows. It is also highly desirable to restrict new functionality to the 'edge' of the Internet. In LOT, this is achieved by requiring adoption only by the gateways connecting networks to the Internet.

Simple, Easy, Plug and Play: Secure tunneling mechanisms, and in particular IPsec, have established a reputation of being overly complex to implement and difficult to install and deploy. This complexity and difficulties may be the biggest obstacle preventing the wide-spread deployment of IPsec. It is therefore desirable for LOT to return to the 'KISS principle': Keep It Simple (Stupid), and be simple and easy to install and deploy. LOT uses 'plug and play' tunnels, established automatically (opportunistically).

Scalable: LOT is scalable, to allow for potential large-scale deployment by many of the networks in the Internet. In particular, it requires only a small amount of storage per tunnel.

2.2 LOT Deployment Scenarios

There are two typical deployment scenarios for LOT: network-to-network and network-to-provider. In the network-to-network scenario, illustrated as tunnel B in Figure 1, a LOT tunnel is established (opportunistically) between the edge gateway GW3 of Bob's network, and the gateway GW2 of Alice's ISP. This ensures that whenever Bob receives a packet from any host behind the ISP it was really sent by a host behind the ISP.

The other typical scenario is network-to-provider, as illustrated in Figure 1. Here, a customer runs LOT in the gateway connecting it to the ISP (GW1), and establishes (automatically) a LOT tunnel - tunnel A, to another LOT gateway (GW2), installed by the ISP. This deployment can help ISPs with complex networks enforce ingress filtering.

It is also possible that multiple LOT tunnels would be established along the route between two networks. For example, in Figure 1 we show two tunnels from Alice's network to Bob's network: a tunnel from Alice's network gateway to her ISP's gateway (tunnel A), and another tunnel from the ISP's LOT gateway, to Bob's LOT gateway (tunnel B).

Finally, we note that often, a network may be connected to the Internet or other networks via multiple gateways. LOT also supports this (common) scenario, as illustrated in Figure 2. Specifically, unlike other tunneling protocols such as IPsec VPN and GRE, LOT avoids impact on routing efficiency, by

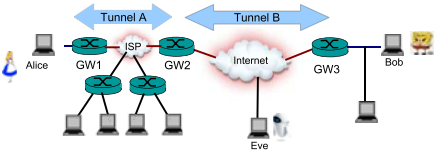


Fig. 1. Two LOT tunnels: from customer to ISP (Tunnel A), and from ISP to remote network (Tunnel B)

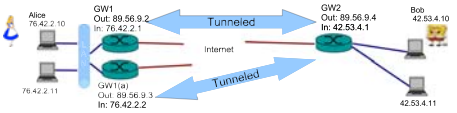


Fig. 2. LOT deployment when one network is connected via multiple gateways

tunneling packets without changing their source and destination addresses to these of the gateways, forming a 'transparent' tunnel providing better QoS to end hosts. For more details see Section 4.

3 LOT Handshake

Every LOT connection begins with a handshake during which cookies are exchanged. Later, these cookies are attached to packets sent by the peers to verify they are not spoofed. In this section we present the handshake protocol; Figure 3 illustrates the process of setting up LOT tunnel between gateways GW1 and GW2.

The LOT handshake protocol is triggered by GW1, as it forwards a packet from some host Host1, to another host Host2, whose IP address does not belong to one of the address blocks with whom GW1 has already established a LOT tunnel. GW1 begins the handshake by sending the LOT hello message (step 1 in Figure 3) to Host2.

GW1 sends the LOT hello request packet to a reserved UDP port to which we refer as LOT_PORT. This allows GW2 to intercept the hello packet and respond. In any case, if GW1 does not receive a valid response, then the handshake silently fails; notice that at this stage, GW1 did not allocate any state for the handshake (preventing DoS attacks similar to SYN clogging).

To further limit overhead, GW1 sends LOT Hello request only with rather low probability p (e.g. 0.01 or 0.001) per forwarding of packet to destination (Host2) without established tunnel; GW1 may also keep cache of destinations to which it recently sent LOT Hello and avoid sending to destinations in cache.

