ASK: Active Spam Killer

Marco Paganini
paganini@paganini.net
www.paganini.net

Appear in the FREENIX track of the 2003 Annual Usenix Conference

Abstract

We present Active Spam Killer (ASK), a program that attempts to validate unknown senders before allowing delivery of their message. Validation occurs by means of a challenge reply sent to senders who are not yet known to the system. Messages are kept in a queue pending confirmation until the sender replies to the challenge. Further messages coming from confirmed senders are delivered immediately. In a sample of 1000 spam mails, ASK was 99.7% effective at blocking spam, resulting in only 3 spam messages being delivered. Other programs' best ratios were 97.8% or as many as 22 spam messages delivered.

1 Introduction

Unsolicited Commercial Emails (UCEs), also commonly called "spam" represent a serious problem to most Internet users, who are constantly bombarded with all sorts of scams, promotions, and offensive material. This situation has prompted the creation of many different tools to eliminate the problem, with varying degrees of success.

The most common way to deal with spam is to parse incoming mails and decide whether they should be delivered or not based on their contents. Reasonable results have been obtained with this technique, but the complexity and diversity of human languages make it a difficult task. Such *content filtering* tools propose to attack one of the weak spots in spam: the message content itself.

The effectiveness of such tools can be drastically reduced when the incoming mail employs an unknown language or even an unknown character set. Also, cleverly crafted emails may never be detectable as the difference between those and perfectly valid emails is subtle.

Active Spam Killer (ASK) proposes to attack a different weak spot in the spam chain: the validity of the sender's email address. When a message from an unknown origin is received, a challenge (also known as *confirmation message*) is sent back to the mail originator. This message contains brief instructions to the sender on how to get authenticated into the system and cause delivery of the original message. The confirmation message is crafted in such a way that a simple reply keeping the "Subject" line intact will suffice. The confirmation message also contains a unique MD5 [16] hash computed by combining the contents of the original email with a secret key known only to the recipient. This prevents false confirmation returns as the code is based on the unique characteristics of the receiver.

The message remains stored in the *pending mail queue* until a *confirmation return* is received (a reply to the confirmation message with the MD5 hash in the "Subject" header). When that happens, ASK checks the pending mail queue for a message whose MD5 signature matches the one in the confirmation return. If found, the original message is delivered from the pending mail queue and the sender's email address is automatically added to the whitelist. This sender has now been validated and all future messages from this address will be immediately delivered.

The sender's address can also be added to two other lists: the *ignorelist* and the *blacklist*. The first causes emails to be silently ignored while the second not only ignores the email but also sends a message back to the originator explaining that future emails coming from this address are blocked.

This *challenge-authentication* scheme guarantees that delivered emails always come from valid senders. Unwanted (but technically valid) senders can be easily ignored as they offer a simple key to their detection: their own email addresses.

Although an auto-responder could be used to defeat this method, this is unlikely as it would expose the sender to legal complications, account and service cancellations, and a fairly large number of ”Invalid Address” response messages during normal operation.

The rest of this paper is organized as follows: Section 2 provides background information on popular anti-spam techniques. Section 3 details ASK's design and operation. Section 4 compares the effectiveness of ASK against other anti-spam tools. Section 5 surveys other challenge-based solutions. Finally, Section 6 presents our conclusions and future work.

2 Background

Compared to the traditional method of mailing printed propaganda, spam costs almost nothing. No mailing infrastructure is needed except for an Internet account and a personal computer. A one or two percent return on investment potential applied to millions of emails results in a number large enough to justify the approach to a number of people.

The first ingredient is a large number of email addresses, usually obtained by harvesting addresses off the Internet. A study conducted by the Federal Trade Commission concluded that 86% of all addresses posted to Web pages and to the newsgroups received spam [5]. Anything resembling an email address is collected and used. Those without the resources to mine the addresses themselves can purchase CDs containing ready-to-use lists of addresses.

Next, a *Mass Emailer* like DynamicMailer [3] is used to deliver the message to the list of addresses. These programs normally provide certain facilities like header forging, sending emails directly from the user's workstation (bypassing the need for an SMTP server) and others.

Spammers must always act as inconspicuously as possible to avoid account cancellation, revenge from angry users, and legal problems. The sender's email address is often forged to an invalid address or to a throw-away account from a free web-mail provider. These accounts are rapidly filled with bounce messages as a result of outdated and invalid emails in the list.

Instructions on how to obtain the product or service offered are normally contained in the body of the mail, with URLs pointing to the Web page of the seller.

Next, we outline the most common anti-spam techniques in use today.

2.1 Realtime Blackhole Lists

Misconfigured mail servers will relay any incoming message to any destination without control. These servers can be used to deliver mail indiscriminately and in many cases, anonymously.

Realtime Blackhole Lists (RBLs) are online databases containing the IP address of such servers. These databases are queried by *Mail Transfer Agents* (MTAs) upon receipt of a new message. If the IP address originating the connection is listed in the database, the connection will be terminated and the appropriate error code will be sent. Some RBLs also keep dial-in IP addresses in an attempt to detect messages sent directly from dialup workstations.

Many such databases exist today, with *MAPS* [9] being a well known provider of such services. Most MTAs support this feature with minimal configuration effort.

RBLs are not very effective if used as the only spam prevention method as they can only block mail (spam or not) coming from spammers who use open relays to send their messages. Even though some RBLs specifically list dial-in lines, often it is not practical to use those as many legitimate users send emails directly from their workstations over dialup lines.

Some RBL providers have complicated procedures to remove entries from their databases, leading to a substantial number of false positives (See Section 4.1). Also, there are normally no whitelist mechanisms available to regular users, meaning that it is impossible to grant access to a certain known valid email or IP without supervisory privileges.

