Title
Slicing Spam with Occam's Razor

Authors
Fleizach, Chris
Voelker, Geoffrey M
Savage, Stefan

Publication Date
2007-06-10

Peer reviewed

Slicing Spam with Occam's Razor

Chris Fleizach, Geoffrey M. Voelker and Stefan Savage
Department of Computer Science and Engineering
University of California, San Diego

Abstract

To evade blacklisting, the vast majority of spam email is sent from exploited MTAs (i.e., botnets) and with forged "From" addresses. In response, the anti-spam community has developed a number of domain-based authentication systems – such as SPF and DKIM – to validate the binding between individual domain names and legitimate mail sources for those domains. In this paper, we explore an alternative solution in which the mail recipient requests a real-time affirmation for each e-mail from the declared sender's MX of record. The *Occam* protocol is trivial to implement, offers authenticating power equivalent to SPF and DKIM and, most importantly, forces spammers to deploy and expose blacklistable servers for each domain they use during a campaign. We discuss the details of the protocol, compare its strengths and weaknesses with existing solutions and describe a prototype implementation in Sendmail.

1 Introduction

By almost any metric, spam email has become a pervasive blight on e-mail users and service providers alike. The low marginal costs of spam delivery combined with the effectiveness of early content-based filtering and domain-based blacklisting have led spammers to develop large-scale remailing infrastructures in response. Thus, a modern spam campaign can comprise hundreds of millions of messages, addressed from tens of thousands of domain names, and delivered via thousands of distinct Mail Transfer Agents (MTAs). Indeed, with some reports indicating hundreds of millions of compromised "bot" hosts on the Internet [Markoff, 2007], the ability to produce 100 million spam messages a day has become a trivial task.

In response, the anti-spam community has focused considerable attention on limiting domain address spoofing and, through it, the ability to create an effective large-scale spam mailing infrastructure. For example, the Sender Policy Framework (SPF) and DKIM systems allow receivers to validate if an email's "From" domain address is consistent with the source of the message (authenticated via digital signature or the IP address(s) of the sending MTA). While each of these approaches has its benefits, neither is in pervasive use today and at least some of the early adopters have been spammers themselves.

In this paper we present a real-time challenge-based authentication protocol called *Occam* based on an exceedingly simple algorithm: when an email arrives, the receiving MTA sends a validation query back to the server who "should" have sent the message (the MTA responsible for the domain claimed as the source). If this MTA responds that it has indeed sent the message, then all is well; if not, then the domain has been spoofed and the contents are likely a spam or phish. In a real sense, this is mail authentication stripped to its barest essentials.

We believe the Occam protocol offers two contributions over previous approaches. First, Occam is extremely simple to deploy. For all small to medium-sized domains, Occam can simply be enabled – with no site-specific configuration at all – and yet deliver equivalent authenticating power to SPF or DKIM. In Section 5.2, we discuss how large domains can use Occam. Second, because Occam is a challenge-based authentication system, it shifts the burden of mail authentication to the sender on a per-domain basis. Thus, to participate in the protocol a spammer must provide online server resources available for any domain they use and this server must be capable of answering queries about any e-mail sent from that domain. This requirement increases the infrastructure demand on the spammer and, moreover, the addresses of these servers must be exposed during a spam campaign, thus

becoming prime targets for blacklisting.

The remainder of this paper is structured as follows. In Section 2 we relate our approach to previous anti-spam solutions, and then in Section 3 we describe our proposed Occam protocol. In Section 4 we analyze how spammers might try to sidestep Occam were it deployed and highlight the strengths and limitations revealed by this evaluation. We describe our Sendmail-based prototype for small domains, along with how large domains could implement Occam in Section 5. We follow with an analysis of performance overhead in Section 6 and a conclude with a summary of our findings in Section 7.

2 Related Work

The prolific growth of unsolicited email messages, or spam, has transformed the Internet landscape over the past ten years. By 2006, industry estimates suggested that such messages comprised over 80% of all Internet email with a total volume up to 85 *billion* per day [MessageLabs, 2006, Keizer, 2006]. The estimated costs to industry are similarly staggering, amounting to 20 billion dollars worldwide lost in productivity in 2003 [Lyman, 2003]. In response, an enormous industry has been created to deal with and attempt to eliminate spam.

Anti-spam defenses fall broadly into two categories: those that focus on *spam content* and those that focus on *spam infrastructure*.