The hello request contains:

- GW1's current time $time_1$.
- An initiation cookie $cookie_1 = PRF_{k_1}(Host_2||time_1)$ where PRF is a pseudo-random function, e.g. AES, and k_1 is a secret key. When GW1 is the only gateway connecting a network partition containing Host1 to a network partition containing Host2, as in Figure 1, then k_1 is known only to GW1. When GW1 is part of a set of gateways connecting a network partition containing Host1 to a network partition containing Host2, e.g. together with GW1a in Figure 2, then k_1 is shared among these gateways (GW1 and GW1a in this example).
- GW1's network addresses block $netblock_1$, specified by a pair $(address, l)$ where $address$ is a network address (32 bits for IPv4, e.g. 128.1.2.3), and l is the number of bits in the 'network part' of the address, i.e. the address

block contains all addresses with the same l most-significant bits as *address*. We use the familiar CIDR notation $address/l$.

- GW1's direction d_1, which has two possible values: in and out. If d_1 =in, then all addresses x in $netblock_1$ are in the network partition 'behind' GW1 . If d_1 =out then *all* network addresses are 'behind' GW1, *except* for the addresses in $netblock_1$, addresses in its partner network block (i.e. GW2 network block) and a designated set of addresses denoted Martian (see [14]), containing addresses which can be appear in multiple locations in the network (e.g. 10.0.0.0/24).

When GW2 intercepts the hello message (step 2 in Figure 3), it ignores it with a constant (configurable) probability q (which is typically close to 1, e.g. 0.9), to protect GW2 from DoS attack of flooding it with LOT Hello requests. This implies that the expected number of packets sent by GW1 to GW2 till the LOT tunnel is established, is roughly $\frac{1}{p(1-q)}$. Additionally, GW2 checks if there is already an existing tunnel to Host1.

When GW2 selects to respond, then it sends LOT hello response, identifying GW2's network block $netblock_2$, its 'direction' d_2 (similar to d_1 in LOT Hello Request above), and n_2, the minimal number of verification rounds required by GW2. Subsection 3.1 explains how GW2 determines n_2. The response also contains $cookie_1$ and $time_1$ as received from GW1, and GW2's own cookie, $cookie_2$, computed as we now explain.

Although GW2 received the LOT Hello Request from GW1, it sends the Hello response not to GW1, but to a pseudo-random address IP_2 within netblock_1 (if d_1 =in; if d_1 =out, then IP_2 is a pseudo-random address *outside* netblock_1). Both IP_2 and $cookie_2$ are computed by the simple *LOT challenge function* presented in Algorithm 1: $(IP_2, cookie_2) = F_k(\text{mynetblock}, \text{netblock}_1, d_1, time_2, 1, n_2)$. The fourth parameter ($i$) is the number of the verification round (for this message, simply 1).

In addition to $cookie_2$ and its network block, GW2 also attaches to its message n_2, GW1's cookie, and the time $time_1$ received from GW1. GW2 sends this message using the source IP address $Host_2$; the destination of GW2's hello response message is IP_2.

Next comes the network block validation phase, which begins when GW1 receives GW2's Hello Response packet (with valid time and cookie). Notice that GW1 can validate the packet's authenticity given $time_{GW1}$ only without requiring the to keep state (the $Host_2$ used to create GW1's initiation cookie is specified as a source IP). In addition, GW1 verifies the time is reasonable (i.e. $cookie_{GW1}$ in GW2's response is not too old). During the network block validation phase, the two gateways GW1 and GW2 verify each other's network blocks using a statistical challenge response test, with several iterations, to verify the network block claimed by the other gateway. We describe the details later, in subsection 3.1.

