ShutUp: End-to-End Containment of Unwanted Traffic

Saikat Guha, Paul Francis
Cornell University, Ithaca

Nina Taft
Intel Research

Abstract

While the majority of Denial-of-Service (DoS) defense proposals assume a purely infrastructure-based architecture, some recent proposals suggest that the attacking endhost may be enlisted as part of the solution, through tamper-proof software, network-imposed incentives, or user altruism. While intriguing, these proposals ultimately raise the deployment bar by requiring both the infrastructure and endhosts to cooperate. In this paper, we explore the design of a pure end-to-end architecture based on tamper-proof endhost software implemented for instance with trusted platforms and virtual machines. We present the design of a "Shutup Service", whereby the recipient of unwanted traffic can ask the sender to slowdown or stop. We show that this service is effective in stopping DoS attacks, and in significantly slowing down other types of unwanted traffic such as worms. The Shutup service is incrementally deployable with buy-in from OS or antivirus vendors, requiring only minimal changes to the endhost software stack and no changes to the protocol stack. We show through experimentation that the service is effective and has little impact on legitimate traffic.

1 Introduction

An ongoing problem in the Internet is that of unwanted traffic: DoS, worms, and port scanning. A large variety of systems have been deployed or proposed that address one or more of these types of unwanted traffic, including purely infrastructure-based DoS defense systems [57, 56, 2, 23], hybrid endhost- and infrastructure-based DoS defense systems [4, 3], and worm detection and mitigation systems [54, 53, 40]. In spite of this, DoS attacks remain commonplace, and the Internet remains vulnerable to flash worms [31]. The latter is particularly worrisome, because a malicious flash worm could do extensive damage to millions of systems.

In this paper, we explore a new point in the solution design space. In particular, we propose a pure end-to-end *ShutUp Service*, whereby a host that receives packets can tell the sending host to rate limit or stop sending packets to itself. This is enforced at the sending host by a tamper-resistant enforcement agent, for instance implemented in the NIC device or in a protected virtual machine. This ShutUp service can be used in two ways:

1. It is used by the receiver of packets (the *recipient*) to manage DoS attacks and flash crowds. The ShutUp is used as a capability-based rate-control system. After an initial short grace period during which the sender (*initiator*) can send at full speed, the enforcement agent at the initiator imposes a low rate unless the recipient explicitly allows a higher rate. The recipient may request a smaller or zero rate at any time, for instance in response to increased volume.

2. It is used by the enforcement agent at the initiator as a signal that the host may be misbehaving. If the agent receives a significant number of these signals, it responds by slowing the rate at which connections to new hosts can be initiated. This slows down scanning worms, and port scanning (collectively called *scanning attacks* in this paper). In the case of port scanning, the slow down reduces the severity of the problem. In the case of scanning worms, the slow down buys precious time for other mechanisms (e.g. human response) to kick in.

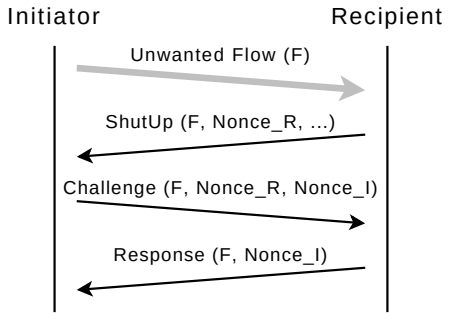

Figure 1: Basic ShutUp operation: Recipient NM sends SHUTUP for unwanted data. Initiator SM validates ShutUp was sent by purported recipient before blocking the flow at the source. Nonces protect against spoofing.

The ShutUp service is implemented through the simple request-challenge handshake shown in Figure 1. While the design of the ShutUp service requires no infrastructure support, our design intentionally allows it to operate between any pair of systems on the physical path of packets between initiator and recipient. This maximizes the deployment options, allowing intervention by the middle

where appropriate. The main rationale for this E2E approach are as follows.

- By putting enforcement at the misbehaving host, we maximize the scenarios whereby the ShutUp service can be effective. For instance, hosts within an enterprise would be protected from each other.
- By allowing recipient endhosts to issue ShutUp requests, we can exploit application-specific knowledge to detect unwanted packets. At the same time, by allowing firewalls or even routers on the data path to issue ShutUp requests, we can exploit the broader knowledge a firewall has due to its vantage point.
- The ShutUp service requires no new changes to existing protocols or to infrastructure equipment. Enterprises already deploy host-based security solutions, such as firewall and anti-virus software, and so the ShutUp deployment model is a natural fit. While we do not want to pretend that globally deploying ShutUp (or any other defense system) is easy, the deployment model for ShutUp is intriguing because it could be accomplished with buy-in from a relatively small number of organizations in one or a few industry segments: OS, firewall, or antivirus vendors. Neither router vendors nor ISPs need be involved.

The deployment model for the ShutUp service is that it would be enabled in hosts by default when they are installed or when software is upgraded. Individual users could opt-out by disabling the ShutUp service enforcement mechanism in their own hosts, but not without some small difficulty or expense. Disabling the ShutUp service doesn't open new threats that don't already exist today. Within an enterprise, IT organizations can arrange to have ShutUp enabled everywhere, thus getting its full benefit. Globally, however, we can expect to see some (hopefully small) percentage of hosts without ShutUp enforcement. This certainly allows individuals to launch attacks from their own hosts, but not from botnets. Since most users should prefer not to be sending unwanted traffic, we would expect the ShutUp service to remain enabled in most hosts. For attacks launched from ShutUp-disabled hosts, other defenses, such as ISP firewalls with ShutUp support, or manual filtering of the attacker's packets, would have to be used.

Although the ShutUp service borrows from recent work in DoS defense systems that exploit endhost support, it is unique in several ways. Argyraki and Cheriton [4] propose the use of a handshake similar to ShutUp, but require enforcement devices in the network near the attacker, and a control device on the attacker-side of the bottleneck resource. By contrast, ShutUp is a pure E2E mechanism. In particular, ShutUp avoids the need for a control device outside of the bottleneck resource through a capability mechanism. To our knowledge, Shaw is the first to propose putting enforcement at the endhost [41]. The approach has some limitations, in particular with respect to source address spoofing and the scoping of ShutUp requests, and in any event only outlines the approach and does not experiment. It is fair to characterize ShutUp as a deep exploration of the same vision. AIP [3] is a new network layer architecture with two-level self-certifying addresses. Among the many uses of this architecture, it can defend against DoS through an E2E handshake with enforcement at the endhost (putting the mechanism in a tamper-proof NIC card). Notably AIP still requires infrastructure support; to detect source address spoofing by attackers. ShutUp works with legacy network layers, does not require cryptographic mechanisms, and prevents source address spoofing at the tamper-resistant driver in the attacking endhost.

This paper makes the following contributions. *First, we present the first design and implementation of a pure E2E ShutUp service.* The ShutUp protocol itself is quite simple, allowing us to confidently reason about its security properties and the correctness of the implementation. The enforcement module, for instance, is implemented in less than 200 lines of Python code. This small footprint also maximizes the deployment options. For instance, the ShutUp protocol may be implemented in a NIC or on small wireless devices. It requires no encryption or key distribution, allowing us to avoid the complexity and hazards of key distribution. *Second, we analyze the ShutUp service's effectiveness as a DoS prevention mechanism.* We show, for instance, that a host with 10Mbps of access bandwidth can completely stop a 10Gbps attack from $\sim$10,000 hosts behind broadband links. *Finally, we analyze the ShutUp service's effectiveness as a defense against scanning attacks.* Of particular interest here is the trade-off between effectiveness (how quickly scanning attacks are discovered and how much they are slowed down) and false positives (identifying legitimate activity as scanning attacks). For example, we show that ShutUp slows down scanning worms to the same degree as existing approaches while reducing false positives by two orders of magnitude.

2 ShutUp Details

This section starts with an overview of the ShutUp components, followed by a detailed description of its operation.