2.2 Content Filtering Tools

Tools in this category try to detect spam by investigating the content of received emails.

Early attempts used simplistic keyword filters, normally implemented with generic mail processing tools. Specific strings like "Dear Sir" in the body of the email would flag the message as spam. This produced inaccurate results and a fairly large number of false positives.

A more sophisticated approach to the problem employs *scoring* of certain keywords, sentences and characteristics pertinent to spam mail. A simple "Dear Sir" *might* indicate that we are dealing with spam, but "Dear Sir," "Click Here," and "Call now" in the same email message are a very clear indication. SpamAssassin [20] utilizes this technique.

Usually, a complex set of rules exists with a positive or negative numeric score assigned to each rule. Rules that clearly indicate spam receive high values. Negative values reduce the probability of the email being classified as spam (even in the presence of other clear indications). A rule to match a string like "Dear Sir/Madam" will have a lower score than "Work from home" as the latter is a clearer indication of the common home employment Internet scams. Expressions like "Usenix" and "Algorithm" would receive a negative value as they strongly suggest that this is not a spam mail. The mail will be marked as spam if the sum of all matching scores exceeds a user-defined threshold.

There are specific and distinct rules for the email headers and body. The rules applied to the headers will be equally effective, regardless of the language used to send the email. Body analysis however is greatly affected by language: a set of rules that perfectly detects emails in the English language could completely miss emails written in Italian or Korean.

A new variation of this technique, employing a naïve Bayes classifier to isolate junk mail has become quite common [17]. Bogofilter [15] is a popular tool in this category.

These tools can be trained to learn about good and bad combinations of tokens and their probabilities of occurring together in incoming mails. As they learn more about what is spam and what is not, the chances of correctly detecting junk emails increase.

Bayesian classifiers suffer from the same problem as the other content filtering approaches: a well-crafted message body is likely to pass unharmed, as few identifiable elements are present. Messages without a text body (for example, advertising as an attached image file) might pose a problem too since very little textual content is available for analysis [10].

To illustrate how difficult it is to classify e-mails as spam based solely on content, consider the following email:

```
From: abcd12345@domain.com
To: yourname@yourdomain.com
Subject: Check this out.

Hi,

I found a tool called ‘‘CleanUp’’ at
(http://www.example.com). It clears
the contents of the browser cache
and history lists so other users
won’t know the websites you have
been browsing. Very useful!
```

This could represent either a perfectly valid email (if coming from a known sender) or a clear indication of spam (if coming from an unknown sender). Analysis based solely on content is very difficult in this case, specially for someone without the knowledge about any previous association between the recipient and the sender.

2.3 Distributed Anti-Spam Networks

Spam messages are typically sent unaltered to a large number of recipients. Distributed Anti-Spam Networks attempt to curb spam by preventing its propagation. Vipul’s Razor [14] and the Distributed Checksum Clearinghouse (DCC) [18] are examples of programs using this technique.

Agents installed on the user’s workstation will generate *fuzzy signatures* of every incoming mail. These signatures ignore small changes in the text so that slight variations in the spam body or headers will still generate similar signatures.

Next, a centralized database is queried looking for the signature computed from the incoming mail. If a match is found, the email is assumed to be spam and is discarded.

Users are responsible for reporting spam to the database (normally by means of a special account where the spam mail should be forwarded). Once spam has been reported by one user, all others will be protected against that specific spam mail or similar ones.

Under normal circumstances, the chances of false positives is very small, as actual users report spam to the system. Also, the chances of catching spam that is already in the database is quite good.

This approach, however, can only identify spam once a previous (or similar) case exists, meaning that newly sent spam will not be detected. A reasonable number of spam reporters is required to make the system work reliably and the quality of the database is largely dictated by the quality of the reports. Invalid or incorrect reports could poison the database, causing valid emails to be reported as spam. There is also a scalability concern, given the centralized nature of the database servers containing the signatures.

Another point to note is that an extra TCP connection is needed to contact the database servers, making this a non-viable alternative for users behind restrictive firewalls.

2.4 Challenge-Based Authentication

Challenge-authentication agents work on the premise that mail should only be delivered after senders identify themselves. ASK, TMDA [11], and QConfirm [13] are examples of programs that employ such a technique (with variations). When a new mail is received, the sender is checked against a database of known addresses. If the sender is known, the email is delivered immediately. Otherwise, a challenge mail will be sent back to the originator requesting confirmation. Once the sender becomes known to the system, further messages coming from the same address will be immediately accepted.

This method exploits the fact that most spammers use invalid return addresses in their messages. Since no (or limited) text analysis is performed, it is impossible to force delivery of the message by crafting the message body to look like an innocent message.

Unlike the previous alternatives, in challenge-response systems, subsequent action is required from the sender to deliver the email. Spammers utilizing valid email accounts could reply to the challenge and get authenticated to the system. Special provisions exist to avoid mail-loops or sending challenges to mailing-lists.

3 Design

We designed ASK to be simple to install and readily available to the widest possible audience. Supervisory rights or re-configuration of the mail server is not needed, allowing regular users to install the program under their home directories.

ASK was developed in Python [7], an easy to read and portable language that has been gaining a lot of popularity lately. Python is available for most Unix variants, making ASK portable across many platforms.

ASK works by reading emails from the standard input. After processing, emails can be directly stored into the user's mailbox or sent to the standard output for post-processing by other mail filters. This makes the program compatible with a number of Mail Transfer Agents and Mail Filters, like Sendmail [6], Qmail [4], Exim [12], Postfix [21] and others. Support for Procmail [19] is also embedded in the program, as well as direct delivery to "mbox" and "Maildir" style mailboxes.