Finally comes the cookie exchange phase. After phase 2 is complete and both sides were authenticated, each side maps in its data base the remote network block to a tuple containing:

$F_k(netblock, d, time, i, n)$

if $d ==$ in **then**
  $\varphi = PRF_k(netblock||time||i||n)$
  $DestIP = netblock + \varphi[0...(31 - l)]$

end
else
  $iteration = 0$
  repeat
    $\varphi = PRF_k(netblock||time||i||n||iteration)$
    $DestIP = \varphi[0...31]$
    $iteration+ = 1$

  until $DestIP$ *not in* {netblock $\bigcup$ mynetblock $\bigcup$ Martian} ;
end
$Challenge = \varphi[32..63]$
return $DestIP$, $Challenge$

Algorithm 1. LOT challenge function F, pseudo-randomly determining challenges for the netblock authentication phase. F uses a pseudo-random function PRF, which may be implemented e.g. with AES

- The last challenge which will be used as the "tunnel cookie".
- The remote peer's time specified in the last challenge, and the security parameter n, which were used to create the last challenge (see Figure 3 step no. 4). These allow the recreation of the tunnel cookie by the recipient gateway, enabling it to verify its authenticity.

The entire handshake protocol is sent over UDP. To avoid problems caused by the loss of the last challenge containing the tunnel cookie, the respondent (GW2 in Figure 3) ACKs this packet. The ACK is authenticated by the last challenge received from the remote peer (see Figure 3). Notice that at this point GW2's identity is already validated by GW1 so GW1 may keep state to measure timeout. If the sender does not receive an ACK for his last message (containing the tunnel cookie), then he retransmits his cookie (few retransmissions are allowed, e.g. three). The last packet must be assured to have reached its destination as both peers must realize a LOT tunnel was established. If any other packet is unanswered, then the sender does not retransmit, the handshake will simply fail and the two sides will try to construct the tunnel again later.

Furthermore, to avoid race conditions after LOT handshake is complete, LOT has a short grace period which allows unauthenticated packets to pass through the gateway to the network for a short period of time; it is only after the grace period is finished, that LOT tunneling becomes mandatory.

Support for networks with multiple gateways. LOT provides support for networks with multiple gateways, such as Alice's network described in Figure 2. While a challenge may be routed to its destination through either one of the network gateways, the stateless nature of the LOT handshake allows every

gateway to respond provided all the network gateways share the same secret key. When a tunnel is set up, the LOT gateway who handled the last LOT handshake packet informs other gateways of the new tunnel.

3.1 Netblock Validation

When authenticating a gateway it is not enough to simply send one challenge to it or to a single host in the network behind it since the attacker may control some hosts (but not entire network). However, sending challenges to all hosts is very inefficient and opens the door to DoS attacks on the responder.

In Figure 3 we illustrate the handshake, including a probabilistic protocol to validate the network block claimed by the peer. Validation is done simultaneously for both gateways.

To avoid DoS attacks both on the authenticator itself or any other entity, the authenticator sends only one packet for every packet it receives. This also helps to prevent the usage of the authenticator by a malicious entity for reflection DoS attacks.

At each step of the validation, each gateway sends one packet to a random address in the netblock claimed by its partner; this is a form of a challenge - if the gateway really protects this network block, then it can easily intercept these packets (step 4 in Figure 3). The addresses and challenges are derived using the function described in Algorithm 1.

If the network block reported by the remote gateway is correct, then the remote gateway can intercept the challenge and respond. When a gateway intercepts a challenge it first validates its own cookie specified in the challenge by reconstructing it using its secret key and the parameters given within the challenge packet itself (see Figure 3). This assures the sender has received the previous LOT message. In addition, the responder compares its current time with the time specified in the challenge validating the challenge is not too old.

Then the responder creates its own challenge, chooses randomly an end host in the remote network (using the function F described in Algorithm 1), and sends the challenge to the chosen host. The challenge is sent along with an echo of the cookie received and the other parameters used to validate the challenge response. See packets 4 in Figure 3. If the echoed cookie is invalid the challenge is discarded.

This process of challenge and response is done n times depending on the probability of verification required as analyzed in Section 5. An authenticator may use different n values to authenticate large networks with the same probability of small networks, however n and the current iteration number are obtained in the response from the remote gateway (see packet 4 in Figure 3), since they are inputs to F, they can not be forged.