2.1 ShutUp Components

There are two components in ShutUp, *notification modules* (NM) and *ShutUp modules* (SM). As illustrated in

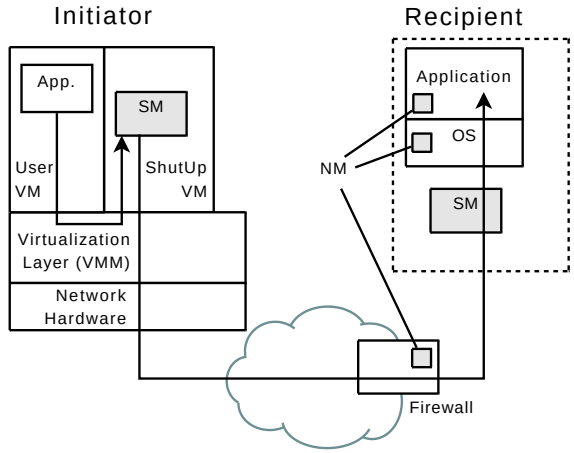

Figure 2: The ShutUp module (SM) runs in a separate tamper-proof VM. Network traffic to and from the user VM is routed through the SM by the VMM. The Notification Module (NM) runs at multiple layers: in the recipient application, endhost OS and recipient firewall. The initiator and recipient endhost setup is symmetric (abbreviated for clarity).

Figure 2, NMs are deployed at the recipient at multiple levels (application, endhost OS, etc.) from where they can analyze inbound traffic at multiple layers. The NM is not a trusted component. A SM is deployed at the initiator, out of reach of untrusted components, from where the SM can exert direct control over inbound and outbound traffic. The recipient NMs are responsible for identifying unwanted flows and sending ShutUp messages, and the initiator SM is responsible for enforcing the recipient's decision and for correct operation of the ShutUp protocol. Since an endhost can both initiate and receive flows, NMs and an SM are deployed in each endhost.

2.1.1 ShutUp Module (SM)

One of the challenges in implementing ShutUp is protecting the SM from being subverted by endhost malware, while at the same time not placing undue constraints on the resources available to it. This can be done by running user software and the SM in separate virtual machines (VM). The virtualization layer allows only the ShutUp VM access to the physical network interfaces. Traffic to and from the user VM is routed through the ShutUp VM (Figure 2), where the SM can filter packets as necessary.

There are a number of options for protecting the SM, for instance in NIC hardware [3] or TPM module [48], but we believe that a VM protected by a VMM [13] is a good choice. First, a VM has access to more resources (multiple cores, memory) than a hardware implementation. On a typical PC we estimate a SM will require around 256kB of state isolated from the user; tamperproof memory in hardware is expensive, whereas a VMM can virtualize main system memory. Second, a VM can be restored to known-good states and updated once deployed. Updating hardware securely requires physical access, whereas booting into the VMM before accessing user programs bootstraps a secure software environment. Existing software update mechanisms can then download, verify and update the SM. Finally, a VM is needed to demultiplex ShutUps for middleboxes. With a hardware SM, a ShutUp for a middlebox (proxy, tunnel endpoint) resulting from traffic initiated by a single user would affect the entire middlebox, thus affecting all users behind the middlebox. In a VM, a trusted version of the middlebox service can run alongside the SM to redirect ShutUps to the offending user.

ShutUp Primitive

SHUTUP(FLOW, APPNAME, NONCE_R, TTL, ID_{NM})

THROTTLE(FLOW, APPNAME, NONCE_R, TTL, RATE, ID_{NM})

Recipient NM directs initiator SM to ShutUp or throttle a flow (5-tuple; wildcards allowed).

CHALLENGE(FLOW, NONCE_R, NONCE_I, ID_{NM})

Initiator SM requests confirmation of request

RESPONSE(FLOW, NONCE_I)

DISCLAIM(FLOW, NONCE_I)

Recipient NM confirms or denies original request

Query Primitive

QUERYAPPNAME(FLOW, NONCE_I)

APPNAME(FLOW, APPNAME, NONCE_I)

SM queries remote NM for application name (to block scanning attacks before probe is sent)

Table 1: ShutUp primitives and message contents

2.2 Basic Operation

ShutUp offers two primitives (Table 1). The primary primitive, SHUTUP, is used to block or rate limit individual flows.

The basic ShutUp operation illustrated in Figure 1 is fairly straightforward. The recipient NM sends a SHUTUP request (unencrypted) in response to unwanted traffic. The request includes a nonce (NONCE_R) for verification purposes. To save state, the nonce may be computed as a cryptographic hash of the flow identifier and a local time-varying secret key. The initiator SM challenges the purported recipient with a second nonce (NONCE_I) and includes the first nonce in the challenge. If NONCE_R is validated, the recipient completes the challenge by returning NONCE_I. Otherwise, the recipient signals an error. The first nonce protects against a spoofed SHUTUP message and replays, while the second nonce protects against a spoofed response. Once the ShutUp is validated, the SM

Algorithm 1 at SM ONRECVFROMAPP(P)

```
1:  if ISUNVALIDATED(P.IP_src, P.MAC_dst) then
2:      rate limit                                   ▷ May be spoofed
3:  if FOREXISTINGFLOW(P) then
4:      F ← GETFLOWSTATE(P)
5:      ADDSRCPORTOFFSET(P, F)                       ▷ Obscure 5-tuple
6:      if ISSHUTUP(P) then                  ▷ App sending ShutUp/Throttle
7:          if RCVDDATASINCESENTSHUTUP(F) then
8:              forward                              ▷ To net
9:          else
10:             drop                                 ▷ Redundant ShutUp
11:     else if FLOWRATELIMITED(F) then
12:         rate limit                               ▷ Set by recipient
13:     else if FLOWBLOCKED(F) then
14:         drop
15:     else if INITIALTHROTTLETIMEOUT(F.IP_dst) then
16:         rate limit                               ▷ Recipient under DoS?
17:     else
18:         forward
19: else                                             ▷ New flow
20:     if ISSHUTUP(P) then
21:         drop                                     ▷ No flow to ShutUp
22:     else if not NEWFLOWALLOWED(P) then
23:         drop                                     ▷ For scanning attacks
24:     else
25:         F ← NEWOUTBOUNDFLOW(P)
26:         ADDSRCPORTOFFSET(P, F)
27:         forward
28: if not WASDROPPED(P) then
29:     UPDATESENTTIMESTAMPS(F, P)
```

Algorithm 2 at SM ONRECVFROMNET(P)

```
1:  SETVALIDATED(P.IP_dst, P.MAC_src)                ▷ Learn IP
2:  if FOREXISTINGFLOW(P) then
3:      F ← GETFLOWSTATE(P)
4:      if ISSHUTUP(P) then                          ▷ Got ShutUp/Throttle
5:          if SENTDATASINCERCVDSHUTUP(F) then
6:              SENDCHALLENGE(P)
7:          else
8:              drop                                 ▷ Redundant ShutUp
9:      else if ISVALIDCHALLENGERESPONSE(P) then
10:         if ISFORTHROTTLE(P) then
11:             RATELIMITFLOW(F, P.RATE, P.TTL)
12:         else
13:             BLOCKFLOW(F, P.TTL)
14:             CHECKSCANNING(P.IP_src, P.APP)
15:     else
16:         SUBDSTPORTOFFSET(P, F)                   ▷ Unobscure 5-tuple
17:         deliver                                  ▷ To app
18: else                                             ▷ New flow
19:     if ISSHUTUP(P) then
20:         drop                                     ▷ No flow to ShutUp
21:     else
22:         F ← NEWINBOUNDFLOW(P)
23:         deliver
24: if not WASDROPPED(P) then
25:     UPDATERCVDTIMESTAMPS(F, P)
```

blocks (or rate limits) the unwanted traffic.

2.3 SM Operation

Algorithms 1 and 2 list in pseudo-code the SM operation when forwarding a packet from the application to the network and vice versa respectively. The SM: 1) prevents spoofed source address, 2) unilaterally rate limits flows after an initial grace period unless the NM explicitly allows a higher rate (effectively acting as an E2E capability), 3) enforces compliance with ShutUp requests, and 4) detects and slows scanning attacks. The state maintained at the SM is listed in Table 2.

2.3.1 Preventing Source Address Spoofing

A host that can spoof source addresses can launch an attack without allowing ShutUp requests to reach it. To prevent this, the SM must prevent source spoofing. The difficulty in doing this lies in determining what an acceptable source address is. A firewall can be configured with this information. An endhost SM must however determine whether the endhost is authorized to use the present address. It is hard for the SM to make this determination, especially if the endhost statically assigns the address. We however consider cryptographic solutions [3] unnecessarily complex in this context.