The former category is based on the premise that the contents of spam e-mails can be automatically differentiated from legitimate e-mail and filtered appropriately. The best known of these approaches is Bayesian filtering, popularized by Paul Graham [Graham, 2002], and now widely-supported in systems such as SpamAssassin [SpamAssassin.org, 2007]. In this model, each e-mail is broken into a series of features (e.g., words) that are in turn represented by the prior probability of finding this feature in spam vs legitimate e-mails. By regularly training the classifier on labeled data, these probabilities can adapt to the changes in both spam and legitimate e-mail content. A range of enhancements have been developed over the years [Drucker et al., 1999, Delany et al., 2005, Androutsopoulos et al., 2000], but today this basic model – aided by human tuned rules – remains at the heart of most content-oriented defenses.

However, content filtering of this kind requires regular training and tuning, and thus is most effective in large e-mail services.[1] Moreover, most learning algorithms are fragile against adversarial training or to mimicry attacks [Wittel & Wu, 2004, Graham-Cumming, 2004] and spammers have employed both to evade such filters. Finally, even the best systems misclassify a small fraction of spam e-mail and allow it to proceed unfiltered. Thus, by simply sending more variants, a spammer can still maintain a high delivery rate in spite of filtering. Indeed, the very success of content filtering has implicitly supported the increased total volume of spam today.

However, to support high spam volumes, spammers must in turn deploy large-scale mailing infrastructure. Thus, the second major class of anti-spam defense focuses on either detecting and blacklisting this infrastructure or minimizing its value. For example, popular blacklists like SpamHaus [Spamhaus, 2007] allow for community efforts to update and maintain databases of IP addresses that are found to act as open relays or as spam sending hosts. Previous studies have looked at the *completeness* of such blacklists. For example, using a "spam trap" as an oracle, Jung and Sit found that 80% of spam sources are eventually identified in *some* spam blacklist [Jung & Sit, 2004], results which are corroborated by Anderson et al. [Anderson et al., 2007]. However, recently Ramachandran et al. have presented evidence of spam mailings using large numbers of low-volume hosts – few of which may ever source enough traffic to be blacklisted [Ramachandran et al., 2006]. Occam is likely to help in such scenarios since spammers must offer up a capable server for each domain name forged and thus blacking efforts can be concentrated on this smaller set of servers.

Since spammers are focused on speed and volume, their MTA's are frequently specialized to this task at the expense of protocol robustness. Thus, another defense against spammer infrastructure involves detection of protocol non-compliance (e.g., invalid return paths or Message-IDs), or pushing the protocol outside its norms (e.g., greylisting) [Slettnes, 2004].

Yet another technique is to increase the burden on spammer's infrastructure by demanding solutions to "computational challenges". Computational challenge, or "proof of work", systems require some form of computation to be performed by the sender to legitimize a message [Back, 2002, Seltzer, 2003]. While this cost is minor to a client, it can significantly lower the rate at which a spam server can send. Consequently, the effectiveness of this approach ultimately depends on how its computational demands compare with the marginal cost for a spammer to obtain more relay servers and on the rate of return on a mailing [Laurie & Clayton, 2004]. However, since chal-

[1] In one recent report, it was stated that AOL filtered over 2 billion spam messages per day in 2003 in this manner [Pfleeger & Bloom, 2005].

lenges typically require MUA modification they have not become widely used. Occam has some similarities to these efforts since it is a challenge/response protocol, but rather than demanding computational effort, Occam demands information about the recently sent message.

Finally, to minimize the spammer's ability to spoof "From" addresses within their infrastructure, a number of systems use reverse DNS lookups to authenticate whether an e-mail originated from a legitimate source. For example, the Sender Policy Framework (SPF) [Wong & Schlitt, 2006] provides a DNS-based means for specifying rules about which IP addresses can be used to send e-mail from a particular domain. Similarly, the DomainKeys Identified Mail (DKIM) system allows MTAs to append a digital signature to each message they forward [Allman et al., 2007]. Thus, when a message is received, the receiver can retrieve the domain's verification key via DNS and validate the origin of the message.

However, each of these approaches entail some amount of administrative overhead, in crafting SPF policies or integrating DKIM signing into mail infrastructure. Consequently, in one recent study only 5% of domains had some SPF policy in place [Liu, 2007]. Moreover, while both approaches help secure the binding between an e-mail and the domain it was sent from, neither prevents spammers from acquiring their own domains and complying with the protocol. Indeed, some early anecdotal reports suggested that spammers were among the earliest adopters of SPF.

Occam is primarily a domain authentication protocol like SPF and DKIM. We believe it distinguishes itself by being very easy to deploy (no configuration required except for large sites) and by forcing spammers to deploy servers for each domain to handle Occam protocol challenges.

This sampling of anti-spam solutions is by no means a complete list, but we believe it represents a rough overview of the current approaches. We echo Leiba and Borenstein's observation that effective spam defense is multi-faceted [Leiba & Borenstein, 2004] and many of these techniques are complementary in practice.