4 LOT Tunneling

LOT tunnels communication using a cookie obtained during the LOT handshake. The idea of attaching a pseudo-random field to packets to assure their origin

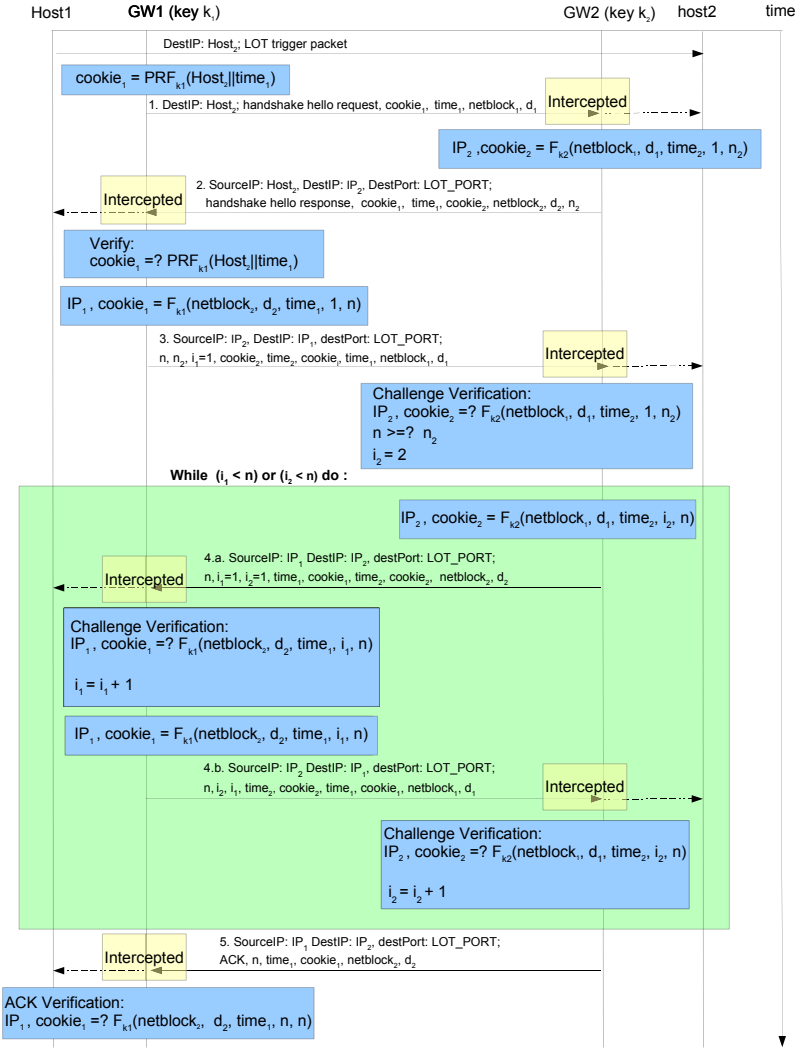


Fig. 3. LOT handshake and netblock validation, dashed arrows represent packets which were blocked from reaching their original recipient

authenticity was previously introduced as a key extension for GRE in [9] and the FI field in [3]. LOT tunnels are 'transparent'. Namely, LOT does not change the source and destination IPs of tunneled packets. The transparency allows packets to be routed through either one of a network gateways for networks topologies such as Alice's network described in Figure 2. This characteristic allows load distribution between the network gateways. Notice that providing all network gateways share the same network key they are all able to tunnel outgoing packets and authenticate incoming packets.

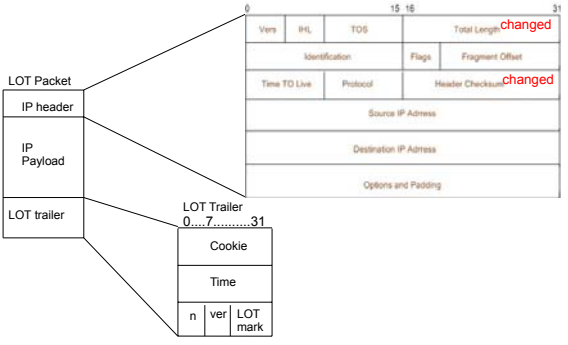


Fig. 4. LOT tunneled packet. Changed IP header fields are marked as 'changed'.