The SM rate limits sending "unvalidated" packets to each layer-two neighbor, where validation requires receiving a packet for the purported IP address from that L2 neighbor (lines 1.1 and 2.1). Since the SM does not know which addresses are spoofed and which not, rate limiting, rather than blocking, is necessary to give an unvalidated address a chance to be validated. If the address is in fact spoofed, the validation does not succeed as long as the spoofing host is not on the same subnet as the recipient, as responses are not routed back to the SM, and the rate limit is maintained indefinitely. Since virtually all applications send packets in both directions [26], validation is effectively piggybacked on application traffic. The approach does not require any infrastructure or changes to the protocol stacks.

We choose to validate the IP address and destination MAC pair rather than just the IP address. Doing so foils two colluding endhosts on the same network attempting to validate a spoofed address. For instance, a colluding L2 neighbor may trigger validation by sending a packet to the endhost's MAC with the destination address set to the spoofed address — the resulting validation whitelists the spoofed address, but for use with that neighbor only. Therefore for the security of the mechanism, the SM must prevent spoofing the source MAC address. But since the SM has exclusive access to the physical network hardware, this is not difficult. (Section 3 discusses the case where the colluding host does not have an SM.)

Per L2 Neighbor	
IPs	: Source IP addresses validated
Per Destination IP	
$acknowledged$	: Sent throttle before initial timeout
Per Active Flow	
id	: 5-tuple plus source port offset
$rate$	: Rate limit set for flow (possibly 0)
TTL	: Time rate is in effect
$4 \times timestamp$	: Time ShutUps and data for the flow were last sent and received
Per Application	
$\#tokens$	: Number of ShutUps allowed
$rate$	: Current replenishment rate
$shutups$	: Recipients that have sent ShutUps
$whitelist$	: Recipients that never sent ShutUps

Table 2: State maintained by SM

2.3.2 Delivering ShutUp Messages

ShutUp messages contain the 5-tuple of the data flow they refer to. The messages are sent along the datapath encoded as ICMP packets with the 5-tuple contained in the encapsulated payload headers. The reason for this is that, NATs and firewalls that translate and forward ordinary ICMP messages for a flow [44] will transparently do so for ShutUp messages as well. Another option would be to use a shim layer between IP and the transport, which we avoid since it modifies the protocol stack, thus breaking existing middleboxes and application firewalls.

2.3.3 Flow Initiation and ShutUps

A general problem with using multiway handshakes in DoS prevention mechanisms is that the validation message is likely to be dropped at the bottleneck link. If the attack is large enough, very few validation messages will get through, and it will take a long time to slow the attack. To deal with this, the SM enforces a rate limit on flows that must be explicitly lifted by the recipient NM. The rate limiting operates as follows.

The SM initially allows the application to send at an unlimited rate on new flows, but only for a short time interval. The interval is picked conservatively (10 seconds in our experiments) within which time the recipient must send a THROTTLE message indicating the allowed rate limit for the flow. If neither a THROTTLE nor a SHUTUP is received, the SM automatically rate limits the flow. When the throttle's TTL expires, the SM again rate-limits the flow (to 10kbps in our experiments). The NM must periodically send throttles to maintain the flow at high speed.

In addition to rate limiting flow packets, the SM limits both inbound and outbound ShutUp requests. For outbound ShutUp requests sent by the application, the SM ensures that the associated flow exists, and has received data since the last ShutUp was sent. This gates ShutUp messages with application traffic preventing endhosts from abusing ShutUp messages to launch certain attacks. Similarly, the SM ignores inbound ShutUp requests for flows that do not exist and flows on which no data has been sent since the last ShutUp was received. These safeguards require the SM to maintain per-flow state and timestamps.

Legacy recipients: The automatic rate limiting above does not affect legitimate flows to NM-enabled recipients, but creates tradeoffs for legacy recipients that lack a NM. On the one hand, the rate limit mitigates DoS attacks on legacy recipients incapable of sending ShutUps. While on the other hand, legitimate flows to legacy recipients that last more than a few seconds are unnecessarily slowed. To avoid this latter issue, the SM disables the rate limit if it receives *any* packet for the flow from the receiver. But to prevent an attacker from abusing this mechanism, the SM makes it hard for the attacker to spoof a flow packet, for instance by adding a random offset to the source port in the 5-tuple[1]; an attacker may attempt to brute force the 5-tuple, but the SM can detect such an attempt. The security implications of this mechanism are considered in Section 3.

2.3.4 Slowing Scanning Attacks

ShutUp usually slows the scanning application rather than cordoning off the entire infected endhost, in order to reduce collateral damage to other applications. Identifying the application from just the flow 5-tuple is notoriously hard. Firewalls often resort to complex deep-packet inspection to identify application flows to dynamically selected ports. ShutUp avoids this complexity by having the NM provide the application *name* in the ShutUp message. Doing so is not hard for the NM since it is deployed in the recipient application or endhost OS and can readily query the application. Since the NM is not trusted in any event, the name reported by the application is not verified.

At its simplest, the SM considers an anomalous rate (per unit time) of ShutUps an indication of a scanning attack, at which point new flow initiations for the application are rate limited (lines 2.14 and 1.22). In principle, a number of other options can be used for detecting scanning attacks, ranging from the fraction of flows that receive ShutUps [53, 40], to the more complex introspection of the user VM [14]. We base our choice on the simplicity of the approach and the low number of false positives we

[1]Like NAT except only the local port for locally initiated flows is modified. Since inbound packets can be unambiguously delivered, NAT traversal approaches are not needed.

encountered as we report in later sections.

The SM algorithm for slowing scanning attacks is as follows. The SM maintains a token-bucket for each application. A token is consumed by each ShutUp from a distinct recipient. Flow initiations to recipients are blocked if no tokens are present, however, flows to a dynamic list of previously successfully contacted recipients are not affected. Over time, tokens are replenished at a configured rate up to a maximum value, but persistent scanning decreases this rate at which tokens are replenished. The reasoning behind these design decisions are as follows.

Distinct recipients: Consuming one token for multiple ShutUps from the same recipient limits the potential damage caused by a (malicious) recipient. For this purpose, the SM maintains per-application state consisting of recipients that have previously sent ShutUps.

Blocking by application: While the application name is piggybacked on ShutUp requests as mentioned above, at flow initiation time the packet 5-tuple does not, in general, identify the application. Since the first flow packet can itself exploit a vulnerability [31], this creates a chicken-and-egg scenario where the first flow packet must be blocked if it is for the offending application, but to determine the application, the flow must be allowed and a ShutUp sought. ShutUp breaks this dependency with the Query primitive (Table 1), which is a simple request-response exchange whereby the SM queries the NM for the name of the application the flow would reach if it were to be allowed. The query is not needed for well-known ports and is only invoked when the host is believed to be participating in an attack. Legacy recipients without an NM cannot respond to these queries.

New Recipients: To mitigate the impact of false positives, the SM maintains a list of recipients contacted previously that have never sent a ShutUp for the application. While flow initiations to new recipients are rate limited during a suspected scanning attack, flows to recipients on this list are whitelisted.

Rate limiting flow initiation: A common concern with thresholds is how the threshold is picked. A low static threshold creates false positives, which we determined in our dataset to be bursty, whereas a threshold high enough to accommodate bursts allows sub-threshold scans that does not adequately slow the attacker. For this purpose, ShutUp uses a dynamic threshold: the rate at which new flow initiations are allowed is picked randomly (our implementation uses an exponential distribution around a configured mean), and this rate is halved every time the token-bucket underflows. In effect, the rate limit allows large but short bursts of ShutUps for legitimate applications, while forcing persistent scans to a much lower rate.

2.4 NM Operation

A single recipient NM cannot detect the wide range of unwanted traffic we are interested in (address scans, DoS, intrusions etc.). The design therefore accommodates separate collaborating NMs operating at various layers; each NM tags ShutUp messages with its ID and responds to challenges intended for it. However, ShutUp does not focus on *how* these attacks are detected — we rely on existing or future methods of detecting unwanted traffic at the application, endhost OS, and recipient firewall including IDS (e.g. [36]), CAPTCHAs [50], and exploit detectors (e.g. [9]) as applicable.

Out of the three, the design of the endhost OS NM is non-trivial. If the recipient application is not running, the endhost OS must determine whether or not to send a ShutUp. The choice is not clearcut because sending a ShutUp prevents the source from reattempting the flow, at least until the ShutUp expires. Such ShutUps hinder interactive applications such as web browsing if the web server is momentarily unavailable. Further, if enough endhosts generate false positives, it unduly triggers the scanning defense at the initiator. This affects P2P applications, for example, where stale membership information results in peers attempting to contact a recently running (or crashed) application for some time. To avoid these false positives, applications in ShutUp register an application-specific "linger" time when binding to a port (few minutes for P2P applications, infinity for server applications). The endhost OS does not send ShutUps for an unbound port until the linger period expires. In addition, the OS NM may have a small default linger time to accommodate legacy apps that do not set a linger time.

