Mitigation of Unsolicited Traffic across Domains with Host Identities and Puzzles

Miika Komu[1], Sasu Tarkoma[2], and Andrey Lukyanenko[1]

[1] Aalto University
[2] University of Helsinki

Abstract. In this paper, we present a general host identity-based technique for mitigating unsolicited traffic across different domains. We propose to tackle unwanted traffic by using a cross-layer technique based on the Host Identity Protocol (HIP). HIP authenticates traffic between two communicating end-points and its computational puzzle introduces a cost to misbehaving hosts. We present a theoretical framework for investigating scalability and effectiveness of the proposal, and also describe practical experiences with a HIP implementation. We focus on email spam prevention as our use case and how to integrate HIP into SMTP server software. The analytical investigation indicates that this mechanism may be used to effectively throttle spam by selecting a reasonably complex puzzle.

1 Introduction

One challenge with the current Internet architecture is that it costs very little to send packets. Indeed, many proposals attempt to introduce a cost to unwanted messages and sessions in order to cripple spammers' and malicious entities' ability to send unsolicited traffic. From the network administration viewpoint, spam and DoS traffic comes in two flavors, *inbound* and *outbound* traffic. Inbound traffic originates from a foreign network and outbound traffic is sent to a foreign network. Typically, spam and packet floods originate from networks infested with zombie machines. A *zombie* machine is a host that has been taken over by spammers or persons working for spammers, e.g., using Trojans or viruses.

We address the problem of unsolicited network traffic. We use two properties unique to the *Host Identity Protocol (HIP)* protocol: First, hosts are authenticated with their public keys which can be used for identifying well-behaving SMTP servers. Second, a computational puzzle introduces a cost to misbehaving hosts. Our approach has a *cross-layer* nature because a lower-layer security protocol is used to the benefit of higher-layer protocols.

2 Host Identity Protocol

The Host Identity Protocol (HIP) [9] addresses mobility, multi-homing, and security issues in the current Internet architecture. HIP requires a new layer in

T. Aura, K. Järvinen, and K. Nyberg (Eds.): NordSec 2010, LNCS 7127, pp. 33–48, 2012.

the networking stack, logically located between the network and transport layers, and provides a new, cryptographic namespace. HIP is based on *identifier-locator split* which separates the *identifier* and *locator* of an Internet host. The identifier uniquely names the host in a cryptographic namespace, and the locator defines a topological location of the node. Communication end points are identified using public cryptographic keys instead of IP addresses. The public keys used for HIP are called *Host Identifiers (HIs)* and each host generates at least one HI for itself.

The HIs can be published as separate HIP-specific records in the DNS [11]. Legacy applications can use HIP transparently without any changes. Typically, the application calls the system resolver to query the DNS to map the host name to its corresponding address. If a HIP record for the host name does not exist, the resolver returns a routable IPv4 or IPv6 address. Otherwise, the resolver returns a Host Identifier fitted into an IPv4 or IPv6 address. *Local-Scope Identifier (LSI)* is a virtual IPv4 address assigned locally by the host and it refers to the corresponding HI. *Host Identity Tag (HIT)* is an IPv6 address derived directly from the HI by hashing and concatenation. An LSI is valid only in the local context of the host whereas a HIT is statistically globally unique.

When an application uses HIP-based identifiers for transport-layer communications, the underlying HIP layer is invoked to authenticate the communication end-points. This process is called the *base exchange*, during which the end points authenticate to each other using their public keys. The host starting the base exchange, the initiator, is typically a client, and the other host, the responder, is typically a server. During the base exchange, the initiator has to use a number of CPU cycles to solve a computational puzzle. The responder can increase the computational difficulty of the puzzle to throttle new incoming HIP sessions. Upon successful completion, both end-hosts create a session state called *HIP association.*

The base exchange negotiates an end-to-end tunnel to encapsulate the consecutive transport-layer traffic between the two communicating end-hosts. The tunnel is required because routers would otherwise discard traffic using virtual, non-routable identifiers. Optionally, the tunnel also protects transport-layer traffic using a shared key generated during the base exchange. By default, the tunnel is based on IPsec [7] but S-RTP [14] can be used as well. It should be noted that a single tunnel can encompass multiple transport-layer connections.

With HIP, transport-layer connections become more resilient against IP address changes because the application and transport layers are bound to the location-independent virtual identifiers, HITs or LSIs. The HIP layer handles IP-address changes transparently from the upper layer using the *UPDATE* procedure [10]. In the first step of the procedure, the end host sends all of its locators to its connected peers. Then, the peers initiate so called *return routability test* to protect against packet-replay attacks, i.e., to make sure that the peer locator is correct. In the test, each node sends a nonce addressed to each of the received peer locators. The peer completes the test by signing each nonce and echoing it back to the corresponding peer. Only after the routability test is successfully completed, the peer can start using the locator for HIP-related communications.

HIP sessions can be closed using the *CLOSE* [9] mechanism. It is consist of two packets, in which one of the peer sends a CLOSE message to the other, which then acknowledges the operation using CLOSE-ACK. After this, all state is removed and the tunnel is torn down on both sides.