LOT attaches its data at the end of the packet, right after the application layer's data. A tunneled LOT packet is described in Figure 4. The attachment of LOT data at the end of the packet is more efficient - other possible places for inserting LOT data such as adding a LOT IP option or placing it right after the transport layer protocol header require the tunneling gateway to 'break' the packet and insert the LOT data in the middle - shifting forward packet bytes to make room for LOT data.

Figure 4 shows the structure of a packet sent via the LOT tunnel, between the two LOT gateways. The LOT trailer contains the following fields:

LOT cookie. The LOT cookie is added to provide proof for the packet's authenticity (i.e. proof the packet originates form the network behind the tunneling gateway).

LOT mark. A two bytes identifier which identifies LOT packets. During grace period, LOT gateways also forward to their netblock incoming packets from source addresses which should be tunneled. The LOT mark distinguishes between such untunneled packets and tunneled packets which should be decapsulated first. If during the grace period an untunneled packet which contains (accidentally) the LOT mark arrives the receiving LOT gateway will treat it as a tunneled packet. The packet will likely be dropped because of invalid cookie. This event may happen only during the short grace period and even that with a rather low probability of 2^{-16} (as the LOT mark is 16 bits long).

Version. A single byte field that holds the LOT version to support future versions.

n. The value of the security parameter n used to create the cookie.

time. The time specified by the cookie's creator. The value n and the time allow the packet's authenticator to reconstruct the cookie using the function F (i.e. compute $F_{key}(netblock, time, n, n)$ as in Algorithm 1), the receiver's network block is retrieved from the LOT database described in section 3 and authenticate the packet. In addition, the time field may be used to enforce expiration dates on cookies as the receiving gateway can use it calculate the time passed since the cookie was created.

LOT's addition to a packet is relatively small and consists of only 12 bytes.

When a LOT gateway receives a packet from a host in the network it protects to forward to an address in the outer network, it first checks if there is an entry in its database of existing LOT tunnels, whose destination address block includes the destination IP of the received packet. If found, it adds the corresponding LOT trailer from the database to the end of the packet and modifies the IP length and header checksum fields as described above. Otherwise, if the destination IP address is not included in any existing address block in the LOT database, then LOT forwards the packet as it was received, and randomly sends a LOT handshake hello packet (see Section 3).

When a LOT gateway receives a packet from the outer network to forward to a host in its own network, then it checks if it has an entry in its data base matching the source IP specified in the packet. If it does not it forwards the packet to the destination host. Otherwise, the gateway decapsulates and forwards the packet, but only if it contains a LOT mark and a valid cookie at the end.

For efficiency LOT gateways keep a small cache of valid cookies and their corresponding network blocks and rebuild the cookie using the function F only if it is not in the cache.

5 LOT Security Assumptions and Properties

LOT is designed for security against 'blind' attackers, which can send a limited number of (legitimate or spoofed) packets per second. We allow the adversary to intercept (receive) packets only to a reasonable subsets of IP addresses at any given time (second); we assume the adversary cannot eavesdrop on packets sent to other addresses. Specifically, the adversary controls an arbitrary set $A[t]$ of different IP addresses at any given time t (in seconds), s.t. $|A[t]| \leq \alpha$, where α is a bound on the number of adversary-controlled addresses (per second).

For any network block B (set of all addresses with given prefix), we assume that either $B \subseteq A[t]$ (entire block is corrupt) or $\frac{|B \wedge A[t]|}{|B|} \leq \beta$ (adversary controls at most β of the addresses in the block, where β can be e.g. $\frac{1}{2}$ or $\frac{3}{4}$). This assumption appears reasonable since network addresses are typically assigned either in blocks or as random samples from a large set (e.g. by DHCP).

LOT ensures the following security properties against such adversaries, for any pair of hosts, Host1 behind LOT gateway GW1 and Host2 behind LOT gateway GW2:

No spoofing. If Host1 receives packet whose sender address is Host2, at time $t > 1$, then a host behind GW2 sent that packet (recently, i.e. in time t' s.t. $t - 2 \leq t' \leq t$). We can also allow small probability p of spoofing (this probability should be small but not necessarily 'negligible', since a spoofed packet can only cause limited damage - e.g., 0.01 may often be OK).