3 Attacking ShutUp

In this section we consider attacks on ShutUp components and mechanisms through which an attacker may attempt to disrupt flow establishment.

Attacker Model: We assume that the attacker has complete physical control over a small number of hosts. In particular, the attacker can disable the SM on these hosts. In addition, the attacker has software control over a much larger number of compromised hosts.

We initially assume the attacker is not on the datapath between the initiator and recipient. An on-path attacker, for instance on a compromised router, can disrupt communication by dropping or modifying packets enroute even in the absence of ShutUp. We later relax this assumption to consider eavesdroppers — on-path attackers that can observe but not modify in-flight packets.

Software Compromise: Vulnerabilities in the SM implementation or virtualization layer could allow an attacker to gain control of the SM through software meth-

ods. In case of compromise, the VMM can restore a pristine SM from readonly media, and rely on automatic software update mechanisms to apply all relevant patches. Physical access is required only if the VMM itself is compromised or for updates to the readonly media.

Compromising the NM is less severe. A compromised NM may fail to ShutUp unwanted flows, or worse, ShutUp flows that are not unwanted. In the first case, NMs deployed at other layers can still ShutUp unwanted flows. In the second case, the danger is not so much to the recipient, since such an attacker can block flows passing through the NM even without cooperation from the SM, but rather to the initiator for being falsely implicated of unwanted traffic. However, the attacker must compromise multiple NMs at recipients contacted by the initiator to successfully trigger the scanning defense, mitigating the severity of the attack.

Spoofed Packets: In order to spoof packets at an unlimited rate without disabling the SM, the attacker must be able to complete the address validation process by spoofing the MAC address of the firsthop router, which requires physical access to a second endhost on the same network. With software-only access to a given network, an attacker can at best hijack an address in that network, but since the endhost would receive all packets for the hijacked address, in particular ShutUp messages as well, there is little impact on ShutUp for doing so. Consequently, the number of networks an attacker can spoof packets from is limited by the number of hosts the attacker has physical access to — significantly better than today where up to 25% of edge ASs allow spoofing [6]. If an attacker compromises a legacy (no SM) host on the subnet, it can fake the MAC address of the router and then create arbitrary addresses on compromised SM-enabled hosts, however, if the attacker already has such a foothold in a legacy host, it can in any event spoof arbitrary addresses.

In order to successfully fake ShutUps, an attacker must be able to eavesdrop packets. This is because, as previously mentioned, all ShutUp messages must be validated before any action is taken by the SM, and the attacker must be able to guess the nonces involved to fake the validation. But since the nonces are not encrypted, an eavesdropper can win a validation race. However, since the eavesdropper cannot stop the in-flight challenge, the real recipient will generate a conflicting response. The SM can therefore at least detect the presence of an eavesdropper, although not which of the responses is legit. In any event, an eavesdropper can disrupt flows even without ShutUp, so ShutUp does not introduce a qualitatively new attack.

Abusing ShutUp Messages: ShutUp messages are gated by application traffic at the SM. The mechanism prevents an attacker from flooding a victim with ShutUp messages absent an application flow between the two. Even when a flow exists, the number of ShutUp messages that can be sent is upper bound by the number of flow packets. The same mechanism gates reflection attacks consisting of ShutUp challenges if an attacker spoofs a stream of ShutUp messages; the amplification factor for such attacks is at most one.

Avoiding Initial Rate-Limit: An attacker may attempt to disable the initial rate limit for unacknowledged flows by abusing the mechanism intended for legacy recipients. Doing so requires the attacker to guess the random initiator port, which requires on average 2^{15} guesses. After the first few guesses, the SM can detect the attack and lock in the rate limit such that only a validated Throttle may lift it. Thus an attacker cannot disable the initial rate limit, nor affect flows to NM-equipped recipients, and can at best target flows to legacy recipients to be rate limited; the recipient can counter by deploying an NM.

Triggering Scanning Defense: An attacker can trigger the scanning defense at an endhost by convincing the initiator to contact recipients not expecting the flows. For instance, a malicious website may return a webpage with inline image URLs pointing to recipients not running a webserver. Alternatively, a compromised router enroute to multiple destinations may fake ShutUps. While Shutup does not defend against such attacks, ShutUp limits the impact. First, in case of a duped application the scanning defense only applies to the one application and not to other applications running on the endhost. Second, the defense does not affect communication to recipients that have never previously ShutUp the endhost, allowing the application to operate with diminished reachability. Finally, as the defense is lifted unless the application is persistently scanning, a legitimate application can resume normal operations after cautioning the user against reattempting flows to the problematic destinations.

4 Stopping DoS with ShutUp

The ShutUp service can be used to mitigate both application level as well as network level DoS attacks. At the application level, the efficacy depends on how quickly and accurately the attack is detected, which is contingent on detection mechanisms external to ShutUp; once detected, the application NM sends ShutUps to the attackers. Defending against network level DoS, where the attackers saturate a bottleneck link, is more involved, because the ShutUp challenge-response mechanism must operate through the same bottleneck.

If the bandwidth of challenges incident at the bottleneck link is B, and aggregate attacker bandwidth is X times the bandwidth of the bottleneck link, then only B/X challenges will cross the bottleneck. To put in num-

bers, if a small number of attackers, say 5000, command a bandwidth 10 times that of a 100Mbps bottleneck, an incident challenge bandwidth of 10Mbps would ShutUp around 1000 attackers per second assuming 128 byte challenge packets, stopping the attack in 5 seconds.

Relying solely on the challenges getting through the bottleneck suffices for small attacks but not large attacks. This is because the incident challenge bandwidth B required to maintain the same rate of validated ShutUps increases linearly with X. However, since challenges are driven directly by ShutUp requests, B is upper bound by the victim's upload bandwidth, which remains fixed as X increases. Moreover, B is more realistically a fraction, typically one-tenth, of the bottleneck bandwidth assuming $\sim$1kB attack packets. This is because the victim can, at best, generate one ShutUp per attack packet crossing the bottleneck, each of which results in a challenge. Generating more ShutUps or challenges creates the possibility of amplification attacks. Since B cannot be increased indefinitely, ShutUp decreases X for large DoS attacks.

In large DoS attacks (e.g. $X > 20$), the SM enforced automatic rate limit after the first few seconds cuts X by several orders of magnitude. The rate limit alone does not address congestion at the bottleneck, as even with the diminished bandwidth, the attackers can be numerous enough to saturate the bottleneck. The purpose of the rate limit is to increase the fraction of challenges in the attack traffic, which is possible because B is independent of X as long as the bottleneck is saturated. This allows the challenge-response mechanism to kick in more effectively and stop large DoS attacks.

The above discussion assumes that only the NM at the victim is responding to the attack, however, other NMs may additionally be involved. For instance, ShutUp challenges to an ISP NM upstream of the bottleneck link would succeed. Or in case of an endhost behind the bottleneck colluding with attackers by not sending ShutUps for traffic crossing the bottleneck, a firewall NM between the bottleneck and the endhost may instead send ShutUps to protect other endhosts behind the bottleneck. Overall ShutUp requires an NM to be present somewhere along the attack path to stop a DoS.

Simulation Results: We used *ns-2* to simulate how well ShutUp mitigates DoS attacks. We use a fan-in topology where a varying number of attackers (from 10 to 9600) send traffic to a bottleneck link; the recipient is on the other side of the bottleneck. The topology models a DoS where attack traffic does not self-interfere except at the bottleneck. In the Internet, attackers sharing a common link other than the bottleneck may cause additional packet loss. Nevertheless, since ShutUp is entirely end-to-end (or edge-to-edge), we expect such AS- and router-level topology to have little impact. To convince ourselves of this, we simulated ShutUp on the router-level topology collected by the Rocketfuel project [43] and a sampling of the AS topology collected by the RouteViews [49] project and found the results to be qualitatively similar; however, these simulations were restricted to small attacks (upto $X = 100$) due to simulator limitations.