3 Occam's Razor

Occam is a new protocol for combating the growth of spam email. Occam is motivated by the observation that most spam email contains forged addresses typically sent by botnets [Ramachandran & Feamster, 2006]. The strength of the protocol lies in its simplicity. Using Occam, receivers simply ask the sending

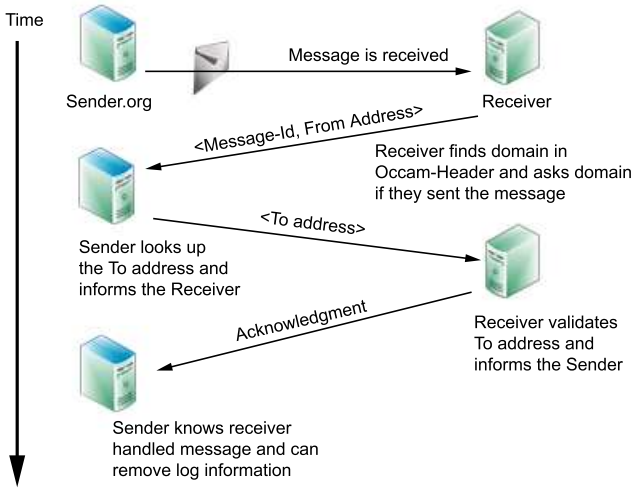

Figure 1: Outline of the Occam protocol for a valid email exchange.

domains identified in a message whether they actually sent the message. With legitimate email, the domains will acknowledge sending the message. Spam email that forges the domain, however, will not be acknowledged and receivers can classify the email as illegitimate. As a result, Occam requires spammers to provide available resources for acknowledging the spam they send, and induces spammers to control their own domains and identify those domains in their spam email. These requirements increase the resource burden on spammers and further expose spammers to effective blacklisting. In this section we focus on the operation and implementation of the protocol, and further discuss the implications of Occam in Section 4.

Figure 1 illustrates the operation of the Occam protocol. When a receiving server receives a message, it parses the *Occam-Header* to determine the sending domain. The Occam-Header looks like email address, with a user and a domain, but the domain specifies the server which should be contacted. In most situations, the Occam-Header will be identical to the envelope-sender, also known as the Return-Path. The following shows a sample Occam-Header.

```
Occam-Header:  bob@serverB.org
```

The receiver should only use the DNS MX records when resolving the domain. This requirement helps prevent using botnets as valid domain servers, since many botnets are desktop computers and generally do not have MX records (even though they may have A records). As a concrete example, assume that server A receives a legitimate message for the user *alice@serverA.org* that contains the Occam-Header *bob@serverB.org*. The Message-Id listed in the email message is *Id-1234*.

The receiver then sends a query message to the server for that domain. In this case, server B is the server that resolves to the MX record for serverB.org. The query includes the "From" address from the email together with the Message-Id:

```
From:  bob@serverB.org
Message-Id:  Id-1234
```

The domain server uses these fields to identify the message that was sent in a log of recently sent messages.[2] As a result, Occam requires each domain server to maintain a log of sent messages. Each log entry only needs to include the "To" address, the "From" address and the Message-Id. The "From" address is not always necessary, but provides information for a domain server to identify potentially abusive clients that try to guess Message-Ids and subvert Occam. The domain server can expire the log entries when the receiver acknowledges the response from the server. These fields can be compressed to about 30 bytes per record, so that even an outstanding log of 100 million entries would only require 300 MB, small by any standards for contemporary servers.

If the domain server finds the message in the log, it returns the "To" address of the corresponding message back to the receiver as acknowledgement. The acknowledgment allows the sending server to remove the entry from its log. In our example, server B responds with:

```
To:  alice@serverA.org
```

Note that since the domain server synchronously sends the response to the receiver using the same socket binding as the query request, it does not need to specify the Message-Id; the receiver knows which request the response matches.

The receiver validates that the "To" address from the domain server matches the "To" address in the original email message; requiring the "To" address in responses prevents malicious domain servers from simply acknowledging all Occam queries. If the "To" addresses match, then the receiver finally delivers the message to the user's mailbox and acknowledges the match to the domain server:

```
Status:  Received
Message-Id:  Id-1234
```

The domain server can now remove the corresponding entry from its log of messages sent. Note that, in this exchange, the receiver asynchronously acknowledges the match to the domain server and must explicitly specify the Message-Id.

[2]Continuing the theme of simplicity, Occam is a text-based protocol.

These exchanges correspond to the case when the domain sender has sent legitimate email. There are three cases where the protocol detects illegitimate email. First, if the domain server does not find the message in its log when queried, it responds to the receiver accordingly:

```
Status:  Unknown
```

Or, if the domain server does not find the message logged and it wants to know who the message was sent to, it can respond to the receiver asking for the "To" address:

```
Status:  Unknown To
```

The receiver can then respond with the "To" field from the message and remove the email as it sees fit. In our example, server A would return:

```
To:  alice@serverA.org
```

Second, if the receiver finds that the "To" address does not match what was in the email, then the receiving server concludes that the sending server did not actually send the message. In both cases the receiver can then take appropriate action against the illegitimate email.