No blocking. Packets sent by Host1 to Host2 are received (without significant extra delay).

LOT ensures these properties, assuming reasonable processing capabilities, reasonable network delays of less than one second, and a steady stream of requests between Host1 and Host2. We further assume that both Host1 and Host2 are *not* contained in any network block where more than β of the addresses are controlled by the adversary (recently).

Formal specifications and analysis would appear in the full version. Below we discuss three basic issues: *network block validation*; the *tiny network block threat*; and *prevention of DoS on LOT* itself.

Network Block Validation Security: Under the above assumptions we investigate the security of the netblock validation protocol. We calculate the probability that an attacker successfully completes validation process for some *netblock* of size x, running against some LOT gateway GW, while in reality attacker controls only $l < x \cdot \beta$ of the addresses in *netblock*.

The probability that the attacker's fraud is not discovered in a single step, is the probability of choosing a host controlled by the attacker (i.e. in $A[t]$), i.e. $\frac{l}{x}$. Thus the probability p that the attacker's fraud is not discovered after n steps: $p = \left(\frac{l}{x}\right)^n$.

This yields that to ensure maximal probability of spoofing p, it is enough to use: $n = \left\lceil \frac{\log(p)}{\log\left(\frac{l}{x}\right)} \right\rceil$.

Notice that n grows logarithmically by p and the ratio $\frac{l}{x}$. If we assume an extreme case where $\frac{l}{x} = 0.75$ and $p = 0.001$ we yield $n = 25$. Namely, only 25 iterations (25 challenges) are needed to verify a gateway's claim with probability of 0.999, even if it controls up to 75% of the entities in the network. We believe smaller values of n would usually suffice.

The Tiny Network Block Threat: Setting up a LOT tunnel for tiny network blocks could cause the LOT gateway database to become extremely large. Moreover, an attacker may often be able to obtain short-term control over different all or most of the addresses in multiple tiny network blocks, e.g. by obtaining DHCP leases for different IP addresses. It then can set up LOT tunnels between the victim gateway and these tiny network blocks.

When the IP addresses used by the attacker are re-used (e.g. by the DHCP server), packets sent to the victim gateway by legitimate client re-using the addresses will be dropped (since they are not properly tunneled); furthermore attacker can continue to send spoofed packets using the addresses, even after it lost control over the address. Notice, however, that unlike the situation with other tunneling mechanisms such as GRE, the attacker cannot intercept ('hijack') packets since LOT does not modify the destination IP address.

There are two solutions to this threat. The first is simply to allow LOT tunnels only for sufficiently-large address blocks. This would reduce memory requirements of LOT gateways, and prevent the above attack, as even if the attacker controls enough zombies their addresses would have to remain consecutive after each time they change.

The second solution allows tiny address blocks, by RE-VALIDATING LOT TUNNELS. Namely, LOT gateways will perform a simple validation protocol when a packet without the 2-byte LOT mark is received from a host within a (small) tunnel. In this case the receiving gateway, GW1, drops the received packet, but with low probability it also sends a challenge to the originating host. If the tunnel is valid then the originating host gateway (GW2) can intercept the challenge and reply. If it does reply not, as would be the case in the 'tiny network block' attack described above, timeout occurs and GW1 tears down the LOT tunnel.

The challenge contains a random field and is authenticated by the cookie used by GW1 to tunnel packets sent to the network behind GW2. The response echoes the random field and includes the cookie used by GW2 to tunnel packets to the network behind GW1. The packet may be resent several times if the challenge is unanswered to avoid problems caused by transport layer unreliability.

PREVENTION OF DOS ON LOT. LOT is designed to avoid 'amplification' and 'reflection' Denial of Service (DoS) attacks using the LOT handshake or encapsulation mechanisms. In particular, during handshake, LOT performs only very limited computations and sends only a single packet, in response to any incoming packet. Furthermore, LOT requires only very limited (constant) storage per peer LOT gateway. Therefore, we believe that LOT cannot be abused for DoS attacks. See also the results of our experiments reported in the next section.