Emails pending confirmation (those for which no confirmation return has been received) are stored as individual text files. The file names contain the MD5 hash sent in the confirmation, minimizing CPU utilization when matching ordinary confirmation returns. The pending mail queue can also optionally follow the Maildir format, allowing Qmail users to access and manage their queues remotely via IMAP.

ASK is normally invoked by the user's `~/.forward` file mechanism or by procmail. Configuration is stored in the `~/.askrc` file and all the control files and spool directories are created by default under the `~/.ask` directory.

A flowchart of the program's operation can be seen on Figure 1. In the following paragraphs, numbers enclosed in squares represent references to the corresponding boxes in the flowchart.

Incoming mails are always checked against the whitelist $\boxed{1}$, blacklist $\boxed{2}$, and ignorelist $\boxed{3}$ (in this order). The lists are implemented as text files containing a set of regular expressions. Emails can be authenticated based on the sender's email address, the recipient's email address or the message subject. Plain substring comparison is also available for simple matches.

A match in the whitelist will cause instant delivery of the email $\boxed{16}$. If a match is not found in the whitelist, the program tries to match the email on the blacklist and then on the ignorelist. In either case, the original email is discarded $\boxed{19}$, but matching the blacklist means a warning message will be sent back to the sender indicating that further emails are blocked and ignored $\boxed{11}$.

The next step is to check for *mail bounces*, or error messages sent by other MTAs $\boxed{4}$. Special processing takes place to prevent the delivery of every bounce sent to an invalid sender. This is discussed in more detail in Section 3.1.

ASK provides a remote queue and list management control by email. To use this feature, users must send themselves an email containing certain commands in the "Subject" header. ASK checks for *remote commands* $\boxed{5}$ and executes the commands if appropriate. The complete set of remote commands with examples is discussed in Section 3.2.

The next step is to check for confirmation returns $\boxed{6}$. These messages contain a specific string in the "Subject" field, followed by the MD5 hash that uniquely identifies the message inside the pending queue. The MD5 hash is extracted from the incoming mail and used to form a file name that contains the original message. If such a filename exists $\boxed{14}$, the sender's email in the confirmation message *and* the one in the queued message are added to the whitelist $\boxed{17}$. The original message is removed from the queue and delivered $\boxed{18}$.

If ASK detects an invalid confirmation (for which no files exist in the pending queue) the message will be labeled as "Invalid Confirmation" and delivered. This measure prevents loops with other ASK users.

A special case that deserves attention is that of spam messages with the sender's address forged to be the same as the recipient. In this case, a confirmation message cannot be sent as it would in turn be sent to the ASK user. To avoid this, ASK implements the concept of a *mailkey*: a string or short phrase that must be present in every outgoing mail sent by the ASK user. Common choices are words or short phrases from the user's signature. ASK will first check for the presence of the mailkey in incoming mails $\boxed{7}$. If it is found, the email is immediately delivered $\boxed{16}$ and processing ends. If not, ASK compares the sender's address to the ASK user's address (configured at installation time) $\boxed{8}$. The message will be queued with a status of "Junk" if a match is found $\boxed{15}$.

The mailkey serves a second purpose: to minimize the number of confirmations sent to replies. Ordinarily, ASK has no means of knowing if a certain message is a reply to an email sent by the ASK user or not. Most MUAs, however, quote the entire original message in a reply. This gives ASK an opportunity to detect the mailkey in replies and deliver the message without a confirmation. ASK can also be configured to automatically add the sender's email to the whitelist if a message contains the mailkey.

Sending confirmation messages to mailing-lists would be undesirable. For this reason, ASK tries to determine if a message came from a mailing-list $\boxed{9}$ before a confirmation is sent. Mailing-list messages will be immediately queued $\boxed{15}$ and no confirmation will be generated. The set of heuristics used to detect mailing-list and other machine-generated emails is described in Section 3.3.

At this point $\boxed{10}$ the message has passed all the tests that could cause its delivery or dismissal:

- The message is not in the whitelist.
- The message is not in the ignorelist.
- The message is not in the blacklist.
- The message is not a bounce of any kind.
- The message is not a remote command request.
- The message is not a confirmation return.