Third, if the receiving server does not immediately receive a response from the domain server, it can limit the rate of querying, much like SMTP does when repeatedly trying to deliver messages to bad addresses. After each attempt that does not succeed, the receiving server doubles the amount of time to wait before retrying. After a timeout, the messages can be marked as illegitimate if no response was ever received.

4 Slicing spam

Having described the operation of the Occam protocol, we now discuss how spammers might respond to Occam, and what constraints and burdens Occam places on spammers for them to continue to deliver spam successfully. We then discuss the advantages Occam offers compared to current methods, as well as its limitations. Finally, we discuss how Occam can be incrementally deployed in the Internet today.

4.1 How might spammers respond?

The goal of Occam is to impose a substantially higher resource burden on spammers, and to further expose spammers to effective blacklisting. Naturally, spammers will respond to Occam and change how they deliver spam.

Put the bots to work. Occam requires senders to validate and acknowledge the email they send. Spam-

mers could try to distribute this load across the bots they already use to send out their spam and use the existing domain name for the bots in the Occam-Header. Occam, however, identifies senders using only the MX records to resolve domain names for servers. As a result, Occam makes it challenging for spammers to use generic bots for this purpose. Since many bots are hosts that will not have MX records that resolve to them, such bots would not be able to validate spam that they send. The implication is that bots cannot be used for the crucial step of validating messages with Occam, although they can, of course, continue to send messages.

In response, spammers could compromise or purchase bots that do have appropriate MX records, but harvesting specialized bots increases their cost and diminishes spammer profits. Alternatively, spammers could establish a DNS domain structure where each bot is assigned a separate sub-domain or entirely new domain. Spammers could create MX records to have the domains used in the Occam-Header resolve to the bots spending the spam. Such an elaborate DNS domain structure, though, makes spammers more vulnerable to blacklisting and increases cost. If spammers use many sub-domains, one per bot, the sub-domain structure would create a telling signature that the entire domain is being used as a source of spam. Given this signature, all of the sub-domains could then be easily blacklisted by blacklisting the entire domain. Spammers could use many domains instead of sub-domains, but doing so greatly increases the cost and management burden of managing the bots for sending spam. Further, the list of domains directly expose the identity of the bots and expose them to blacklisting. Finally, allowing bots to respond to Occam queries assumes they can accept incoming connections on low numbered ports, a policy which many ISPs do not allow.

Centralization. Rather than distributing the load across many bots, spammers could instead use a centralized server to handle the request load for validating the spam messages they send. In this scenario, spam would have an Occam-Header that resolves to this server. This server would then acknowledge requests from receivers so that spam could successfully be delivered. Spammers could still successfully deliver spam, but Occam forces this server to stay online as spam is sent. It increases the complexity and cost of managing and operating a spam campaign, and the server becomes an obvious target for blacklisting.

With Occam, spammers have to keep track of the email targets that they distribute to each of their bots. Because Occam requires the validating server to respond with the "To" address used in the original email, the validating server cannot blindly acknowledge all requests from receivers. Instead, spammers must precompute and distribute Message-Ids with "To" and "From" addresses to the bots being used for spam relay. The validation server must keep this list so that it can successfully reply to receiver requests. And spammers must provision the server so that it can handle a validation request load that grows in proportion to the number of spam messages sent. A spammer can use a botnet to send out millions of spam emails, but for them to be successfully delivered the spammer will need a server that has the resources to handle responding to millions of validation requests. As a result, Occam shifts the resource burden from the receiver to the spam sender. In other approaches, such as DKIM, the burden remains on the receiver, which must download keys and cryptographically validate each message. Furthermore, Occam requires spam senders to keep their validating server available and responsive during a spam campaign. Concentrating validation on a centralized server exposes that server to blacklisting. Once the server is associated with spam, adding the server to a blacklist will prevent spam validation and thereby delivery. Since spam campaigns can persist for days [Anderson et al., 2007], early blacklisting can substantially impede spam delivery.

DDoS Reflector. Finally, spammers could use the protocol as a reflector DDoS attack. A spammer could send millions of messages claiming to be from a targeted domain identified in the Occam-Header. As a result, receivers will then direct millions of validation requests to that domain. If a site has multiple MX entries in DNS, this could result in a multiplicative increase in the number of queries. Larger sites could likely handle this load, but smaller sites could be overwhelmed (we show in Section 5.2 how larger sites can avoid the amplification problem). However, when overloaded, a site could just as easily start dropping requests and rely on Occam's backoff and retry mechanism to distribute the load over time. More generally, though, if attackers want to use Occam to launch DDoS attacks, it would be easier for them to launch the attacks directly rather than use Occam. If an attacker can send out millions of messages, they are capable of a straightforward DDoS attack on the domain rather than performing an indirect attack through Occam. Indeed, they could simply send millions of messages directly to that domain, independent of whether Occam is used or not.