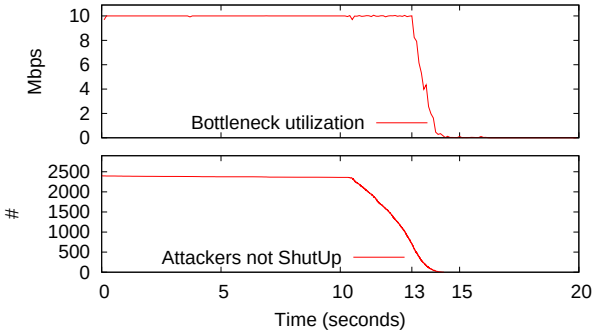

Figure 3: The effect of a DoS attack under ShutUp. Aggregate attack traffic is 240 times the bottleneck bandwidth. The SM slows down attackers at 10s, increasing the rate of challenges through the bottleneck.

The latency between the attacker and the bottleneck is parameterized by end-to-end RTT data collected by the Meridian project [55]. Each attacker sends 1Mbps of attack traffic. The bottleneck link is set to 10Mbps with droptail queuing. All other links are set to 10Gbps. We simulate attacks from 10Mbps to 9.6Gbps of attack traffic, that is between 1 and 960 times the bottleneck bandwidth.

To determine the contribution of the ShutUp mechanism and automatic rate limit, we compare the effectiveness with and without the rate limit. When enabled, the SM automatically rate limits traffic to 10Kbps after 10 seconds. The choice of these parameters is primarily to explore the scaling properties of the DoS defense mechanisms in ShutUp rather to than guide deployment.

Figure 3 plots a representative attack with aggregate bandwidth 240 times the bottleneck. The figure is split in two parts: the top plots the bottleneck link utilization over time, while the bottom plots the number of attackers that have not received a validated ShutUp. We observe three distinct phases during the attack. Phase 1 lasts for 10 seconds where the bottleneck is saturated and the number of active attackers decreases marginally by 1.6% because few challenges get through. Phase 2 begins at 10 seconds when the automatic rate limiting kicks in; the active attackers drop rapidly as more challenges succeed, but the aggregate attack traffic, while decreasing, still exceeds the bottleneck. Phase 3 begins at 13 seconds when the aggregate attack traffic equals the bottleneck capacity, at which

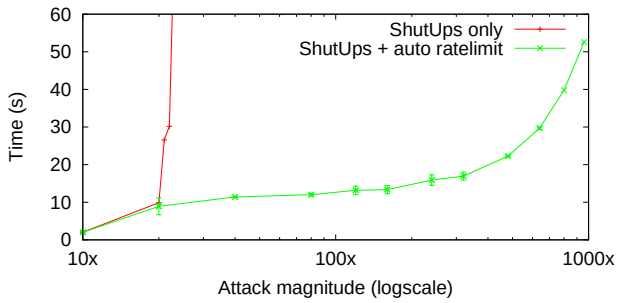

Figure 4: Time taken to stop a DoS attack as attack magnitude increases

point all challenges are delivered and the remaining attackers blocked within 1.3 seconds. The figure illustrates that both the ShutUp mechanism and the automatic rate limit mechanism are necessary to contain large DoS attacks, and together, sufficient to block the attack in 15 seconds in the above example.

Figure 4 plots the time taken to stop a DoS as a function of the attack magnitude. For small attacks, enough challenges get through that the ShutUp mechanism alone can completely stop the attack, but there is a threshold ($X = 20$) beyond which the automatic rate limiting is needed to keep pace with the attack magnitude. Only when the attack magnitude increases by two orders of magnitude — the same factor as our chosen automatic rate limit parameter — does the combined approach experience a disproportionate increase. Overall, ShutUp is able to stop attacks ranging widely in their magnitude up to nearly 1000 times the bottleneck bandwidth.

To determine the effectiveness of a partial deployment, we simulated attacks where we varied the fraction of legacy attackers that do not have an SM, and are therefore immune to ShutUps. We simulated ShutUp deployments from 90% to 10% holding the aggregate bandwidth of legacy attackers constant at 90% of the bottleneck link. The results were not surprising: in all cases, ShutUp is able to desaturate the bottleneck in under two seconds by stopping all SM-enabled attackers. ShutUp reduces attack bandwidth by the same fraction as the degree of deployment (i.e. 90% reduction for 90% deployment) — what we might call "incremental improvability", where the more endhosts that have it, the better the results.

5 Slowing Scanning Worms

The scanning defense in ShutUp allows worm outbreaks to be contained (or slowed) using the service. A scanning worm, by its nature, propagates by discovering and infecting vulnerable endhosts. There are several ways how such worms may receive ShutUps. Probes to endhosts not running the vulnerable application, probes to honeynets [42] or network telescopes [30], attempting to infect a vulnerable endhost protected by application level defenses [9], traffic monitored by in-network intrusion detection systems [36] and firewalls may all result in ShutUps.

As mentioned, the SM slows down new flow initiations when a threshold rate of ShutUps is crossed. In contrast, threshold random walk (TRW) based detectors [53, 40] place bounds on the fraction of bad flows to good flows. Both approaches have their pros and cons. TRW can detect an extremely low rate of scanning, but is susceptible to collusion where worm instances can maintain a fixed fraction by establishing "good" flows amongst themselves or with botnet members to absolve an equal number of probes. ShutUp is not affected by such collusion, but does not target sub-threshold scanning; to the extent the randomness and exponential rate limit in ShutUp compel the worm to scan slowly, ShutUp buys time for other approaches to detect and disinfect compromised hosts. Consequently, we focus on fast scanning worms.

ShutUp ignores worm probes that do not generate a response (not even a ShutUp), for instance probes to nonexistent addresses or to hosts behind NATs; interpreting such silent probes as implicit ShutUps would increase the ShutUp frequency and thus reduce the reaction time, but it would also increase false positives (Section 6.2). Inside an enterprise, last-hop routers could certainly be configured to send ShutUps for such probes. On the Internet, however, we expect policy will prevent stealthy NAT/firewalls from sending ShutUps for such probes [16]. Fortunately, as we report below, the impact of ignoring these silent probes is minimal for a typical worm attack. We hope to revisit this design decision in the future if the behavior of legitimate applications improves.

Simulation Results: We simulated worm outbreaks in a custom simulator to determine how well ShutUp performs as compared to the TRW-based approach proposed in [53]. We parameterized our simulator with Internet census data collected by the ANT project [17]. As per the data, out of the 2.8 billion allocated unicast public IPv4 addresses, 187 million (6.7%) respond to probes. The remaining addresses are either unused, or used by stealthy hosts. We simulate a Code Red-like worm [32] that scans at $\sim$11 addresses per second, and has a vulnerable population of 359K endhosts. The vulnerable endhosts are distributed uniformly at random among the 2.8B allocated addresses. If the worm probes a vulnerable address, the destination is instantly infected (unless already infected) and begins scanning. Otherwise, if the probe is to the 6.7% addresses that send a response, a ShutUp is simulated, else the probe times out (remaining 93% of the time). At $t = 0$, the attacker infects 1000 vulnerable hosts

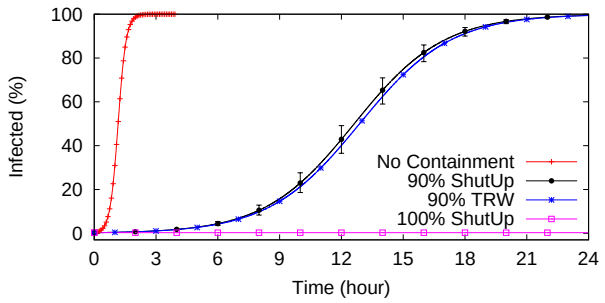

Figure 5: Worm outbreak under ShutUp. While a full ShutUp deployment can completely contain scanning worms, a partial deployment slows a worm significantly

to initiate the outbreak.

Figure 5 plots the number of infected hosts as a function of time. Without containment, the worm infects 95% of the vulnerable population in 1hr 43m. With full ShutUp deployment, the worm is completely contained at 0.3% infections; a full TRW deployment performs identically (not shown). If ShutUp deployment is only 90%, worm propagation is slowed down by an order of magnitude. In 1hr 43m, the worm is able to infect 0.63% versus the 95% without containment, and time taken to infect 20% vulnerable hosts is increased by 8hr 44m. Surprisingly, the difference between Shutup and TRW, while noticeable, is small even though ShutUp ignores probes to the 93% stealthy/unused addresses while TRW does not. This is because the fraction of responding hosts that are vulnerable is small enough that a sufficient rate of ShutUps is generated to contain the infected host before it discovers a vulnerable host. Consequently, trading off sensitivity to silent probes for fewer false positives is well justified which, as we report later, significantly reduces false positives for ShutUp in comparison to TRW. Overall, a full ShutUp deployment can completely stop a worm, while a partial deployment can slow it down significantly.