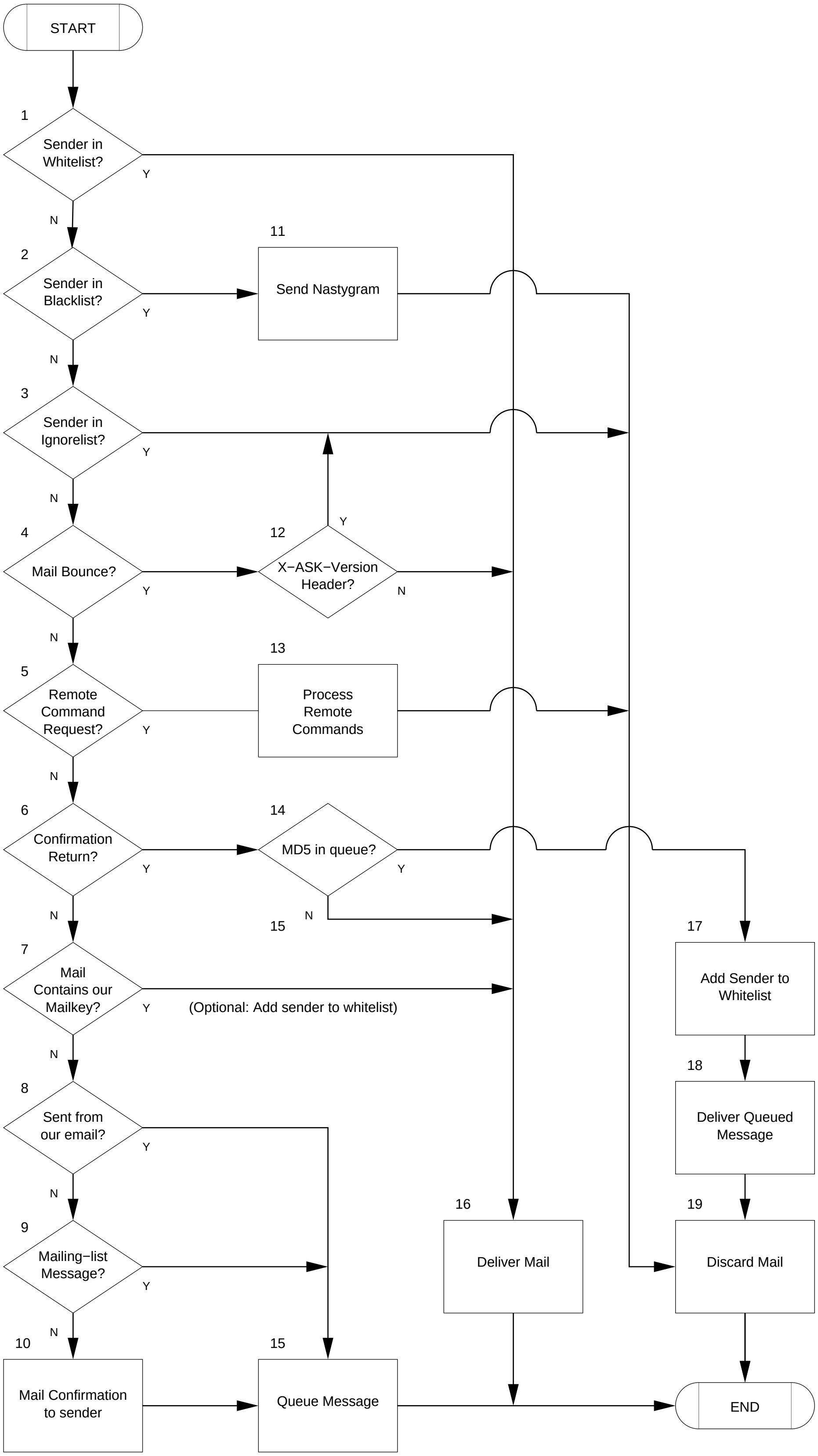


Figure 1: ASK Mail Processing Flowchart

- The message does not contain our mailkey.
- The message does not come from one of our own emails.

The final step is to compute the MD5 hash and send the confirmation message to the sender. This is discussed in detail in Section 3.4.

Next, we discuss special cases and features of ASK.

3.1 Bounce Treatment

A *Mail Bounce* is a message sent by the MTA when the email cannot be delivered for any reason.

For ASK's purposes, bounces can be broken down into three distinct types: regular bounces, forged bounces, and confirmation bounces.

Regular bounces occur due to a legitimate failure in the process of sending mails whereas forged bounces are actually spam messages with the sender set to "MAILER-DAEMON."

There is no way to distinguish between forged bounces and regular bounces without integrating ASK with the MTA, so both bounces are immediately delivered.

Confirmation bounces are generated when an error happens during the delivery of a confirmation message. These are very common as most spammers forge their emails to contain invalid sender addresses. Confirmation bounces are of no interest to the user and are discarded. To that effect, ASK adds the *X-ASK-Version* header to all confirmation messages it sends. Since most MTAs quote the headers of the original message in their replies, ASK can use this to discard these bounces as invalid senders [12]. This is a workaround for those users who cannot access their MTA configuration to make themselves trusted to the system. Trusted users can configure ASK to send confirmations with an empty "Return Path" which instructs the receiving MTA that no bounces should be sent in case of an error.

3.2 Remote Commands

Remote commands are special strings embedded in the "Subject" header that instruct ASK to perform certain operations. Common tasks are supported, allowing regular maintenance to be performed by users without shell access. Supported operations include:

- Requesting an email containing a brief help screen on the other commands.
- Forcing the delivery of emails in the pending queue.
- Removing emails from the pending queue.
- Adding the sender of an email in the pending queue to the whitelist, ignorelist, or blacklist.
- Editing any of the lists.

ASK offers two flavors of remote commands: *text mode*, which causes an editable template to be generated and delivered to the user's account or *HTML mode* where clickable "mailto" links are used in the body of the email to generate further individual emails containing commands to execute specific actions.

To avoid problems with forged email addresses, ASK never replies back to the sender when dealing with remote commands. Instead, a reply will be delivered to the user's email, set in the configuration file [13]. This reply contains an editable (or clickable if operating in HTML mode) template and some authentication tokens. Upon receipt of this second email, the requested actions will be executed.

As an example, let us suppose that user `user@domain` wants to request a listing of the queue by means of the *ASK PROCESS QUEUE* remote command. The sequence of events would be:

1. The user sends an email from `user@domain` to `user@domain` with "ASK PROCESS QUEUE" in the subject.
2. ASK detects the "ASK PROCESS QUEUE" subject [5] and delivers an email containing the list of queued files (including their MD5 hashes) to `user@domain` [12]. As a security measure, the email is *not* delivered to the sender but rather to the owner's email that was set at configuration time. This guarantees that only the account owner will be able to execute remote commands.
3. The user receives the email containing the "ASK QUEUE REPORT" subject and the list of queued files and hashes. The email is edited by the user with the appropriate commands and sent back to `user@domain`.
4. ASK receives the email, this time with the correct MD5 hashes. The commands in the email are executed [12].

The supported remote commands can be found in Table 1.

3.3 Mailing-List Handling

ASK implements a generic test that is able to match most machine-generated mails including mailing-lists and other challenge-based programs.