4.2 Advantages

The Occam protocol offers a number of distinct advantages over other methods that are currently in use.

Ease of administration. Occam does not require effort by administrators to make the system work, an important consideration for the thousands of small domains that may not have the technical expertise for more complex approaches. To use DKIM, for instance, domain administrators must create and insert a public key into a special DNS record. They then must configure the outgoing MTA to append signatures to all messages using a private key on each message. Presumably, they would also want to set up the MTA to handle incoming mail using DKIM as well. This process requires a certain degree of proficiency that may not be available for many small domains. The Occam protocol, however, can be implemented directly in MTA software packages, such as Sendmail, qmail or Microsoft Exchange Server. It can then be rolled out into a software upgrade, a process that is more familiar to users. We note that Occam is trivial to deploy for a small domain [3]. Obviously, larger domains will need a more involved process to implement Occam than compared to SPF, which only requires a DNS entry to be inserted. We argue that adoption of a protocol depends equally on its acceptance by small and large domains and Occam makes this process easy for the small ones. Larger domains are in position to roll-out their own deployment.

Enhanced culpability. The Occam protocol enhances what approaches like SPF and DKIM can accomplish. Both systems validate that a message came from a specific domain or that a sender is authorized, but the burden of proof rests with the receiver; again, with DKIM, the receiver must perform a cryptographic operation on each message. Moreover, the approach is still open to abuse. For instance, a spammer can just as easily set up a domain that has a perfectly valid SPF rule that specifies any IP address can send mail for the domain. A botnet can then send an unlimited number of messages that all look legitimate from the standpoint of SPF. They could alternatively find "open SPF relays" that allow any sender to send messages. This undermines the values of blacklisting domains based on SPF abuse. Occam shifts more of this burden to the spammer. It forces the actual sender of a message to be involved in it's authentication in an online manner. Legitimate hosts stay online and available as a matter of course. Spammers have gone to extreme lengths to avoid being detected and pinned down to a valid online presence. Thus Occam makes spamming more difficult to accomplish without creating an exposed and more expensive centralized infrastructure. Occam, in effect, undermines the value that botnets provide to spammers.

[3]Occam is also trivial to implement, as described in Section 5.1.

Real-time validation. Occam requires that the "work", in our case responding to a validation query, be performed online by the sender of the message. This requirement contrasts with protocols like Hashcash, where the "work" can be precomputed during idle time across thousands of botnets before any spam campaign begins. With Occam, the spammer must be able to respond successfully to all the queries that arrive in real-time. The effect of responding in a timely fashion is that the spammer must have a valid domain name that resolves to a server in their employ. This server must be available to accept queries on the Occam port and be provisioned well enough to respond to many queries. The Occam protocol forces the spammer to expose this higher value target, presumably more expensive to obtain, and makes the domain and IP used an easy target for blacklisting. If the spammer attempts to switch to a different IP address, the domain still remains blacklisted. Since the spammer must own that domain, blacklisting a domain can no longer affect the credibility of domains that are normally "hijacked," as is done currently.

Input for reputation systems. As mentioned above, a spammer could register many domains and keep changing DNS records so that they point to new servers able to answer queries. However, these rapid DNS changes would create a telling signature in their short TTL and IP churn. According to [Taylor, 2006], webmail services are establishing reputations for domains that allow them to filter spam more effectively. These two characteristics, IP churn and short TTL, would be clear indications that the domain was involved in sending spam, evidence that reputation systems could use to reliably identify spamming domains.

Anti-Phishing capability. An unexpected benefit of using Occam is that domains will immediately become aware of when they are being phished (or, more generally, being spoofed). Since receivers will begin querying a spoofed domain for non-existent messages, Occam enables domains to discover immediately when they are being spoofed. Moreover, Occam provides a mechanism for a receiver to determine the "To" address to which a phishing email was sent. Such information would be useful to companies that must often deal with phishing attacks, as it allows them to flag accounts to watch for suspicious activity or to take other measures to contact the users that they know have been exposed. The ability to be notified immediately of phishing and spoofing would consequently be available not only to large and well-funded companies, but any organization, thereby reducing the effectiveness of more elaborate attacks like spear-phishing.

Phishers could try to avoid having the original domain

know about the phishing attempt by specifying one of their domains in the Occam-Header. However, aside from the difficulties spammers would have in using their own servers, the discrepancy provides a strong indication that a message is illegitimate if the domain in the Occam-Header lies outside of the top level domain for an organization.