HIP employs *rendezvous servers* [5] to address the *double jump* problem. This occurs when two connected HIP hosts lose contact with each other when they are relocated simultaneously to new networks. The rendezvous server has a stable IP address and offers a stable contact point for the end hosts to reach each other.

The computational puzzles of HIP [1] play a major role in this paper and have been investigated by others as well. Beal et al. [3] developed a mathematical model to evaluate the usefulness of the HIP puzzle under steady-state DDoS attacks. They also stated that the difficulty of the DoS-protection puzzle should not be too high because otherwise an attacker can just choose a cheaper method such as simple flooding of the network. Tritilanunt et al. [13] explored HIP puzzles further with multiple adversary models and variable difficulties. They also noticed that solving of HIP puzzles can be distributed and a non-distributable puzzle algorithm would provide more resilience against DDoS. Our work differs from Beal et al. and Tritilanunt et al. because our use case is spam rather than DDoS and our approach is based on cross-layer integration.

3 System Model

The basic idea is to assign each node in the network with an identity based on a public key. The hosts may generate their private keys by themselves, or a third party can assign them. Computational puzzles are a well-known technique for spam prevention [4,2,6] but are typically used on a per message basis. In our case, puzzles are applied to each pair of Host Identifiers. The difficulty of the puzzle is varied based on the amount of unwanted traffic encountered.

Our example use case for the technique is spam prevention. Typical spam prevention techniques are applied in a sequence starting from black, white or gray listing techniques and sender identification, and ending in content filtering. Our approach involves a similar sequence of spam testing but relies on the identity of the sender rather than its IP address.

3.1 Basic Architecture for Spam Mitigation

In the email systems deployed in the Internet, there are *outbound email servers* which are used for sending email using SMTP. Typically, the users access them either directly or indirectly with a web-based email client. Usually users are authenticated to these services with user names and passwords. In many cases, direct access to outbound SMTP servers is restricted to the local network as a countermeasure against spam. However, spam is still a nuisance and there are networks which still allow sending of spam. In this paper, we use the term *spam* *relay* for a malign or compromised outbound email server that allows sending spam, and the term *legitimate relay* for a well-behaving outbound email server.

Correspondingly, *inbound email servers* process incoming emails arriving from outbound emails servers. Users access these servers either indirectly via web interfaces or directly with protocols such as POP or IMAP. Typically, the inbound email server tags or drops spam messages and also the email client of the user filters spam messages.

Our idea in a nutshell is to require a HIP session with an SMTP server before it will deliver any email. The sender has to solve a computational puzzle from the server to establish the session. If the sender sends spam, the server ends the HIP session after a certain spam threshold is met. To continue sending spam, the sender has to create a new session, but this time it will receive a more difficult puzzle from the server.

The proposed architecture follows the existing SMTP architecture but requires some changes. First, the inbound and outbound SMTP servers have to be HIP capable. Second, we assume the spam filter of the inbound server is modified to control the puzzle difficulty. Third, we assume the inbound SMTP servers publish their Host Identifiers in the DNS.

3.2 Deployment Considerations

A practical limitation in our approach is that HIP itself is not widely deployed. Even though we compare the HIP-based approach to the current situation later in this paper, the benefits of our design can be harnessed to their full extent only when HIP, or a similar protocol, has been deployed more widely in the Internet. Alternatively, our design could be applied to some other system with built-in HIP support such as HIP-enabled P2P-SIP [8].

We assume that Host Identities are published in the DNS which requires some additional configuration of the DNS and also SMTP servers. However, based on our operational experience with HIP, this can be accomplished in a backward-compatible way and also deployed incrementally. First, the DNS records do not interfere with HIP-incapable legacy hosts because the records are new records and thus not utilized by the legacy hosts at all. Second, *bind*, a popular DNS server software, does not require any modifications to its sources in order to support DNS records for HIP. Third, SMTP servers can utilize a local DNS proxy [12] to support transparent lookup of HIP records from the DNS.

3.3 Pushing Puzzles to Spam Relays

We considered two implementation alternatives for pushing puzzle computation cost to spam relays. In the first alternative, the UPDATE messages could be used to request a solution to a new puzzle. However, this is unsupported by the current HIP standards at the moment. In the second alternative, which was chosen for the implementation, inbound servers emulate puzzle renewal by terminating the underlying HIP session. The termination is necessary because current HIP specifications allow puzzles only in the initial handshake. When the spam relay reconnects using HIP, a more difficult puzzle will be issued by the server.

3.4 Re-generating a Host Identity

One obvious way to circumvent the proposed mechanism is to change to a new Host Identity after the server closes the connection and increases the puzzle difficulty. Fortunately, creating Host Identities is comparable in cost to solving puzzles, which can discourage rapid identity changes. In addition, non-zero puzzle computation time in the initial session further discourages creation of new identities.