A common approach in mailing-lists is to rewrite the sender's address or add a "Reply-To" header pointing to the list distribution address. Without special treatment, confirmation messages would end up being sent to the list's distribution address, causing considerable confusion.

Even though there is no official way of telling whether a message was automatically generated or not, most mailing-list managers today follow some guidelines. ASK will not send a confirmation to a message if at least one of the following criteria is met:

Command	**Description**
ASK HELP	Produces a list of available commands.
ASK PROCESS QUEUE	Process the messages in the queue. Individual messages can be delivered or erased. It is also possible to directly add the sender of a given message to any of the lists (white/ignore/black).
ASK EDIT WHITELIST	Sends back the contents of the whitelist to be edited by the user.
ASK EDIT IGNORELIST	Sends back the contents of the ignorelist to be edited by the user.
ASK EDIT BLACKLIST	Sends back the contents of the blacklist to be edited by the user.

Table 1: Supported Remote Commands

- If the message contains at least one of the following headers: "Mailing-List," "List-Id," "List-Help," or "List-Post."
- If the message contains the "Precedence" header set to "bulk" or "list."
- If the message contains the "Return-Path" header set to null. This header is used to indicate where error messages should be returned. If set to null, no error messages should be returned [8].
- If the username part of the sender's email contain one of the following words: "majordomo," "listserv," "listproc," "netserv," "owner," "bounce," "mmgr," "autoanswer," "noreply," or "nobody." Those are very common names in mailing-lists and auto-responder return addresses.

Messages that match one of the conditions above and are not in the whitelist will still remain in the pending queue, where they can be manipulated with remote commands. No confirmation message will be sent though.

3.4 Sending the Confirmation Message

Sending the confirmation is the last step when processing a message from an unknown sender. Before the confirmation is sent, ASK performs two final steps in order to avoid mail loops with badly configured autoresponders and other auto-reply services:

1. ASK verifies if the current email is already queued by means of the MD5 hash (same hash, same email). This offers a basic degree of protection against services that send exactly the same message multiple times.
2. A circular list is kept with the last *N* addresses used when sending out a confirmation message. If the current sender's email address appears more than *X* times in the list, no confirmation is sent. This guarantees that no more than *X* messages will ever be sent to the same sender in a given period of time. Incoming mails that generate confirmations will push the old ones out of the list, giving the sender a chance for more confirmations after some time. The values of both *X* and *N* are user configurable.

The confirmation message itself is a simple ASCII text email containing brief instructions to the sender and the MD5 hash in the "Subject" field. The confirmation has to be brief and simple or else some senders might just skip it altogether.

A typical confirmation message is presented on Figure 2.

The MD5 hash is generated by concatenating the message text to a user configurable secret, making it impossible for a spammer to get added to the whitelist by crafting a fake confirmation.

The text is easily configurable by the user and more than one language may be present in the confirmation at the same time (normally English and the user's mother language). Ready to use templates are available in English, Spanish, French, German, Brazilian Portuguese, Dutch, Italian, and Finnish.

4 Evaluation

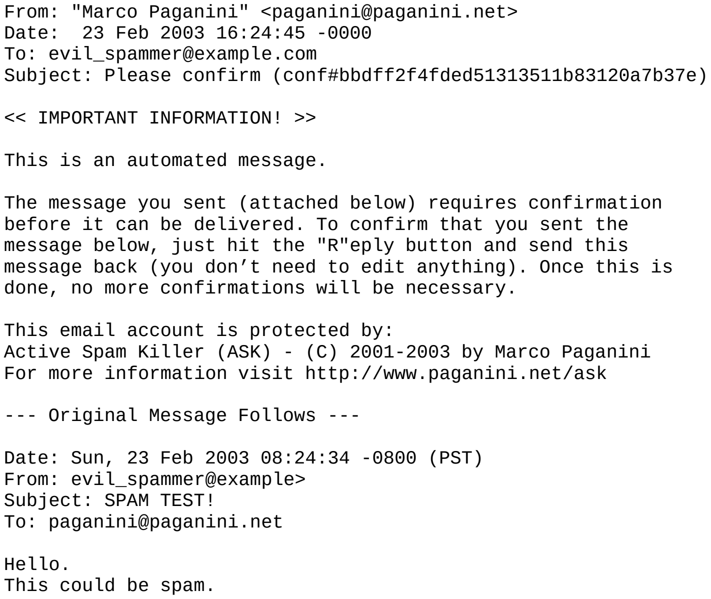
In this section we compare the effectiveness of ASK and other popular anti-spam alternatives.

4.1 Filtering Effectiveness

We selected five sets, from a total of 7000 emails, to evaluate the tools. We used sets *spam1* and *non-spam1*, containing 1000 messages each, to test the content-filtering and RBL tools. We used sets *spam2* and *non-spam2*, containing 2000 messages each, to train the bayesian classifier. We used a larger sample during the training phase as it improved the effectiveness of Bogofilter considerably. We used set *spam3* to test the effectiveness of a challenge-based approach.

Sets spam1, non-spam1, spam2, and non-spam2 were donated by ASK users. We collected set spam3 from SpamArchive [2]. Set spam3 contains no messages previously queued by ASK as those are by definition unconfirmed.