Low overhead. The Occam protocol is simple to implement and straightforward to deploy. It also imposes low overhead to operate. The overhead is proportional to the number of messages received and sent, imposing little additional burden on both small and large sites. We further explore Occam overhead in Section 6 by experimenting with a prototype Occam implementation.

4.3 Disadvantages

As with any approach, Occam has disadvantages as well, which we discuss below. For the large majority of domains, though, we believe the benefits of Occam outweigh these disadvantages.

Mobile mailers. There are some legitimate reasons that a sending server might not be able to respond to an Occam query. One is to retain the ability to send mail from a host intermittently connected to the Internet, while allowing another server to handle incoming mail and SMTP related functions, like error messages. We believe this flexibility in SMTP is abused by spammers and that it is in the best interest of most servers to exert greater control over who is allowed to send mail claiming to be from their domains.

Denying service. The Occam protocol also opens up a potential denial-of-service attack against email receipt. An adversary could potentially try to query for and acknowledge email requests from a sending server in an attempt to make them remove their logs prematurely, thereby preventing delivery of the email by the receiver to the original recipient. As an example, if server A sends a message to server B, a malicious server C could try to guess the Message-Id and the From address and reply more quickly to server A than server B does. Server A would acknowledge sending the message and remove its log information about the message, causing a subsequent validation by the real receiver, server B, to fail.

However, precisely since the Message-Id and the From address are required information for querying a server, there is a reduced chance such an attack would succeed since an attacker has to guess these fields. Correspondingly, it would benefit all MTA software to add more entropy to the Message-Id fields. Further, most queries by the legitimate receiver would happen in a relatively short amount of time, limiting the window of opportunity of an attacker. Finally, a sending server could keep the necessary information around for some period of time after it has been successfully queried before removing it.

4.4 Deployment

Full benefit of the Occam protocol necessarily requires the cooperation of almost all domains sending email. However, domains can gain incremental benefits from incremental deployment, much as DKIM has been deployed. Initially, pairs of domains would recognize that both are observing the protocol through out of channel mechanisms or probing procedures, and would then make arrangements to use the protocol. For example, if eBay began observing the protocol, Google mail would be able to verify all mail that purported to be originating from eBay. Over time, Occam implementations could be rolled into future upgrades of popular mail transfer agents that administrators install. If domains see that a certain site is not using the protocol, they can use their existing approaches for determining and dealing with spam as usual; and the fact that the message could not be validated using Occam could be additional evidence in the spam filtering decision process. If Occam use reaches critical mass, then domains could assume that any site not using the protocol should be regarded as an abusive server and its messages ignored outright.

5 Implementation

We discuss our experiences in designing a prototype implementation of Occam that would work well for small to medium sized domains. We then explore how large domains can implement Occam without incurring synchronization and centralization overhead.

5.1 Prototype implementation

We implemented a prototype of Occam to assess the feasibility of the protocol and evaluate its overhead. The Occam prototype consists of a daemon that operates in cooperation with an MTA. We used the Sendmail MTA [Sendmail Consortium, 2007] due to its popularity and available source code. To ease the development of a prototype, we did not fully integrate our entire implementation within Sendmail or use a Sendmail milter. Instead, we made the minimal changes to Sendmail as necessary to delegate operation of the Occam protocol to a separate application. In all, we only made changes to five lines of Sendmail source code across two files. When Sendmail sends a mail message,

the new code logs the "From", "To", and Message-Id fields to a file. When Sendmail receives a message, it uses a Sendmail operation to place the file containing the message into quarantine. If the Occam daemon determines that the message is legitimate, it will move the message file out of quarantine. Sendmail will then deliver the message to the user's mailbox without further changes.

The Occam daemon operates independently of Sendmail, consuming the records of sent messages produced by Sendmail that are stored in the log and scanning for quarantined messages. It is a multi-threaded C++ program that implements both a client and server mode, with a thread executing each mode. When the server starts, it binds to UDP port 25 and drops root privileges. It then loads in the sent messages from the log file. When it receives a query, it first checks the log file to see if updates have been made before checking its own cache. When expiring a queried entry, it clears its memory cache of the entry and removes the entry from the log file. Using a database to manage the log entries could simplify the implementation further.

The client mode of the daemon continuously scans the mail queue for messages that Sendmail has quarantined. When a new message appears, the daemon opens it and retrieves the Occam-Header, "From" address, "To" field and Message-Id. It then parses the Occam-Header to obtain the domain. It looks up the MX records only for that domain and stores them in order of their MX priority. For each server in that list, it queries the server to find out whether the server was the sender of that message. If a server acknowledges sending the message, it removes the quarantine status of the message by changing the first letter of the filename. It then informs Sendmail of the update by invoking it with the "-qf" flag to cause Sendmail to check its queue of updated messages. If no server acknowledges sending the message, the daemon unlinks the message file to dispose of it.