3.5 Switching Identities

It is reasonable to expect that a server relaying spam is able to generate new host identities. Let C_K denote a key-pair generation time and C_N the cost of making the public key and the corresponding IP address available in a lookup service. We expect a spam relay to reuse its current identity as long as the following equation holds:

$$C_j < C_K + C_N + C_0, \qquad (1)$$

where C_j is the puzzle computation time of the jth connection attempt. In other words, the spam relay continuously evaluates whether or not to switch to a new identity. If the next puzzle cost is greater than the initial cost, the spam relay has motivation to switch the identity. We note that the spam relay may devise an optimal strategy if the cost distribution is known.

When the puzzle cost is static, there is no incentive for the spam relay to change its identity unless blacklisted because the cost would be greater due to the C_K and C_N terms. For a dynamic cost, the spam relay is expected to change identities when the cost of a new identity and a new connection is less than maintaining the current identity and existing connection. For a DNS-based solution, the C_N term has a high value because DNS updates are slow to take effect.

Our proposed approach addresses identity-switching attacks using three basic mechanisms. First, a node must authenticate itself. This means that the node must be able to verify its identity using the corresponding private key. This does not prevent the node from using multiple identities or changing its identity, but ensures that the key pair exists. Second, a node must solve a computational puzzle before any messages are transported.

Third, a level of control is introduced by the logically centralized lookup service. The DNS maps host names to identities and IP addresses. A node must have a record in the lookup service. The limitation of this approach that control is introduced after something bad (e.g. spam) has already happened. The bad reputation of malicious nodes can be spread with, for example, DNSBL lookups performed by SMTP servers.

Nevertheless, identity switching could used to reduce the proposed system and, therefore, we have taken it into account in the cost model analysis of the next section.

4 Cost Model

In this section, we present an analytical cost model for the proposed identity-based unsolicited traffic prevention mechanism. We analyze the performance of the proposed mechanism when a number of legitimate senders and spam relays send email to an inbound SMTP server. It should be noted that our model excludes puzzle delegation in the case of multiple consecutive relays because it is not advocated by the HIP specifications.

4.1 Preliminaries

Let us consider a set of N_L legitimate email relays and N_S spam relays. Each legitimate relay sends messages at the rate of λ_L messages per second and each spam relay at λ_S. We assume that the inbound email server has a spam filtering component. It has a false negative of probability α, which refers to undetected spam. Thus, $(1-\alpha)$ gives the probability for detecting a spam message. The filter has also a false positive of probability β, which denotes good emails classified as spam. Even though the inbound server could reject or contain the spam, we assume the server just tags the message as spam and passes it forward.

An inbound SMTP server has a spam threshold κ given as the number of forwarded spam messages before it closes the corresponding HIP session. After the session is closed, the outbound email relay has to reopen it. Let the number of reopened sessions be ξ in case of spam relays, and η in case of legitimate email relays. The base exchange has an associated processing cost for the SMTP source, T_{BE}, given in seconds. This processing cost includes also the time spent in solving the puzzle. Let T_M denote the forwarding cost of a message. The finite time interval T, for which we inspect the system, is expressed in seconds.

4.2 Cost Model

To demonstrate scalability, we derive the equation for the load of the inbound SMTP server with and without HIP. The server load is determined by the number of HIP sessions at the server and the number of email messages forwarded. Without HIP, the email processing cost in seconds at the server is

$$R_N = T \cdot T_M \cdot (N_L \cdot \lambda_L + N_S \cdot \lambda_S). \tag{2}$$

In case of HIP, let us define the accumulated puzzle computation time function $G(\xi) = \sum_{i=0}^{\xi} C_i$. First, we consider the case with constant puzzle computation time that is independent of number of session resets, i.e. $C_1 = C_2 = \ldots C_N = T_{BE}$, and $G(\xi) = \xi \cdot T_{BE}$.

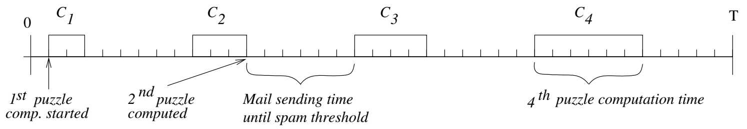

0
c_1
c_2
c_3
c_4
T
1st puzzle comp. started
2nd puzzle computed
Mail sending time until spam threshold
4th puzzle computation time

Fig. 1. Division of system inspection time (T) into puzzle re-computation and mail delivery stages with different puzzle computation times c_i, where i is the number of session resets. All $C_i = T_{BE}$ if the puzzle computation time is constant.

Next, we derive the number of HIP sessions due to session resets caused by spam under the condition of identical puzzle computation time. The following equation presents the number of session resets for a single spam relay:

$$\xi = \frac{(T - T_{BE} \cdot \xi)\lambda_S(1-\alpha)}{\kappa} \tag{3}$$

From equation 3, we can deduce that

$$\xi = \frac{T \cdot \lambda_S \cdot (1-\alpha)}{\kappa + \lambda_S \cdot (1-\alpha) \cdot T_{BE}}. \tag{4}$$

The equation for the number of HIP sessions η needed by the legitimate SMTP relays is similar to equation 4, but the false positive rate β is used instead of $(1-\alpha)$, and correspondingly λ_L is used instead of λ_S. We assume that legitimate relays do not send significant amount of spam so that only false positives need to be considered. The cost to a paying customer is given by η, and ξ is the cost to a spam relay. Given a small false positive probability, η is small. Therefore, the mechanism is not harmful to paying customers.