6 Evaluation

To evaluate the impact on real endhosts, we evaluated ShutUp using datasets from an enterprise environment, an academic environment, and a home environment.

Implementation: We implemented a proof-of-concept SM in Python. The SM processes connection events generated by Bro [36], infers ShutUps from TCP errors, and for each new outbound flow, passes a verdict whether the flow would have been allowed, rate limited, or blocked had the SM been running on the endhost during data collection. The implementation is 153 lines long demonstrating the simplicity of the SM. While our implementation allowed us to rapidly prototype (and refine) the ShutUp design, and provided insights into the impact on legitimate flows, performance and auditability of the SM software stack are important deployment concerns that we believe are best addressed using a non-interpreted language.

Trace Data: Our enterprise dataset consists of a month-long trace of packet headers for all traffic to and from endhosts, primarily employee laptops, for a major corporation. The data was collected *at the endhost*, and thus includes traffic even when the endhost was outside the enterprise. This is the ideal dataset for evaluating an endhost SM. The trace contains little peer-to-peer filesharing traffic as enterprise policy forbids the use of P2P applications. The trace covers 357 users during the first quarter of 2007 with the median trace lasting 26 days. Processed through Bro, the trace contains about 24.5M TCP flows initiated by the endhosts to 111K addresses.

Our university dataset consists of a 6-day trace of packet headers from endhosts in the computer science department of a major university collected at the border router. The dataset is ideal for evaluating a firewall SM. As with the enterprise, policy discourages the use of P2P filesharing applications. The data contains 5M flows from 1680 IP addresses during a week when the university was in session.

Our home-user dataset consists of a 8-day trace of primarily BitTorrent traffic collected in a home network. It contains 37K flows over which around 5Gb of data was transferred in both directions, which we use to counter the lack of P2P filesharing traffic in the other two datasets.

Methodology: We treat TCP RST packets in response to the initial SYN packet as ShutUps. Flows over which at least one byte of application data is exchanged do not generate ShutUps regardless of how the flow is terminated (e.g. FIN exchange, RST packets, or TCP timeout). Flows where the initial SYN packet does not elicit any response within 3 seconds are treated as unacknowledged flows. We discard other TCP flows (e.g. when a SYN ACK is seen but no data is transferred); such flows comprise less than 0.1% of our data. We do not analyze UDP traffic as we cannot, in general, determine if a flow would have generated a ShutUp. Furthermore, because we do not have packet payloads, our evaluation is limited to ShutUps that an endhost or firewall NM may generate; consequently, application-level errors over a successful TCP flow (e.g. SMTP errors) count as successful flows.

6.1 Tuning Parameters

Our choice of system parameters is driven by our data.

IP Spoofing Limit: Users in our enterprise trace use, in the median case, 24 unique source IP addresses over the duration of the trace, with 10% of users using more than 70. The number is significantly higher than that mea-

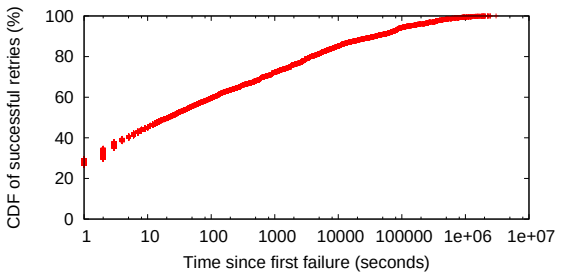

Figure 6: Determining the impact of endhost OS ShutUp TTL and linger time on legitimate retries

sured for the average Internet user (1 IP address over 2 weeks) [7], which can be explained by the difference in vantage points as [7] measures from the perspective of a content provider. On average, the SM must validate a source IP address and MAC pair every 7 hours.

87% of the time the address is validated within one second, typically by the first application flow. In the worst case, an endhost attempted 1 flow per second for 18 minutes before eliciting any response; in such cases, the endhost OS may be configured to ping the first-hop router shortly after assigning a new IP address to more quickly validate the address. Our implementation rate limits packets with unvalidated addresses MAC pairs to a burst of at most 60 packets in the first ten minutes, and one packet per minute after that. The rate limit does not impact 99% of address changes.

Linger period and ShutUp TTL: In order to determine the minimum linger period to be observed by the endhost NM, and the TTL for ShutUp requests, we plot the CDF of the time elapsed until a flow to a destination succeeds after the first encountered failure in Figure 6. The y-value represents the fraction of retries that would have succeeded had a failed flow not been incorrectly ShutUp for a given TTL value (x-axis). Curiously, the plot for our enterprise trace is linear on a semi-log scale across six orders of magnitude — a phenomenon well worth investigating in its own right. In essence, the longer an application has been failing, the longer it is expected to keep failing, whereas the more recent the failure, the more likely it is that a quick retry will succeed, something which we term "failure momentum". A minimum linger time of 2 seconds prevents 35% of retries from being incorrectly blocked. Beyond this, the NM may dynamically pick the TTL in proportion to the length of the outage, however, our implementation uses a static value of 60 seconds.

Scanning threshold: In order to determine a threshold that does not impede legitimate flows while placing a bound on scanning activity, we require an oracle to separate benign traffic from suspicious traffic in our dataset. For this purpose we use the TRW-based scan detection algorithm proposed in [53] to flag potential scanners in our enterprise dataset. We assume the 182 users *not* flagged are least likely to be scanners. For these users, a minimum threshold of 9 ShutUps per minute is necessary to not trigger the scanning defense. Adding a margin for error, we use a threshold of 15 ShutUps per minute before triggering the scanning defense.

Application	*Port(s)*	*Dataset*	*# Hosts*	*# Flows*
ShutUp				
Patch Dist.	63422	Enterprise	9	60
SMTP	25	University	4	171
-	-	Home User	-	-
	Total (all applications):		*7.3%*	*0.02%*
ShutUp (w/ timeout as ShutUp)				
Patch Disc.	63422	Enterprise	119	959
Unknown	3274	University	4	4811
BitTorrent	*	Home User	1	216
	Total (all applications):		*17.6%*	*0.14%*
TRW-based				
Web	80,443	Enterprise	97	15115
SMTP	25	University	3	459750
BitTorrent	*	Home User	1	1078
	Total (all applications):		*8.9%*	*1.7%*

Table 3: Applications generating the most potential false positives for each dataset

State: The state maintained by the SM is dominated by the per application whitelist that is built dynamically. The list can potentially grow unbounded. The 99 percentile size was 2300 entries for a node (8kB of IPv4 addresses); the maximum was 32K (128kB). At most 437 ShutUps were received in all by a host, of which fewer than 15 were concurrently active in the 95^{th} percentile case. In all, 256kB of state per endhost is sufficient for an endhost SM for our dataset. A firewall SM with limited memory may, however, use a fixed-sized cache for the whitelist, taking care to protect against attackers abusing the cache-replacement strategy.

6.2 False Positives

We evaluated ShutUp on all three datasets. For comparison, we also evaluated the TRW-based approach proposed in [53], and a modified version of ShutUp that treats unacknowledged flows as implicit ShutUps. We use well-known port numbers to determine the application name in the enterprise and university trace, while for the home user trace, we tag all flows generated by BitTorrent as such regardless of the port number. For ShutUp, we use the parameters above. For TRW, we primarily use the parameters in [53], which the authors tune using multiple datasets

and conclude are applicable in general. We also tested TRW with higher thresholds, and while this reduced the false positives quantitatively, it did not do so qualitatively: the same applications generate the most false positives.

For each approach, we consider flows that the approach decided to block, but which did in fact exchange data during the trace. We manually examined the most egregious endhosts and eliminated blatant scanners based on sequential address or port scans. However, since we lack payload data to automatically disambiguate certain cases, some successful flows may in fact be successful scan attempts; as such, our results identify legitimate applications potentially affected by ShutUp. Table 3 plots the top applications generating potential false positives for each dataset. Overall, ShutUp results in fewer false positives (0.02% versus 1.7%) for the following reasons:

Enterprise: In the enterprise case, as mentioned previously, the linger mechanism reduces the number of ShutUps generated due to momentary outages. False positives in ShutUp stem from the automated patching application which performs distributed scanning to quickly discover unpatched endhosts; we expect IT to configure enterprise endhost NMs to not send ShutUps for benign scanning authorized by the enterprise, or filter ShutUps for such applications in the network.