The Occam daemon uses UDP for its transport protocol, following the example of DNS which has successfully used UDP to implement an efficient and reliable service for a critical application. Most message exchanges require one UDP packet, with particularly long Message-Ids requiring two. Although using TCP would ensure messages were received by both parties, UDP does not tie up connection resources as TCP does, and does not require extensive handshaking before sending messages. If overhead is of less concern, though, Occam could use TCP instead.

The daemon has a short and simple implementation consisting of 1,000 lines of code: 300 for the server, 400 for the client, and 300 shared among the two modes. Given its ease of implementation, it should be straightforward to extend any MTA with an implementation of the Occam protocol.

5.2 Implementation in large domains

For sites that handle much larger volumes of email, scaling could add complexity to the implementation, but need not. Moreover, these sites likely have support to address scaling issues already. The Occam protocol requires only basic logging and querying functionality, operations that can be streamlined with database servers if necessary. Overhead due to the Occam protocol would not then be significant for domains already sending large quantities of email. More significantly, Occam imposes the same requirements on spammers: large-volume spammers would have to similarly scale their logging and querying ability, often under the constraints of using bots. This is another operation that reduces their return on investment, making spamming less profitable.

Obviously, our prototype implementation is unsuited for domains that have dozens or hundreds of mail server. These domains however have a few options. They could centralize their logging and query operations, although doing so may be awkward for large sites. Instead, large domains can balance the logging and querying operations in a straightforward fashion. A mail server sending mail can add an Occam-Header that points back to that specific server, instead of the entire domain. Doing so distributes load naturally and groups sending mail with answering queries on the same server, requiring no coordination among large, distributed mail server farms. This solution also eliminates the amplification DDoS attack. Each of these mail servers does not need to have multiple MX values for it. The receiving mail servers, who do not respond to Occam, can then refuse incoming Occam queries, and these packets can be dropped at border gateways.

6 Estimated Impact

An important consideration for any new protocol is the impact it would have on the current Internet infrastructure and the servers responsible for deploying it. The Occam protocol does have the potential to raise bandwidth costs and server utilization. We argue that these costs, however, have minimal impact.

First we examined the time overhead for our Occam prototype implementation compared to similar approaches, DKIM and SPF. We measured the time to send 1,000 messages for all three implementations, and used Sendmail 8.13.8 as the MTA for all exper-

Configuration	*Time (s)*
Sendmail	100.40
Sendmail with OpenSPF	100.03
Sendmail with DKIM	251.23
Sendmail with Occam	102.4

Table 1: Average time to send 1,000 messages using Sendmail.

iments. The DKIM implementation ran dk-milter 0.4.1 [domainkeys-milter, 2007] and the SPF implementation ran with smf-spf 2.0.2 [Kurmanin, 2007] using libspf2 1.2.5 [libspf2, 2007]. We used two servers for sending and receiving email, respectively, and a third that handled DNS requests for both MTAs. The DNS server had the following records for SPF and DKIM:

```
SPF: v=spf1 a -all
DKIM: k=rsa; t=y;
   p='MHwwDQYJKoZIhvcNA...'
```

We sent 1,000 messages using a Perl script from one server to the other five times. Table 1 shows the average time for each experiment, with insignificant variation across runs. The results show that the Occam protocol does not add significant overhead and is comparable to sending without additional measures and sending with SPF enabled. We also found that DKIM adds a significant computational overhead.

The communication overhead of Occam is proportional to the amount of email, particularly spam email, sent on the Internet. Given the amount of spam sent on a daily basis, this overhead might be substantial. To estimate the overall communication overhead on the Internet, we can perform a back-of-the-envelope calculation to indicate the added data costs that the protocol would impose. If we take an upper estimate on the number of email messages sent [Radicati Group, 2006], there were 171 billion messages delivered daily in the first quarter of 2006, of which 71% were spam. If every one of those email messages required three UDP packets to determine their status, plus one MX record lookup, where we assume the packet size is a generous 200 bytes, then 800 bytes would be needed by Occam per email. This overhead would add 136 TB of total data into the Internet per day. Spread out over an entire day, the load averages 1.58 GB/s, a very small rate at Internet scales.

7 Conclusion

The Occam protocol provides a simple light-weight mechanism for authenticating e-mail messages. Its simplicity makes it easy to understand and, as well, easy to administer – requiring no site-specific configuration. Moreover, spammers who would chose to adhere to the protocol are forced to support and expose dedicated infrastructure for the duration of their campaign. Finally, as a side-effect, the Occam protocol notifies domain owners when their addresses are being spoofed, which is useful for combating phishing attacks. Occam is not a silver bullet for solving the spam problem and, like most anti-spam technology, is most effective in tandem with existing approaches including spam filtering, reputation systems and blacklisting services. However, we believe Occam's advantages make it a valuable addition to the repertoire of weapons in the fight against spam.