Next, we derive the equation describing the HIP load of the inbound server R_H consisting of both legitimate and spam messages:

$$R_H = N_L \cdot (\eta \cdot T_{BE} + T \cdot \lambda_L \cdot T_M) + N_S \cdot (\xi \cdot T_{BE} + T_S \cdot \lambda_S \cdot T_M), \tag{5}$$

The equation can be simplified by substituting the total time used for sending spam messages T_S with $T - T_{BE} \cdot \xi$ and by applying equation 2:

$$R_H = R_N - T_M \cdot T_{BE} \cdot (\lambda_S \cdot N_S \cdot \xi) + T_{BE} \cdot (N_L \cdot \eta + N_S \cdot \xi) \tag{6}$$

We assume β is small and, therefore, we used T instead of $T - T_{BE} \cdot \eta$ (with η denoting the number of session resets for a legitimate host). To evaluate the effectiveness of the HIP-based solution against a solution without HIP, we define ratio φ as:

$$\varphi = \frac{R_H}{R_N}. \tag{7}$$

Now, consider the case when puzzle computation time is not constant, but rather a function of the number of session attempts. This has to be reflected in equation 4, which becomes

$$\kappa \cdot \xi + G(\xi)\lambda_S(1-\alpha) - T \cdot \lambda_S(1-\alpha) = 0. \tag{8}$$

The equation can be solved using numerical iteration.

4.3 A Comparison of HIP with Constant Puzzle Cost to the Scenario without HIP

For numerical examples, we use HIP base exchange measurements obtained from an experimental setup described further in Section 5. We plot the ratio of non-HIP versus HIP approaches φ shown in equation 7. The HIP base exchange with a 10-bit puzzle was measured to take 0.215 s of HIP responder's time and 0.216 s for the initiator. We note that our analysis excludes the impact of parallel network and host processing. The email forwarding overhead without HIP is set to 0.01 seconds. We assume that the false negative probability of the server is 1/3 and the false positive probability is $1/10^4$. In other words, 2/3 of spam messages will be correctly detected as spam, and good messages are rarely classified as spam. Let N_L be 10^4, N_S be 100, $\lambda_L = 1/360$, and $\lambda_S = 10$. The time-period T for the analysis is 24 hours.

Figure 2 presents the ratio of HIP versus non-HIP computational cost as a function of the puzzle computation time. Both x and y axes are logarithmic. Ratio in the figures denotes φ, the ratio of the HIP and non-HIP capable mechanisms. The point at which the HIP mechanism has less overhead is approximately at 2 seconds. This means that the proposed HIP mechanism becomes superior to the constant non-HIP benchmark case with an 2-second or greater puzzle computation time. Naturally, this point depends on the selection of the values for the parameters.

As the spam relay sending rate increases, the HIP spam prevention mechanism becomes considerably better than the non-HIP benchmark case. With low spam rates, HIP sessions are reset seldomly and spam flows mostly through. When the spam rate increases, the spam relay spends more time on puzzle computation and the spam forwarding rate decreases. Then, the performance of the proposed HIP mechanism improves in comparison to the non-HIP benchmark case.

4.4 A Comparison of HIP with Exponential Puzzle Cost the Scenario without HIP

We also analyze the scenario where the puzzle cost grows exponentially for each new session. The parameters are the same as before, but the computation time of the puzzle grows exponentially with the puzzle difficulty. Moreover, we introduce a *cut-off point* after which the puzzle difficulty does not increase anymore. After the number of sessions reaches the cut-off point, the computation time of the puzzle (and the number of bits) remains at the current level. As an example, given a cut-off point of 23 and an initial puzzle size of 20 bits for the first throttled session, spam relays experience puzzle sizes $\{20, 21, 22, 23, 23, \dots\}$ as they reconnect.

Figure 3 presents the effect of the exponential base exchange time with a varying cut-off point. The y axis is logarithmic. The figure shows that the proposed mechanism performs considerably better than the non-HIP benchmark with a cut-off point of 22 or greater.