University: The outbound mailserver has a ratio of 70% unsuccessful flows, largely due to undeliverable bounces to spam messages, triggering the scanning defense. The whitelist in ShutUp allows flows to legitimate mailservers that have never sent a ShutUp to continue unimpeded.

Home User: BitTorrent generates a large number of unacknowledged flows, presumably to other clients behind NATs, but potentially also to clients that have recently left the cloud. Since ShutUp ignores such flows, the scanning defense is not triggered.

7 Extensions to ShutUp

In this section we present some extensions to ShutUp that enhance the service provided by the SM. In general, these extensions present new potential vulnerabilities that need to be considered.

Weighted ShutUps: The SM can give more importance to ShutUps for more severe violations or from authoritative sources. For instance, a ShutUp from an application protected by Vigilante [9], could instantly trigger the scanning defense. Such ShutUps would need to carry proof of their authenticity, such as Vigilante's self-certifying alert that the SM can verify in the ShutUp VM.

Initiator behavior: The SM can provide information about the initiator's past behavior, such as the diversity of recipients contacted, to new recipients. The NM can compare this to the expected diversity for the application [22] to help determine the legitimacy of the flow.

Collaborating SMs: The SM can collaborate with nearby SMs to determine the uncleanliness [8] of the local network. The metric may be used to adjust the sensitivity of the SM to better contain worms.

Wireless Contention: Wireless nodes sharing the broadcast medium with an overly-chatty node could request it to ShutUp. Once the SM verifies that a majority of wireless neighbors concur, it can throttle the application reducing contention.

8 Related Work

Receiver oriented communication models are not new. IP multicast [11], i3 [45], and off-by-default [5] all allow the recipient to choose what data it receives. Multicast is not intended for one-to-one communication, i3 requires infrastructure, and off-by-default requires modification to routers and Internet routing. In contrast, ShutUp is end-to-end (or edge-to-edge) and requires no infrastructure.

The work closest to the ShutUp service, either because they use a challenge-response approach or because they propose tamper-proof mechanisms in the attacking endhost [41, 3, 4], are discussed in Section 1.

A large class of DoS mitigation approaches operate by installing filters along the attack path. In the simplest example, [38] thins attack traffic using passive and active filters at the upsteam ISP. Taking this idea further, endhosts in [4, 29] install filters in gateway routers, or routers progressively closer to the source. Combining filtering with explicit authorization, [56, 57] configures routers upsteam of the bottleneck to, by default, impede packets not explicitly authorized by the endhost through capability nonces. A second class of approaches [1, 12, 28, 2, 23] dissipate DoS attacks before absorbing the unwanted traffic. [51] uses a currency, such as bandwidth, to grant access through filters at network choke points. In contrast, ShutUp filters DoS traffic at the source (or its firewall) preventing attack traffic from consuming any network resources. Furthermore, ShutUp, more generally, handles scanning attacks within the same architecture.

Worm containment has been subject of much recent work. [33] provides guidelines applicable to any worm containment approach. [54] rate limits all flow initiations, and [15] bounds the diversity of recipients. Targeted more directly at worms, TRW [20] detects scanning worms from the recipient firewall's perspective. Turning the TRW approach inside-out and operating at a router upstream of the infected endhost, [53] and [40] detect and contain worm outbreaks based on an anomalous fraction of failed flows for a particular port number. While the SM can use any combination of these mechanisms, ShutUp extends to applications that use dynamic ports.

Much work has been done in detecting unwanted traffic both in the network and at the recipient endhost that ShutUps complements. In the network, traffic can be classified as portscans [20], worms [24, 39], DoS attacks [18], flash crowds [19], and of dubious origin [10]. Classification can be performed through correlation [47], rule-based matching [25], or entropy in feature sets [27] in, potentially aggregated, flow parameters. At the endhost, application-agnostic exploit detection [9, 42], as well as application-specific detection based on common usage profiles, and deviations therefrom [37, 46] have been proposed. More active methods of detection includes CAPTCHAs [34, 21] or tracking user-activity [35] to verify the presence of a human, and puzzle-auctions [52] to rate limit attacks. The NM can directly use these and future approaches to detect unwanted traffic to ShutUp.

9 Summary and Future Work

In this paper, we propose ShutUp, an end-to-end (or edge-to-edge) service to reduce unwanted traffic in the Internet. We throughly explore the design space and determine the small set of mechanisms required to mitigate, and in many cases completely stop, a wide range of unwanted traffic ranging from DoS attacks to scanning worms. ShutUp does not require any additional infrastructure or changes to the protocol stack, is secure against a large class of attackers, and is easily deployable in enterprises, and globally deployable with buy-in from only a few vendors, providing incremental improvement as deployment proceeds. Through simulations we find that ShutUp scales well to defend against large scale DoS attacks up to three orders of magnitude in excess of bottleneck links, and can slow scanning worms by an order of magnitude. Using extensive trace data from three environments we establish that ShutUp's impact on legitimate traffic is minimal. Overall, we believe there is a compelling case to be made for containing unwanted traffic at the ends.

Beyond this, there are a number of interesting research directions that we hope to explore. Due to lack of space, we only briefly list them here. Foremost among these is to explore the use of self-certifying identifiers with ShutUp. Another is extending ShutUp to curtail unwanted traffic at higher layers such as spam and phishing. Finally, ShutUp appears to be a promising basis for other functions within the endhost SM that are controlled by entities in the network. For instance, a router could ask a host to format packets in a more efficient way (i.e. a stack of tags), to label packets with certain QoS tags, or to shape or rate limit traffic in a way that reduces load on the network. We call this model the "network embassy", because like one country's embassy situated in another country but not controlled by the other country, it represents an autonomous function within the endhost that is controlled not by the endhost but by the network.

References

[1] AKAMAI TECHNOLOGIES, INC. Akamai: How it works.

[2] ANDERSEN, D. Mayday: Distributed filtering for internet services. In *Proceedings of the USITS '03* (Seattle, WA, Mar. 2003).

[3] ANDERSEN, D., BALAKRISHNAN, H., FEAMSTER, N., KOPONEN, T., MOON, D., AND SHENKER, S. Holding the Internet accountable. In *Proceedings of HotNets '07* (Atlanta, GA, Nov. 2007).

[4] ARGYRAKI, K., AND CHERITON, D. R. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Proceedings of the 2005 USENIX Annual Technical Conference* (Anaheim, CA, Apr. 2005).

[5] BALLANI, H., CHAWATHE, Y., RATNASAMY, S., ROSCOE, T., AND SHENKER, S. Off by Default! In *Proceedings of the HotNets'05* (College Park, MD, Nov. 2005).

[6] BEVERLY, R., AND BAUER, S. The Spoofer Project: Inferring the extent of source address filtering on the internet . In *In proceedings of the 1st Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)* (Cambridge, MA, July 2005).

[7] CASADO, M., AND FREEDMAN, M. J. Peering through the Shroud: The Effect of Edge Opacity on IP-based Client Identification . In *Proceedings of the NSDI '07* (Cambridge, MA, Apr. 2007).

[8] COLLINS, M. P., SHIMEALL, T. J., FABER, S., JANIES, J., WEAVER, R., AND SHON, M. D. Using Uncleanliness to Predict Future Botnet Addresses. In *Proceedings of the 2007 Internet Measurement Conference* (San Diego, CA, Oct. 2007).

[9] COSTA, M., CROWCROFT, J., CASTRO, M., ROWSTRON, A., ZHOU, L., ZHANG, L., AND BARHAM, P. Vigilante: End-to-end containment of internet worms. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles* (Oct. 2005).

[10] DEAN, D., FRANKLIN, M., AND TUBBLEFIELD, A. S. An algebraic approach to ip traceback. *Information and System Security 5*, 2 (2002).

[11] DEERING, S. RFC 1112: Host Extensions for IP Multicasting, Aug. 1989.

[12] FREEDMAN, M. J., LAKSHMINARAYANAN, K., AND MAZIÈRES, D. OASIS: Anycast for Any Service. In *Proceedings of NSDI'06* (San Jose, CA, May 2006).

[13] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, October 2003).

[14] GARFINKEL, T., AND ROSENBLUM, M. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of ISOC Network and Distributed System Security* (San Diego, CA, February 2003).