6 Test Runs

We prototyped LOT as a Linux kernel module for Linux based network gateways and used it to test LOTs performance. For testing we used two hosts and two network gateways between them. The end hosts were connected to their gateways via a 100Mbit per second Ethernet network. The gateways were connected between them when via a 10Mbit per second Ethernet network.

All network entities consisted of the same hardware - Pentium D 3.6MHz computers with 2GB RAM and 4MB cache, running Linux with kernel version 2.6.18.

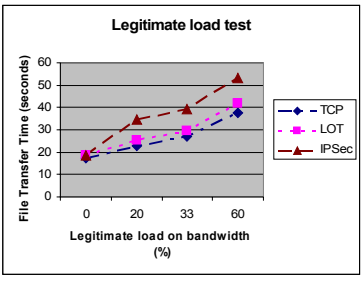
The two end hosts were configured in a client - server manner. Performance was measured by timing a file transfer of 13.8MB size from the server to the client. The transfer time was measured 30 times per test case and its average time was calculated.

LOT's behavior was tested on various scenarios and its performance was compared with respect to TCP communication and IPsec VPN tunnel with null encryption (message authentication only) using openswan 2.6.15 IPsec implementation [24].

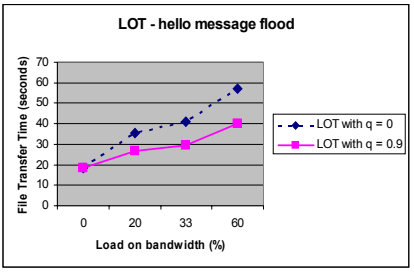
The following subsections describe the various test cases.

6.1 Communication under Legitimate Load

In this set of runs we tested how a LOT tunnel preforms under communication load by legitimate end hosts within the two networks. Figure 5(a) compares



(a) TCP, LOT and IPsec handling tunneled packets load



(b) LOT gateway flooded with (spoofed) handshake hello packets

Fig. 5. Experiments Results Graphs

LOT's performance to TCP (no tunneling) and an IPsec VPN tunnel (authentication only). Both gateways tunneled the communication using IPsec, LOT or simply forwarded it (TCP). Figure 5(a) shows that LOT computational overhead is rather small comparing to IPsec, but larger than no tunneling at all. The more packets sent between the networks, the higher the load on the network gateways and the more difference in performance between TCP, LOT and IPsec. To load the bandwidth, we sent 100 byte UDP packets from Bob (client) to Alice (server) at different rates and timed the 13.8MB file transfer time from the server to the client. Before running the tests on LOT and IPsec, we assured the tunnels were already set up to avoid initial overhead.

6.2 LOT under DoS Attacks

Next we investigated how LOT gateways perform under DoS attacks. Again we measured the file transfer average time and used it as an indicator to LOT's performance.

We tested LOT's performance when one of the tunneling gateways is flooded with spoofed handshake hello messages, i.e. when an attacker sends a hello message (see message 1 in Figure 3) to a LOT gateway specifying a spoofed IP source address and a spoofed network block. Essentially the validation process would fail in such a scenario and no LOT tunnel will be established, however such packets cause the victim to output LOT hello response messages (message 2 in Figure 3). We used GW2 to flood GW1, each hello message specified a random network block (containing the spoofed source address). Initially we tested the attack influence when $q = 0$ meaning, every hello message was replied. Then we conducted the test again using $q = 0.9$ when an expected one out of every ten hello messages causes the victim to output a reply. The results are illustrated in Figure 5(b). Notice the significant influence of the attack when $q = 0$.

Acknowledgments

Thanks to Amit Klein, Yaron Sheffer and the anonymous referees for helpful comments and suggestions.