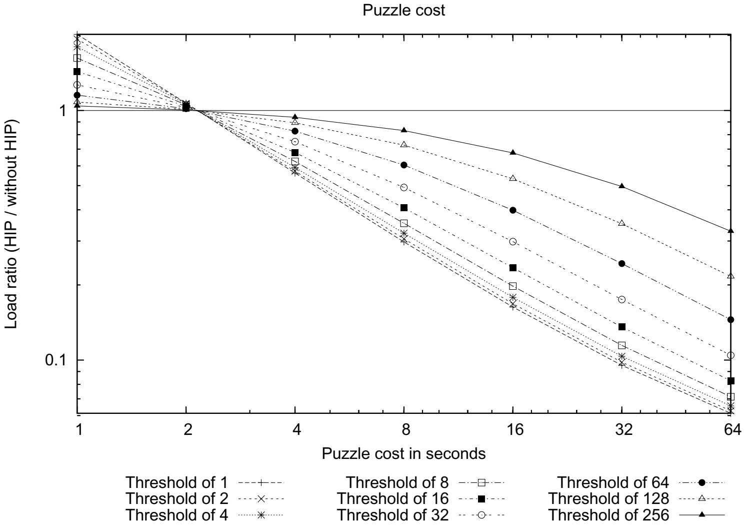


Fig. 2. Fixed-cost puzzles with different spam threshold κ

Now, we have compared HIP with both constant and variable-sized puzzles to the benchmark scenario without HIP. In the next sections, we focus on the identity-switching attacks (without cut-off points) against the proposed HIP-based architecture.

4.5 Optimal Strategies for a Spam Relay

Directly from equation 8, we know that

$$\xi \cdot \frac{\kappa}{\lambda_S \cdot (1-\alpha)} + G(\xi) = T. \tag{9}$$

This means that, for the entire time during which a server relays spam, it splits its performance into ξ steps (one step is one session reset). To contact the inbound server, the spam relay spends $G(\xi)$ time for all puzzle computations, and during every step it sends exactly κ messages and each step consumes $\frac{\kappa}{\lambda_S(1-\alpha)}$ time.

The inbound server chooses the form of the function G, while a spam relay selects the number of session resets to tolerate, ξ. Here, we consider first a naive strategy for the spam relay. It chooses G based on the number of messages to send and does not try to whitewash its own history at the inbound server (i.e. by changing its identity according to equation 1). Under such an assumption, the spam relay has to optimize (maximize) a function of the following form:

$$p_Z \cdot \kappa \cdot \xi - c_Z \cdot G(\xi), \tag{10}$$

where p_Z is the profit for one delivered message and c_Z is the payment for the puzzle computation time. The strategy for the spam relay is to select

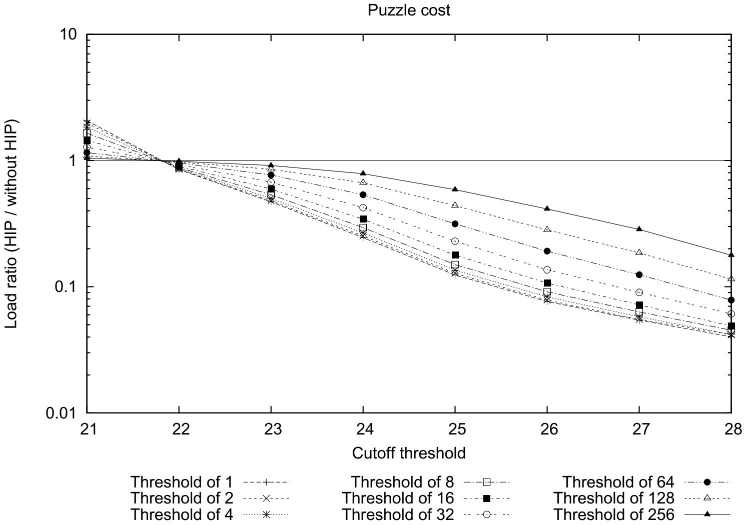


Fig. 3. Variable-sized puzzles with initial puzzle size of 20 and different cut-off points

the number of rounds for which it would like to send κ-sets of messages and the number of rounds to recompute puzzles. Let this value be ξ.

Note that if puzzle difficulty is constant (i.e. $G(\xi) = T_{BE} \cdot \xi$), then solution is one of the boundary cases

$$\xi = \begin{cases} 0, & \text{if } p_Z\kappa \leq c_Z \cdot T_{BE}, \\ \infty, & \text{if } p_Z\kappa > c_Z \cdot T_{BE}, \end{cases} \tag{11}$$

More important is the case when the puzzle computation time is changing. Let the puzzle complexity growth be exponential compared to the increase of puzzle difficulty. Consider that the puzzle computation time on every reset has an exponential form of $C_i = aq^i + b$, then by definition

$$G(\xi) = \sum_{i=0}^{\xi}(aq^i + b) = a\frac{q^{\xi+1} - 1}{q - 1} + b = \frac{aq}{q-1}q^{\xi} + b - \frac{a}{q-1}. \tag{12}$$

Let us generalize this function as $G(\xi) = kg^{\xi} + s$, where g is an exponential growth parameter, s is initial shift, and k is the coefficient.