Table 2 compares the effectiveness of different tools when dealing with spam. *nFN* represents the number of false negatives (known spam classified as a valid mail). *nFP* represents the number of false positives (valid mail classified as spam). *%FN* and *%FP* represent the per-



```
From: "Marco Paganini" <paganini@paganini.net>
Date:  23 Feb 2003 16:24:45 -0000
To: evil_spammer@example.com
Subject: Please confirm (conf#bbdff2f4fded51313511b83120a7b37e)

<< IMPORTANT INFORMATION! >>

This is an automated message.

The message you sent (attached below) requires confirmation
before it can be delivered. To confirm that you sent the
message below, just hit the "R"eply button and send this
message back (you don’t need to edit anything). Once this is
done, no more confirmations will be necessary.

This email account is protected by:
Active Spam Killer (ASK) - (C) 2001-2003 by Marco Paganini
For more information visit http://www.paganini.net/ask

--- Original Message Follows ---

Date: Sun, 23 Feb 2003 08:24:34 -0800 (PST)
From: evil_spammer@example>
Subject: SPAM TEST!
To: paganini@paganini.net

Hello.
This could be spam.
```

Figure 2: A Typical Confirmation Message

centage of false negatives and false positives, respectively.

We distributed RBL providers in three groups, named “RBL Group 1,” “RBL Group 2,” and “RBL Group 3.” The first group contains only one RBL provider and represents the minimum protection case. The second group contains a more reasonable mix with three distinct RBL providers. Group 3 represents an extreme case with nine RBL providers. The composition of these groups is listed in Appendix A.

We tested DCC using two different thresholds (number of reports for a message to be considered spam). In the first test, we considered one report enough to mark the message as spam. The second test used a more conservative approach where five reports are needed as opposed to one.

Program	nFN	nFP	%FN	%FP
Bogofilter	22	16	2.2	1.6
DCC (Threshold 1)	799	0	79.9	0.0
DCC (Threshold 5)	925	0	92.5	0.0
RBL Group 1	997	10	99.7	1.0
RBL Group 2	652	16	65.2	1.6
RBL Group 3	420	308	42.0	30.8
SpamAssassin	118	20	11.8	2.0

Table 2: Compared Effectiveness of anti-spam tools

Bogofilter proved to be the most effective content-filtering tool in its category, followed by SpamAssassin. The percentage of false positives is very close for these two tools. DCC had a considerable percentage of false negatives but presents no false positives.

Using one RBL proved to be almost 100% ineffective in our tests, with only three spam emails detected correctly. Raising the number of RBLs checked to three resulted in a 65.2% rate of false negatives, but also raised the percentage of false positives to 1.6%. Group 3 represents an extreme case, with nine RBLs being checked. Even under these circumstances, a high mark of 42% false negatives was verified, with a non-viable mark of 30.8% false positives. Of all solutions tested, RBLs were visibly the worst performers.

4.1.1 ASK Filtering Effectiveness

To test ASK’s effectiveness, we sent 1000 confirmations with a specially crafted envelope address and sender (to catch bounces and replies). We waited for one week for confirmations to return. The results can be seen on Table 3.

Description	Number of messages
Invalid Address (Bounced)	637
No response	259
Malformed emails	86
Responses received	18
Total	**1000**

Table 3: ASK Effectiveness Test

Of the 18 responses received, 14 were from one specific online marketing company and obviously unsolicited. The remaining four came from bargain notification services, apparently sent after the user susbcribed to their services. Only three replies kept the original confirmation code in the message subject (without which, no message delivery takes place), bringing the rate of false negatives to 0.3%. Even if we assume that all spammers could keep the "Subject" header intact, we will still have a rate of false negatives of only 1.8%.

False positives in a challenge-authentication system occur when valid senders do not reply to confirmation messages. Systematic determination of the false positive rate in a challenge-based authentication tool is a difficult task. A confirmation message would have to be re-sent to known valid senders, causing all sorts of problems. Also, some senders who replied to the first confirmation could get confused by the second confirmation request and ignore it completely.

In an effort to understand other users' experiences with spam and how ASK performs in their environments, we submitted a simple survey [1] to the main ASK mailing-list during March of 2003. A total of 32 users responded. The results can be seen in Table 4.

The results indicate that most users have been using ASK from 6 months to one year, with a substantial amount of new users in the last three months.

Most ASK users received 10 to 20 spam messages a day (33%), with a sizable part receiving 20 to 50 (28%) spam messages a day. It is interesting to note that the distribution is fairly even across the categories, indicating that ASK caters to all classes of users when it comes to amount of spam received.

After beginning to use ASK, 61% of the users report that practically no spam is present, and 21% report that only 1 to 5 spam messages are received per month. This brings the total users with a substantial reduction of spam to 82%.

One important aspect of the survey is to verify whether spammers reply to confirmation messages or not. 35% reported no cases of spam delivered due to this reason while 28% reported one or two cases. This brings the total percentage of users with less than three cases of spammers replying to confirmations to 63%. Another 28% reported between two and ten cases since ASK was installed, which still can be considered a good mark considering that most users seem to be using ASK for more than six months.

Another point we tried to clarify is the possibility of valid users not responding to the confirmations (false positives). Most users (42%) reported very few cases of false positives (1 to 2). 31% reported no cases at all. These results combined indicate that 73% of those who responded to the survey had no significant problems with false positives. This seems to indicate that, as a rule, valid senders tend to reply to confirmation messages.

4.2 Compatibility

ASK requires a Unix/Linux compatible operating system. ASK was written in Python and should run with no or few modifications under most Unix variants.

ASK is compatible with most major MTAs available today, including Sendmail, Qmail, Postfix, and Exim. Procmail support is also available. Any MTA capable of delivering to a pipe should work without problems. ASK natively supports mbox and Maildir mailbox formats, with MH planned.

Incoming mails are read on the standard input, making it possible to cascade ASK with other anti-spam solutions like SpamAssassin or a multitude of Bayesian filters and RBLs.

4.3 Performance

To evaluate ASK's performance, we selected a set of 1000 spam messages contributed by ASK users, totaling 14MB of data. On the average, each message has 14KB and 457 lines of text. Text lines have an average length of 31 bytes.