[15] GOPALAN, P., JAMIESON, K., MAVROMMATIS, P., AND POLETTO, M. Signature metrics for accurate and automated worm detection. In *Proceedings of the he 4th Workshop on Recurring Malcode (WORM)* (nov 2006).

[16] GUHA, S., AND FRANCIS, P. Characterization and Measurement of TCP Traversal through NATs and Firewalls. In *Proceedings of the 2005 Internet Measurement Conference* (New Orleans, LA, Oct. 2005).

[17] HEIDEMANN, J., PRADKIN, Y., GOVINDAN, R., PAPADOPOULOS, C., AND BANNISTER, J. Exploring Visible Internet Hosts through Census and Survey. Tech. Rep. ISI-TR-2007-640, USC/Information Sciences Institute, Marina del Rey, CA, 2007.

[18] HUSSAIN, A., HEIDEMANN, J., AND PAPADOPOULOS, C. A framework for classifying denial of service attacks. In *Proceed-*

ings of ACM SIGCOMM (Aug 2003).

[19] JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proceedings of WWW* (May 2002).

[20] JUNG, J., PAXSON, V., BERGER, A. W., AND BALAKRISHNAN, H. Fast portscan detection using sequential hypothesis testing. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, CA, May 2004).

[21] KANDULA, S., KATABI, D., JACOB, M., AND BERGER, A. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of NSDI 2005* (May 2005).

[22] KARAGIANNIS, T., PAPAGIANNAKI, D., AND FALOUTSOS, M. Blinc: Multilevel traffic classification in the dark. In *Proceedings of ACM SIGCOMM 2005* (Aug. 2005), pp. 217–228.

[23] KEROMYTIS, A. D., MISRA, V., AND RUBENSTEIN, D. SOS: secure overlay services. *SIGCOMM Comput. Commun. Rev. 32*, 4 (2002), 61–72.

[24] KIM, H.-A., AND KARP, B. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the Usenix Security Symposium* (Aug 2004).

[25] KIM, M.-S., KANG, H.-J., HUNG, S.-C., CHUNG, S.-H., , AND HONG, J. W. A flow-based method for abnormal network traffic detection. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium* (April 2004).

[26] KREIBICH, C., WARFIELD, A., CROWCROFT, J., HAND, S., AND PRATT, I. Using Packet Symmetry to Curtail Malicious Traffic. In *Proceedings of the HotNets'05* (College Park, MD, Nov. 2005).

[27] LAKHINA, A., CROVELLA, M., AND DIOT, C. Mining anomalies using traffic feature distributions. In *Proceedings of ACM SIGCOMM 2005* (Aug. 2005), pp. 217–228.

[28] LAKSHMINARAYANAN, K., ADKINS, D., PERRIG, A., AND STOICA, I. Taming IP packet flooding attacks. *ACM Computer Communications Review 34*, 1 (Jan. 2004), 45–50.

[29] MAHAJAN, R., BELLOVIN, S. M., FLOYD, S., IOANNIDIS, J., PAXSON, V., AND SHENKER, S. Controlling High Bandwidth Aggregates in the Network. *ACM Computer Communications Review 32*, 3 (July 2002), 62–73.

[30] MOORE, D. Network Telescopes: Observing Small or Distant Security Events . In *Proceedings of the the 11th USENIX Security Symposium (Security '02)* (San Francisco, CA, Aug. 2002).

[31] MOORE, D., PAXSON, V., SAVAGE, S., SHANNON, C., STANIFORD, S., AND WEAVER, N. Inside the Slammer Worm. *IEEE Security and Privacy 1*, 4 (2003), 33–39.

[32] MOORE, D., SHANNON, C., AND KC CLAFFY. Code-Red: a case study on the spread and victims of an Internet worm. In *Proceedings of the SIGCOMM Internet Measurement Workshop '02* (Marseille, France, Nov. 2002).

[33] MOORE, D., SHANNON, C., VOELKER, G. M., AND SAVAGE, S. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proceedings of the INFOCOM '03* (San Francisco, CA, Mar. 2003).

[34] MOREIN, W., STAVROU, A., COOK, D., KEROMYTIS, A., MISHRA, V., AND RUBENSTEIN, D. Using graphic turing tests to counter automated ddos attacks against web servers. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security* (2003).

[35] PARK, K., AND LEE, V. S. P. K.-W. Securing web service by automatic robot detection. In *Proceedings of USENIX 2006 Annual Technical Conference* (May 2006).

[36] PAXSON, V. Bro: A System for Detecting Network Intruders in Real-Time . *Computer Networks 31*, 23.

[37] RANJAN, S., SWAMINATHAN, R., UYSAL, M., AND KNIGHTLY, E. Ddos-resilient scheduling to counter application layer attacks under imperfect detection. In *Proceedings of INFOCOM 2006* (May 2006).

[38] RIVERHEAD NETWORKS, INC. DDoS Mitigation: Maintaining Business Continuity in the Face of Malicious Attacks.

[39] SCHECHTER, S., JUNG, J., AND BERGER, A. Fast detection of scanning worm infections. In *Proceedings of the Seventh International Symposium on Recent Advances in Intrusion Detection* (Sep 2004).

[40] SCHECHTER, S. E., JUNG, J., AND BERGER, A. W. Fast Detection of Scanning Worm Infections. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection* (French Riviera, France, Sept. 2004).

[41] SHAW, M. Leveraging Good Intentions to Reduce Unwanted Network Traffic. In *Proceedings of SRUTI'06* (San Jose, CA, July 2006).

[42] SPITZNER, L. The honeynet project: trapping the hackers. *Security and Privacy Magazine, IEEE 1*, 2 (March 2003), 15–23.

[43] SPRING, N., MAHAJAN, R., , AND WETHERALL, D. Measuring ISP Topologies with Rocketfuel. In *Proceedings of the Special Interest Group on Data Communications (SIGCOMM)* (Pittsburgh, PA, Aug. 2002).

[44] SRISURESH, P., FORD, B., SIVAKUMAR, S., AND GUHA, S. Internet BCP draft: NAT Behavioral Requirements for ICMP protocol, Oct. 2007. Work in progress. draft-ietf-behave-nat-icmp-06.txt.

[45] STOICA, I., ADKINS, D., ZHUANG, S., SHENKER, S., AND SURANA, S. Internet Indirection Infrastructure. In *Proceedings of the SIGCOMM '02* (Pittsburgh, PA, Aug. 2002).

[46] TAN, P.-N., AND KUMAR, V. Discovery of web robot sessions based on their navigational patterns. *Data Min. Knowl. Discov. 6*, 1 (2002).

[47] THOTTAN, M., AND JI, C. Anomaly detection in ip networks. In *IEEE Trans. Signal Processing (Special issue of Signal Processing in Networking)* (August 2003).

[48] TRUSTED COMPUTING GROUP. TPM Specification Version 1.2.

[49] UNIVERSITY OF OREGON. RouteViews Project.

[50] VON AHN, L., BLUM, M., HOPPER, N. J., AND LANGFORD, J. CAPTCHA: Using Hard AI Problems For Security. In *Proceedings of EUROCRYPT'03* (Warsaw, Poland, May 2003).

[51] WALFISH, M., VUTUKURU, M., BALAKRISHNAN, H., KARGER, D., AND SHENKER, S. DDoS Defense by Offense. In *Proceedings of SIGCOMM '06* (Pisa, Italy, September 2006).

[52] WANG, X., AND REITER, M. K. Defending against denial-of-service attacks with puzzle auctions. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2003).

[53] WEAVER, N., STANIFORD, S., AND PAXSON, V. Very Fast Containment of Scanning Worms. In *Proceedings of the the 12th USENIX Security Symposium* (San Diego, CA, Aug. 2004).

[54] WILLIAMSON, M. M. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the 8th Annual Computer Security Applications Conference (ACSAC 2002)* (Las Vegas, NV, Dec. 2002).

[55] WONG, B., SLIVKINS, A., AND SIRER, E. G. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *Proceedings of SIGCOMM'05* (Philadelphia, PA, Aug. 2005).

[56] YAAR, A., PERRIG, A., AND SONG, D. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE Symposium on Security and Privacy* (Pittsburgh, PA, May 2004), pp. 130– 143.

[57] YANG, X., WETHERALL, D., AND ANDERSON, T. A DoS-limiting Network Architecture. In *Proceedings of the SIGCOMM '05* (Philadelphia, PA, Aug. 2005).