Now, a spam relay has to maximize the function

$$p_Z \cdot \xi \cdot \kappa - c_Z \cdot (k \cdot g^{\xi} + s). \tag{13}$$

Let us find the points where the derivative of this function with respect to ξ is zero:

$$p_Z \cdot \kappa - c_Z \cdot k \cdot \ln g \cdot g^{\xi} = 0. \tag{14}$$

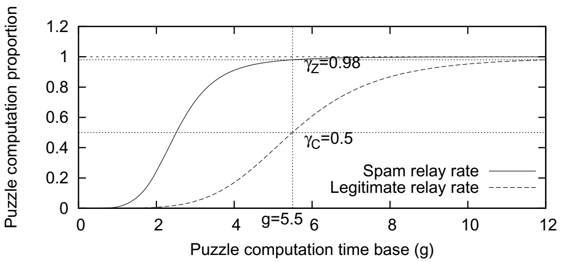


Fig. 4. An example plot to illustrate the proportion of time used by a legitimate and spam relay for puzzle computation

Thus, the maximum point is

$$\xi^* = \log_g \frac{p_Z \cdot \kappa}{c_Z \cdot k \cdot \ln g}. \tag{15}$$

4.6 Optimal Strategies for an Inbound Server

The previous section suggests an optimal strategy for a spam relay under the assumption that there is a payment involved in sending of spam. Otherwise, infinite number of messages would be the optimal strategy for the spam relay. In this section, we have a look at the situation from the view point of an inbound server.

First of all, the main goal for the inbound server is to slow down the flood of spam. It may be formulated in terms of the portion of time which spam relays spend for the puzzle computation time, compared to the overall time. Here, we assume that the inbound relay knows the number of HIP session resets during which spammer reuses its current identity according to equation 1. As previously, let it be ξ. To process ξ resets, a spam relay has to waste $G(\xi)$ of its own time for puzzle computation. The overall time, which it may use for message delivery, we also define as a function of ξ. Thus, the definition of the overall time follows from equation 9

$$T(\xi) = \xi\frac{\kappa}{d} + G(\xi), \tag{16}$$

where d is equal to $\lambda_Z(1-\alpha)$ in case of a spam relay, and is equal to $\lambda_C\beta$ in case of a legitimate email relay. We assume that an inbound server classifies (or receives classification) with relatively good accuracy and, hence, $1-\alpha$ is considerably higher than β.

Then, the proportion of time used for puzzle computation by spam relays (on left side) and legitimate email relays (on the right side) can be calculated as

$$\frac{G(\xi)}{G(\xi) + \frac{\kappa\cdot\xi}{\lambda_Z\cdot(1-\alpha)}}, \qquad \frac{G(\xi)}{G(\xi) + \frac{\kappa\cdot\xi}{\lambda_C\cdot\beta}}. \tag{17}$$

The inbound server has control over variables k, g, s of function $G(\xi) = kg^{\xi} + s$. For simplicity, let k and s be constants because the most relevant variable is the growth base g for the puzzle computation time. The values grow as a function of the parameter g. The function results in values ranging from 0 to 1, where 0 means that the time spent for the puzzle computation is negligible, while 1 means that the puzzle computation takes all of the time.

For the functions (17), the objective of the inbound server is to maximize the time spam relays spend on computing puzzles. Correspondingly, the inbound server should minimize this time for legitimate relays. These are somewhat contradictory conditions because $\alpha < 1$ and $\beta > 0$. Therefore, punishment for possible spam relays affects also legitimate relays.

To overcome this dilemma, we introduce a new constant γ: $0 \leq \gamma \leq 1$, which we select as the maximum value for the possibly legitimate client computation rate, i.e.

$$\frac{G(\xi)}{G(\xi) + \frac{\kappa \cdot \xi}{\lambda_C \cdot \beta}} \leq \gamma, \tag{18}$$

where γ defines the portion of the overall time which a possibly legitimate client spends for puzzle computations. From the inequality 18 it follows, that

$$g \leq \left(\frac{\gamma \cdot (\kappa \cdot \xi + s \cdot \lambda_C \cdot \beta)}{k \cdot \lambda_C \cdot \beta \cdot (1 - \gamma)} \right)^{\frac{1}{\xi}} . \tag{19}$$

On the other hand, the inbound server should maximize puzzle computation rate for possible spam relays (the left function in equation 17, which grows exponentially towards 1 as a function of g). The optimal strategy for the server is

$$g^* = \left(\frac{\gamma \cdot (\kappa \cdot \xi + s \cdot \lambda_C \cdot \beta)}{k \cdot \lambda_C \cdot \beta \cdot (1 - \gamma)} \right)^{\frac{1}{\xi}} . \tag{20}$$

The optimal strategy both for a spam relay, $\xi^*(g)$, as shown in equation 15, and for an inbound server, $g^*(\xi)$, as shown in equation 20, results in an equilibrium point (ξ^*, g^*) in terms of game theory.

The optimal strategies are illustrated in figure 4. For the legitimate relay, the bound for the computation rate is fixed as $\gamma_C = 0.5$ The set of parameters is assigned as $\alpha = 0.5$, $\beta = 0.01$, $\kappa = 100$, $\lambda_C = \lambda_Z = 10$, and we assume that the number of session resets is 5 ($\xi = 5$). Under such parameters, the legitimate relay has $g \approx 5.5$. The resulting puzzle computation for a possible spam relay is $\gamma_Z = 0.98$. In other words, the spam relay spends 0.98 of its time for puzzle computations whereas the legitimate relay spends half of its time. As g grows, both parties are eventually spending all of their time for puzzle computation. Thus, it is a local policy for the inbound server to decide a "good" value for g in terms of how much legitimate servers can be throttled with puzzles. For low spam rates, the value can be small but, with high spam rates, the server may increase the value at the cost of throttling also legitimate relay servers.