We configured ASK to send a confirmation to each message. ASK took 524 seconds (on a Pentium III/500MHz workstation) to process all messages, bringing the average processing overhead to 0.52 seconds per email received.

During this test, we tried different logging levels (0, 1, 10) and different list sizes (5, 100, 500 lines). No significant variation was detected in the results.

To measure the network overhead, we configured ASK to keep the first 50 lines of each message in the confirmation. This resulted in an average size of 3.3KB per confirmation message, with two languages selected, totaling 3.3MB of extra network traffic. We estimate the overhead of confirmation bounces to be around 2MB, but it can be avoided by configuring the system to send confirmation messages with a null "Return-Path."

Another area of interest is the overhead in processing remote commands. ASK took 30 seconds to process a queue with 1000 messages. This is the most expensive remote command, as every file in the queue has to be opened and investigated. Proper queue maintenance should keep the number of queued files well under 1000, reducing considerably the time for this operation.

The CPU overhead is acceptable for most sites, but something to be considered for high volume servers. The network traffic overhead should be of no concern unless network service is paid by volume. If this is the case, a few bandwidth reduction measures can be taken, such as reducing the number of lines quoted from the original message in the confirmation or limiting confirmation messages to only one language.

Question	Answers	Percentage
Total time using ASK	Less than a month	12%
	1 to 3 months	28%
	3 to 6 months	9%
	6 months to one year	42%
	More than one year	9%
Number of spam messages received per day before ASK	1 to 10	33%
	11 to 20	18%
	20 to 50	28%
	More Than 50	21%
Number of spam messages received per month after ASK	Practically none	61%
	1 to 5	21%
	Between 5 and 10	9%
	More than 10	9%
Number of times a spammer replied to the confirmation	0	35%
	1 to 2	28%
	2 to 10	28%
	10 to 20	6%
	More than 20	3%
Number of times a valid sender failed to reply to a confirmation	0	31%
	1 to 2	42%
	2 to 10	15%
	10 to 20	6%
	More than 20	6%

Table 4: ASK User Survey Results

5 Related Work

In this section we discuss other challenge-based authentications and compare some of their key features to ASK.

5.1 Tagged Message Delivery Agent

Tagged Message Delivery Agent (TMDA) is a spam-reduction solution by Jason R. Mastaler. TMDA is also written in Python and integrates well with most MTAs and MUAs.

One of the key problems in challenge-authentication tools is how to match the confirmation return to the stored message, as very few headers from the original message are kept in the reply. ASK sends the MD5 hash that identifies the message (the cookie) in the "Subject" header, knowing that most MUAs quote the original subject in replies.

TMDA takes another approach to this problem. Instead of relying on the contents of the "Subject" header, the sender's own email is rewritten to contain the cookie in it. This is possible due to the use of *extension addresses*, a configurable MTA feature that allows one account to receive emails with differentiated recipient addresses. Under this scheme, emails going to `foo@bar` and `foo-extension@bar` will both be delivered to user `foo` at the host `bar`. TMDA uses use the "-extension" part to include all sorts of control messages and cookies that need to be present in the reply.

A typical message exchange between a TMDA user called `tmda@domain` and a non-TMDA user called `user@domain` is something as follows:

1. User `user@domain` sends `tmda@domain` a message.
2. TMDA intercepts the message for `tmda@domain` and generates a confirmation to `user@domain`. The confirmation's sender address is rewritten to something like `tmda-confirm-cookie@domain` where the cookie is an alphanumeric sequence that identifies the original message.
3. The user sees the confirmation and replies. The reply is sent to `tmda-confirm-cookie@domain`.
4. TMDA receives the confirmation return and extracts the cookie from the recipient's email address. The message is dequeued and delivered.

TMDA also presents the concept of *tagged messages* (hence its name): outgoing emails may have the sender's address rewritten to contain special tags that bind that email to that recipient (future emails from that recipient will only be accepted on the special, tagged email), dated addresses (emails that are valid only throughout a date range) and keyword addresses, temporary addresses that work indefinitely until manually revoked (for Web-based services).

The utilization of extension addresses adds robustness to the system and the possibility of tagged messages. However, it requires complete control over the MTA configuration files at the ISP level (or an ISP with extension addresses enabled, a rather uncommon feature). Furthermore, some integration issues exist with Sendmail, which does not provide envelope information to programs invoked from the user's `~./forward` file. In these cases, procmail must be configured as the *local mailer* in order to make all the required information available to TMDA.

5.2 Qconfirm

Qconfirm is a challenge-authentication system written in the C Language by Gerrit Pape for Qmail users.

Unlike ASK, Qconfirm is not a single program but rather a group of smaller programs with specific tasks (following Qmail's style). Incoming messages are checked by the `qconfirm-check` program which is also responsible for sending confirmation messages to unknown senders.

Pending messages are held in the qmail queue. `qconfirm-check` will send confirmation messages with an empty envelope sender and the "From" header set to "user-qconfirm-key@host," and create a file named `.qmail-qconfirm-key` under the user's home directory.

Confirmation returns have the recipient set to `user-qconfirm-key@host`. This will trigger the `.qmail-confirm-key` file previously created and invoke `qconfirm-accept` to deliver the queued message and add the sender's address to the `.qconfirm/ok` directory (Qconfirm's whitelist).

Common queue and list maintenance tasks are performed by means of a command line utility called `qconfirm`. This program can also be executed remotely by creating special Qmail control files to invoke `qconfirm-control`. This provides an interface similar to ASK's remote commands.

Qconfirm is lightweight and fast but relies heavily on the infrastructure offered by Qmail. This leaves Qconfirm as an option for Qmail users only.