References

[1] Aharoni, M., Hidalgo, W.M.: Cisco SNMP configuration attack with a GRE tunnel (2005), http://www.securityfocus.com/infocus/1847
[2] Badishi, G., Herzberg, A., Keidar, I.: Keeping denial-of-service attackers in the dark. IEEE Trans. Dependable Sec. Comput. 4(3), 191–204 (2007)
[3] Badishi, G., Herzberg, A., Keidar, I., Romanov, O., Yachin, A.: An empirical study of denial of service mitigation techniques. In: IEEE Symposium on Reliable Distributed Systems, pp. 115–124 (2008), http://doi.ieeecomputersociety.org/10.1109/SRDS.2008.27 ISSN 1060-9857
[4] Baker, F., Savola, P.: Ingress Filtering for Multihomed Networks. RFC 3704 (Best Current Practice) (March 2004), http://www.ietf.org/rfc/rfc3704.txt
[5] Bellovin, S.M.: Security problems in the TCP/IP protocol suite. Computer Communication Review 19(2), 32–48 (1989)
[6] Bernstein, D.J.: TCP SYN cookies (1996), http://cr.yp.to/syncookies.html
[7] Beverly, R., Bauer, S.: The spoofer project: Inferring the extent of source address filtering on the Internet. In: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop table of contents, p. 8. USENIX Association, Berkeley (2005)
[8] Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (August. 2008), http://www.ietf.org/rfc/rfc5246.txt
[9] Dommety, G.: Key and Sequence Number Extensions to GRE. RFC 2890 (Proposed Standard) (September 2000), http://www.ietf.org/rfc/rfc2890.txt
[10] Eddy, W.: TCP SYN Flooding Attacks and Common Mitigations. RFC 4987 (Informational) (August 2007), http://www.ietf.org/rfc/rfc4987.txt
[11] Farinacci, D., Li, T., Hanks, S., Meyer, D., Traina, P.: Generic Routing Encapsulation (GRE). RFC 2784 (Proposed Standard) (March 2000), http://www.ietf.org/rfc/rfc2784.txt (Updated by RFC 2890)
[12] Ferguson, P., Senie, D.: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice) (May 2000), http://www.ietf.org/rfc/rfc2827.txt (Updated by RFC 3704)
[13] Harris, B., Hunt, R.: TCP/IP security threats and attack methods. Computer Communications 22, 885–897 (1999)
[14] IANA. Special-Use IPv4 Addresses. RFC 3330 (Informational) (September 2002), http://www.ietf.org/rfc/rfc3330.txt
[15] Jiang, G.: Multiple vulnerabilities in SNMP. Computer 35(4), 2–4 (2002)
[16] Kaufman, C.: Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard) (December 2005), http://www.ietf.org/rfc/rfc4306.txt (Updated by RFC 5282)
[17] Kent, S., Seo, K.: Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard) (December 2005), http://www.ietf.org/rfc/rfc4301.txt

[18] Killalea, T.: Recommended Internet Service Provider Security Services and Procedures. RFC 3013 (Best Current Practice) (November 2000), http://www.ietf.org/rfc/rfc3013.txt

[19] Lemon, J.: Resisting SYN flood doS attacks with a SYN cache. In: Leffler, S.J. (ed.) BSDCon, pp. 89–97. USENIX (2002), http://www.usenix.org/publications/library/proceedings/bsdcon02/lemon.html ISBN 1-880446-02-2

[20] Pang, R., Yegneswaran, V., Barford, P., Paxson, V., Peterson, L.: Characteristics of internet background radiation. In: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, pp. 27–40. ACM, New York (2004)

[21] Peng, T., Leckie, C., Ramamohanarao, K.: Survey of network-based defense mechanisms countering the doS and DDoS problems. ACM Comput. Surv. 39(1) (2007), http://doi.acm.org/10.1145/1216370.1216373

[22] Rescorla, E., Modadugu, N.: Datagram Transport Layer Security. RFC 4347 (Proposed Standard) (April 2006), http://www.ietf.org/rfc/rfc4347.txt

[23] Richardson, M., Redelmeier, D.H.: Opportunistic Encryption using the Internet Key Exchange (IKE). RFC 4322 (Informational) (December 2005), http://www.ietf.org/rfc/rfc4322.txt

[24] Wouters, P., Bantoft, K.: Building and Integrating Virtual Private Networks with Openswan. Packt Publishing (2006)