5 Experimental Evaluation

In this section, we describe how we integrated puzzle control to an inbound SMTP server and its spam filtering system. We show some measurements with variable-sized puzzles and compare this against identity-generation costs to give some engineering guidance against identity-switching attacks. The source code for HIP for Linux and the spam extensions are available as open source [1]. It should be noted that evaluation the mathematical models presented in section 4 e.g. with network simulators is future work.

5.1 Setup

The experimented environment consisted of two low-end commodity computers with the Linux Debian distribution and HIP for Linux (HIPL) [12] implementation. One computer served as a sending SMTP relay (1.60GHz Pentium M) and the other represented a receiving SMTP server (Pentium 4 CPU 3.00GHz). The receiving server detects the spam messages and closes the HIP session when a threshold is reached for the session. The inbound server was configured not to reject any email. We were mostly interested in software changes required to deploy HIP in SMTP servers and in the effects of increasing the puzzle size.

5.2 Results

We implemented the spam throttling mechanism successfully by using unmodified sendmail. We turned on the IPv6 option in the configuration of sendmail in order to use HITs.

The receiving SMTP server was equipped with a modified version of *SpamAssassin Milter*. The changes were straightforward to implement. The milter increased the puzzle size by one for every κ spam message detected and closed the HIP session to induce a new base exchange. The puzzle computation time grew exponentially with the size of the puzzle and the spam sender was throttled, as expected, by the mechanism.

We faced some implementation challenges during the experimentation. Firstly, sendmail queues the email messages and this makes it difficult to provide measurements from the spam filtering process itself. Secondly, if the session with the SMTP server is lost temporary, for example, because the HIP association are is closed, e-mails can accumulate in the queue for an extended time. Thirdly, when sending excessive amounts of email, the built-in connection throttling mechanism in sendmail takes over and queues the emails for long periods. However, sendmail's queuing process was robust and eventually emptied the queue successfully.

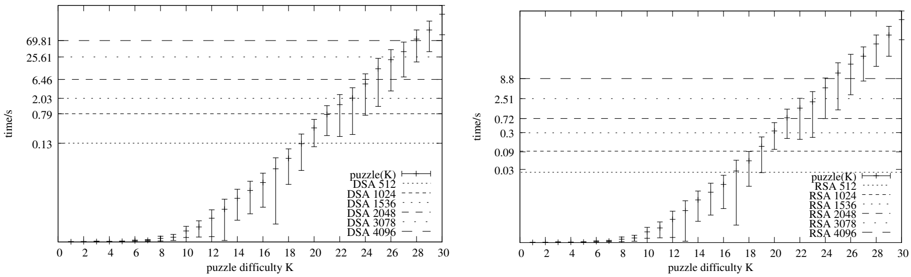
One challenge with proof-of-work techniques is that there are many different devices on the network and their computing capabilities vary. By default, the puzzle difficulty is zero in HIPL. A puzzle with difficulty of 25 bits took 12.4 s on

[1] `https://launchpad.net/hipl/`

average on the low-end machine used in the performance tests. The time was 4.4 seconds on a more recent CPU (Intel Core 2, 2.3 Mhz) on a single CPU core. The puzzle algorithm used in HIP does not prevent parallel computation. Thus, the computation time could be decreased by fully utilizing multi-core architectures.

For identity changing attacks, the strategy should also take into account the public key algorithm. RSA keys can be created faster than DSA keys with a corresponding size. As a consequence, the responder should give initiators that use RSA public keys more difficult puzzles than initiators with DSA keys. Further, it should be noted that creation of insecure, albeit perfectly valid keys, can be faster than creation of secure ones.

Figures 5(a) and 5(b) contrast secure key-pair generation time (horizontal lines) with puzzle solving time (vertical lines). It should be noticed that the y-axis is logarithmic. From the figures, it can be observed that the puzzling solving time is, as expected, exponential with the number of bits used in the puzzle difficulty. The standard deviation grows as puzzle difficulty is increased. In addition, the time to generate DSA key-pairs is considerably higher than RSA. On the average, the creation of a 2048-bit DSA key pair took 6.46 seconds and this was equal to the solving time of a 24-bit puzzle. With RSA, creation of a 2048-bit key pair took 0.72 seconds which corresponded to a 21-bit puzzle. This indicates that the key-generation algorithm and key length need to be taken into account when deciding the initial puzzle size to discourage identity-switching attacks.



(a) Puzzle solving vs. DSA key generation (b) Puzzle solving vs. RSA key generation

Fig. 5. Puzzle computation time results

6 Conclusions

In this paper, we proposed a cross-layer identity solution for mitigating unsolicited traffic between administrative domains. The proposed architecture primarily concentrates on inbound session control but is applicable also to the outbound direction as well. As an example application of the system, we focused on email spam prevention.