6 Conclusions

We presented ASK, a challenge-authentication system that authenticates senders before their emails are delivered. In our tests with 1000 spam messages, ASK was able to block 99.7% of all spam messages, meaning that only 3 spam messages got through.

6.1 Future Work

Currently, ASK is not directly integrated with the MUA or the MTA, meaning that it has no knowledge about outgoing emails sent directly by the user. This design decision was taken to allows users without supervisory rights to install and use the program. Unfortunately, it creates certain situations that could be used by spammers like sending emails that appear to be confirmation messages from other ASK users or forging email bounces. MTA and MUA integration will be offered by means of an *SMTP proxy agent* and an *MTA wrapper*. Both approaches offer ASK the opportunity to pre-process outgoing messages before the actual delivery takes place. Extension addresses can be used to rewrite the outgoing envelope address so that invalid confirmation returns and MTA bounces can be correctly tracked.

The problem of emails coming from unknown sources will be addressed with the introduction of two new concepts: *Bounded addresses* and *user confirmation mode*. Bounded addresses create a temporary address that whitelists the first sender who sends an email to it. That sender will be forever tied to that particular email. This is similar in concept to TMDA's "Keyword addresses," with the difference that they become bound to one particular sender after the first use. This creates a "throw-away" email address that can easily be revoked in case of abuse.

User confirmation mode will be available to those who cannot change their MTA configurations or do not desire to make use of extension addresses. Under this mode, confirmation messages are sent to the account owner instead of the sender. This allows the owner to perform a reply and whitelist an email coming from an unknown account. Once the first reply is received, ASK can resume the normal mode of operation.

Other smaller features are also planned, like MH style mailbox support, automatic queue cleanup, and augmented pattern matching for the lists, including full header and body regular expression matching and boolean NOT qualifiers among others.

6.2 Availability

ASK is Open Source Software released under the GNU GPL Software License. The program's home page, including download and documentation links is located at `www.paganini.net/ask`

7 Acknowledgments

We would like to thank Daniel Bastos, Durval Menezes, Charles P. Wright and all others who contributed with their mailboxes, making the effectiveness test possible. We thank Jason Mastaler and Gerrit Pape for their help reviewing the "Related Work" Section. Thanks go to the anonymous Usenix reviewers and specially our shepherd, Erez Zadok, for his guidance and support.

A Realtime Blackhole Lists

Here we list the RBL providers used for the tests performed on Section 4.1.

- Group 1:
  1. Open Relay Database (ORDB),
     `www.ordb.org`
- Group 2:
  1. Spamhaus,
     `www.spamhaus.org`
  2. Blitzed Open Proxy Monitor List,
     `http://opm.blitzed.org`
  3. Relays.Osirusoft.com,
     `http://relays.osirusoft.com`
- Group 3:
  1. Spamhaus,
     `www.spamhaus.org`
  2. Blitzed Open Proxy Monitor List,
     `http://opm.blitzed.org`
  3. Relays.Osirusoft.com,
     `http://relays.osirusoft.com`
  4. Open Relay Database (ORDB),
     `www.ordb.org`
  5. Not Just Another Bogus List,
     `http://dnsbl.njabl.org/`
  6. Extreme Spam Blocking List (xbl),
     `http://xbl.selwerd.cx/`
  7. Fiveten,
     `www.five-ten-sg.com/blackhole.php`
  8. Spamcop Blocking List,
     `http://spamcop.net/bl.shtml`
  9. Distributed Server Boycott List (DSBL),
     `http://dsbl.org/main`

References

[1] Ask user survey. `www.paganini.net/ask/cgi-bin/survey.cgi`, March 2003.

[2] Spamarchive. `www.spamarchive.org`, 2003.

[3] SohoAny Associates. Dynamicmailer. `www.sohoany.com/Dynamicmailer.html`, 2003.

[4] Dan J Bernstein. Qmail. `www.qmail.org`, 2003.

[5] Federal Trade Comission. Email address harvesting: How spammers reap what you sow. `www.ftc.gov/bcp/conline/pubs/alerts/spamalrt.pdf`, November 2002.

[6] The Sendmail Consortium. Sendmail. `www.sendmail.org`, 2003.

[7] The Python Software Foundation. Python. `www.python.org`, 2003.

[8] J. Klensin. Simple mail transfer protocol. Technical Report RFC 2821, AT&T Laboratories, April 2001.

[9] Mail Abuse Prevention System LLC. Mail abuse prevention system rbl. `http://mail-abuse.org/rbl/`, 2003.

[10] Sharon Machlis. Spam's getting more sophisticated. `www.computerworld.com/softwaretopics/software/groupware/story/0,10801,7%7704,00.html`, January 2003.

[11] Jason R. Mastaler. Tagged message delivery agent. `http://tmda.net`, 2003.

[12] University of Cambridge. Exim. `www.exim.org`, 2003.

[13] Gerrit Pape. Qconfirm - request delivery confirmation for mail. `http://smarden.org/qconfirm/`, 2003.

[14] Vipul Ved Prakash. Vipul's razor. `http://razor.sourceforge.net/`, 2003.

[15] Eric S Raymond. Bogofilter. `http://bogofilter.sourceforge.net/`, 2003.

[16] R. Rivest. Simple mail transfer protocol. Technical Report RFC 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.

[17] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[18] Rhyolite Software. Dcc. `www.rhyolite.com/anti-spam/dcc/`, 2003.

[19] Philip Guenther Stephen R. van den Berg. Procmail. `www.procmail.org`, 2003.

[20] The SpamAssassin Development Team. Spamassassin. `www.spamassassin.org`, 2003.

[21] Wietse Venema. Postfix. `www.postfix.org`, 2003.