References

Allman, E., Callas, J., Delany, M., Libbey, M., Fenton, J., & Thomas, M. (2007). DomainKeys Identified Mail (DKIM) Signatures. RFC 4871 (Proposed Standard).

Anderson, D. S., Fleizach, C., Savage, S., & Voelker, G. (2007). Spamscatter: Characterizing Internet scam hosting infrastructure. *16th USENIX Security Symposium (Security'07)*.

Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Sakkis, G., Spyropoulos, C., & Stamatopoulos, P. (2000). Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *In Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'00)*.

Back, A. (2002). Hashcash - a denial of service counter-measure. `http://www.hashcash.org/papers/hashcash.pdf`.

Delany, S. J., Cunningham, P., Doyle, D., & Zamolotskikh, A. (2005). Generating estimates of classification confidence for a case-based spam filter. In *Case-based reasoning research and development*. Springer Berlin / Heidelberg.

domainkeys-milter (2007). domainkeys-milter. `http://sourceforge.net/projects/dk-milter/`.

Drucker, H., Wu, D., & Vapnik, V. N. (1999). Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*.

Graham, P. (2002). A plan for spam. `http://www.paulgraham.com/spam.html`.

Graham-Cumming, J. (2004). How to beat an adaptive spam filter. *MIT Spam Conference*.

Jung, J., & Sit, E. (2004). An empirical study of spam traffic and the use of DNS black lists. *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement* (pp. 370–375). New York, NY, USA: ACM Press.

Keizer, G. (2006). Spam volume jumps 35% in November. *InformationWeek*. http://informationweek.com/news/showArticle.jhtml;?articleID=196701527.

Kurmanin, E. (2007). smf-spf Sendmail SPF milter. http://smfs.sourceforge.net/smf-spf.html.

Laurie, B., & Clayton, R. (2004). Proof-of-work proves not to work. *The Third Annual Workshop on Economics and Information Security*.

Leiba, B., & Borenstein, N. (2004). A multifaceted approach to spam reduction. *In Proc. of the Conference on Email and Anti-Spam (CEAS'04)*.

libspf2 (2007). libspf2 - SPF libary. http://www.libspf2.org/.

Liu, C. (2007). Handicapping new DNS extensions and applications. http://www.onlamp.com/pub/a/onlamp/2007/01/11/dns-extensions.html.

Lyman, J. (2003). Spam costs $20 billion each year in lost productivity. *TechNewsWorld*. http://www.linuxinsider.com/story/32478.html.

Markoff, J. (2007). Attack of the zombie computers is a growing threat, experts say. *New York Times*. http://www.nytimes.com/2007/01/07/technology/07net.html.

MessageLabs (2006). 2006: The year spam raised its game and threats got personal. http://www.messagelabs.com/publishedcontent/publish/about_us_dotcom_en/%news___events/press_releases/DA_174397.html.

Pfleeger, S. L., & Bloom, G. (2005). Canning spam: Proposed solutions to unwanted email. *IEEE Security and Privacy*.

Radicati Group (2006). Worldwide daily email traffic climbs to 171 billion messages, spam rises to 71 percent, says radicati group. http://www.tekrati.com/research/News.asp?id=6933.

Ramachandran, A., Dagon, D., & Feamster, N. (2006). Can DNS-Based blacklists keep up with bots? *In Proc. of the Conference on Email and Anti-Spam (CEAS'06)*.

Ramachandran, A., & Feamster, N. (2006). Understanding the network-level behavior of spammers. *Proceedings of the ACM SIGCOMM Conference*. Pisa, Italy.

Seltzer, L. (2003). Challenge-response spam blocking challenges patience. *PC Magazine*.

Sendmail Consortium (2007). Sendmail. http://www.sendmail.org.

Slettnes, T. (2004). Spam filtering for mail exchangers: How to reject junk mail in incoming SMTP transactions. http://tldp.org/HOWTO/Spam-Filtering-for-MX/index.html.

SpamAssassin.org (2007). SpamAssassin. http://www.spamassassin.org.

Spamhaus (2007). The Spamhaus project. http://www.spamhaus.org.

Taylor, B. (2006). Sender reputation in a large webmail service. *In Proc. of the Conference on Email and Anti-Spam (CEAS'06)*.

Wittel, G., & Wu, S. (2004). On attacking statistical spam filters. *In Proc. of the Conference on Email and Anti-Spam (CEAS'04)*.

Wong, M., & Schlitt, W. (2006). Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408 (Experimental).