The Host Identity Protocol introduces a public key for the hosts. They key can be used for identifying well-behaving SMTP servers. The proposed approach introduces a cost to sending spam using the computational puzzles in HIP. Large-scale changes to the SMTP architecture are not required because HIP is backwards compatible. However, a practical limitation of the approach is that it requires wide-scale adoption of HIP as a signaling protocol and requires integration of HIP puzzle control to inbound email servers.

We presented a formal cost model that considered static and exponential base exchange puzzle costs. The analytical investigation indicates that the proposed spam prevention mechanism is able to mitigate unwanted traffic given a set of reasonable parameters. We used parameter values based on experimental results for server-side cost of HIP and the puzzle computation time. A spam mitigation approach based on HIP puzzles caused less load at the email server than an approach that was not using HIP.

The exponential cost of the puzzle introduces more work for email servers relaying spam. However, it also results in an incentive for the spammer to switch its identity when it is throttled with more difficult computational puzzles. We identified this as a potential weakness of the proposed system and analyzed this from the viewpoint of the spammer and the email server. As a theoretical result, we provided a method for the server to choose an optimal strategy against identity switching. When choosing a strategy, it should be noted that increasing puzzle costs for spammers also increase costs for legitimate hosts.

We implemented a simple prototype of the system based on a popular email server, sendmail. We integrated throttling support for HIP puzzles with minimal changes to SpamAssassin, a popular spam filtering software. We reported the practical experiences of running such a system and showed real-world measurements with HIP puzzles.

While the simple prototype was a success, we observed that the use of computational puzzles with email relays is challenging. Malicious hosts can overwhelm and exhaust the resources of a relay unless preventive measures are taken. Potential solutions to this include refusal to solve large puzzles for hosts, message rejection, and blacklisting. More work with simulation or larger test beds is needed to establish the efficacy of the proposed cross-layer system and to validate our mathematical models.

Acknowledgements. We would like to thank the following people for providing valuable feedback for this paper: Jaakko Kangasharju, Teemu Koponen, Kristian Slavov, Antti Järvinen, Oleg Ponomarev, Andrei Gurtov and Tuomas Aura. This work was supported by Tekes InfraHIP project and the Academy of Finland, grant numbers 135230, 122329 and 135230.

References

1. Aura, T., Nikander, P., Leiwo, J.: Dos-Resistant Authentication with Client Puzzles. In: Christianson, B., Crispo, B., Malcolm, J.A., Roe, M. (eds.) Security Protocols 2000. LNCS, vol. 2133, pp. 170–177. Springer, Heidelberg (2001)

2. Back, A.: Hashcash (May 1997), http://www.cypherspace.org/hashcash/
3. Beal, J., Shepard, T.: Deamplification of DoS Attacks via Puzzles (October 2004), http://web.mit.edu/jakebeal/www/Unpublished/puzzle.pdf
4. Dwork, C., Naor, M.: Pricing via Processing or Combatting Junk Mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993)
5. Eggert, L., Laganier, J.: Host Identity Protocol (HIP) Rendezvous Extension. IETF (April 2008), Experimental RFC
6. Goodman, J., Rounthwaite, R.: SmartProof. Microsoft (2005), http://research.microsoft.com/en-us/um/people/joshuago/smartproof.pdf
7. Jokela, P., Moskowitz, R., Nikander, P.: RFC5202: Using the Encapsulating Security Payload (ESP) Transport Format with the Host Identity Protocol (HIP) Internet Engineering Task Force (April 2008), http://www.ietf.org/rfc/rfc5202.txt
8. Keränen, A., Camarillo, G., Mäenpää, J.: Host Identity Protocol-Based Overlay Networking Environment (HIP BONE) Instance Specification for REsource LOcation And Discovery (RELOAD). Internet Engineering Task Force (July 2010) (internet draft, work in progress)
9. Moskowitz, R., Nikander, P., Jokela, P., Henderson, T.: RFC5201: Host Identity Protocol. Internet Engineering Task Force (April 2008); Experimental RFC
10. Nikander, P., Henderson, T., Vogt, C., Arkko, J.: End-Host Mobility and Multihoming with the Host Identity Protocol. Internet Engineering Task Force (April 2008); Experimental RFC
11. Nikander, P., Laganier, J.: Host Identity Protocol (HIP) Domain Name System (DNS) Extension. IETF (April 2008); Experimental RFC
12. Pathak, A., Komu, M., Gurtov, A.: Host Identity Protocol for Linux. Linux Journal (November 2009), http://www.linuxjournal.com/article/9129
13. Tritilanunt, S., Boyd, C., Foo, E., Nieto, J.M.G.: Examining the DoS Resistance of HIP. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4277, pp. 616–625. Springer, Heidelberg (2006)
14. Tschofenig, H., Shanmugam, M., Muenz, F.: Using SRTP transport format with HIP. Internet Engineering Task Force (August 2006); expired Internet draft