Payment-Based Email

David A. Turner and Ni Deng
Department of Computer Science
California State University San Bernardino
San Bernardino, CA 92407
(dturner@csusb.edu, ndeng@csci.csusb.edu)

December 31, 2003

Abstract

Spam is a major problem facing email today, and current solutions are ineffective. This paper presents our progress towards defining and implementing a payment-based email delivery system that solves the problem of spam. We present an extension to the email delivery protocol for specifying and collecting payments, explain its security requirements, and illustrate the use of the Lightweight Currency Protocol as an underlying currency system for payment-based email. We discuss the various ways payment requirements can be presented to the end user, and explain our prototyping efforts and how they fit into the larger picture of a peer-to-peer service market based on micro-payments.

Keywords: spam, payment-based email, micro-payments, LCP

1 Introduction

In the United States, spam is plaguing email systems. Many people now receive more spam in their inboxes than legitimate email. Because email has become an important part of everyday communications, spam affects a large number of people. Thus, it is not surprising that the popular press provides continuing coverage of the spam problem.

Solutions to the problem of spam fall into three categories: filtering, legislation and payments. In this introduction, we describe these approaches and their limitations, and we argue that the payment-based approach is the most effective solution to the problem.

Many email service providers use filters to detect and block spam. However, there are a number of problems with this approach that render it ineffective. Failures in spam filters can be classified in two ways: false positives and false negatives. A false positive occurs when the filter labels good email as spam, and a false negative occurs when spam is labeled as good email.

An aggressive filter will have fewer false negatives, and thus will block more spam. However, aggressive filters have a greater number of false positives, which is a serious problem, because users fail to receive potentially important messages. For this reason, spam filters are usually configured so that the probability of a false positive is close to zero, which allows a lot of spam to pass into user inboxes.

Another serious problem with the filter-based approach is that spammers send the same message formatted in different ways in an attempt to bypass the filter. The result is that email systems must process more message traffic, and users may receive multiple copies of the same message.

Over recent years, national and local governments in the United States and other countries have passed laws against spam. However, these laws have not been effective for several reasons. First, there is the problem of defining spam in such a way that distinguishes it from legitimate email. Second, there is the problem of enforcement, which is difficult because email can originate outside of a government's jurisdiction.

The benefit of filters and legislation are that spam can be reduced without requiring any new

user behavior. In Sec. 6, we explain that in certain cases, payment-based solutions add new requirements to users, increasing the *decision costs* associated with email. However, the benefit of a payment model for email is that it can reduce spam more effectively, and reduce the load spam places on network resources. Also, the payment-based approach avoids the problem of defining spam, allowing email to remain an open communication channel.

Some spammers are sending spam on the order of hundreds of millions of email per day. If email delivery charges were as little as $0.01 per email, then 100 million emails would cost one million dollars to deliver, a price in excess of potential profits for most spam. Thus, users can continue to send emails with negligible financial impact, while spammers can no longer afford to operate.

Several other researchers have proposed payment-based schemes. One idea is to require a *proof of work* (POW) payment [8, 2]. These payments are one-time payments that the sending domain sends to the receiving domain; they are not reusable or transferable. After the sending domain connects, the receiving domain presents a mathematical puzzle to the sender that requires a significant amount of time for the sender to solve. After the sender computes the solution to the puzzle, it sends the solution to the receiving domain. While computing the solution to the puzzle consumes significant resources, verifying that it is correct can be done relatively easily. After the receiving domain verifies the response is correct, it accepts delivery of the email. The idea is that spammers will not have the resources to compute the solution to the millions of puzzles that will be presented to them. There are two essential problems with POW schemes when compared to LCP: they waste the resources of senders by requiring a meaningless computation to be performed, and compute time requirements vary too much across the range of CPU speeds.

Fahlman and Wegman of IBM have proposed the use of charity stamps for controlling spam [11]. In this system, senders of email purchase stamps that are required in order for delivery of email, and the proceeds are given to charity. One problem with this system is the requirement of a central authority to decide on the price of stamps, and on which charities earn the proceeds. In contrast, the currency-based system we propose operates in a self-regulated, fully open market with multiple currencies. Power is distributed among participants, rather than concentrated in a central authority. Additionally, these currencies are not restricted to email services, but are redeemable for services outside of email.

Recently, the commercial enterprise *cashette* has started to offer a payment-based email system [1]. When a new user registers with the site, they establish an account that is initially funded with $0.25. During registration, the site displays a code in an image, and requires the human user to enter the code into the registration form. This is a precaution against agents that automatically open up one account after another.

Users with cashette accounts can send mail to other users of cashette accounts. When they do this, the sender transfers a payment in an amount set by the recipient. If the recipient adds the sender to her approved list, then the sender is no longer required to make a payment.

The main limitation to the cashette system is that mail originating from a mail domain other than cashette.com can not be delivered to the recipient unless the recipient has previously added the sender to her approved list. Thus, the cashette system only support mail between users who already know each other, or between cashette users. Cashette can overcome this limitation by adopting the email payment protocol described in this paper, so that users from other mail domains can send email to users in the cashette mail domain.

2 Overview of Payment-Based Email

In payment-based email, the email service provider of the sender makes a payment to the email service provider of the recipient. After payment is received, the sending system is permitted to deliver an email message into the inbox of the recipient. This is the generic procedure used by sender and recipient delivery systems. How costs are passed to end users is a system-specific decision, which we discuss later in this paper.

When the sending system contacts the recipient system, the following events will occur:

1. The recipient system announces that it is a

payment-based service, and enumerates the payment systems it supports.

2. The two systems establish a secure TLS session.

3. The sending system identifies the sender and recipients of an email.

4. The recipient system enumerates the type of payments it will accept in order to deliver this email.

5. The sender makes one of these payments, and then reports to the recipient which payment it made.

6. The recipient system verifies the payment, and replies with OK.

7. The sending system continues with the SMTP protocol in the normal manner to deliver the email.

There are several aspects to this scenario that are worth noting. First, the protocol is an ESMTP extension to the basic SMTP email delivery protocol [9, 4]. This design decision was made in order to make it easier to construct payment-based services using the existing code base, and make it easier for email system implementers to comprehend and adopt the procedure.

Second, the protocol requires the use of TLS [3], so that each end authenticates to each other and transacts over a secure, reliable channel. This is required in order to avoid various attacks motivated by the potential for financial gain.

Third, The system allows for the use of one or more payment systems. This is necessary, because no single electronic payment system has yet emerged as a standard. Additionally, this flexibility allows the system to converge to an effective payment system.

Fourth, when the recipient system enumerates the payments it would accept, it provides the details needed by each system in order to make the payments. The sending system selects one from the list, and makes the payment. Such payment may be done out-of-band using a transfer-based payment protocol such as the Lightweight Currency Protocol (LCP) [15], or in-band using a cheque-based system such as XML-X [16], or a coin-based protocol such as PPay [17]. In this paper, we illustrate payment-based email with payments made using LCP.

3 LCP

Real-world currency does not work well for micro-payment-based systems, because real-world currencies have high transaction costs, and users prefer to avoid accepting the decision requirements for micro-payments. To solve this problem, the Lightweight Currency Protocol (LCP) was designed as a low-cost, low-risk alternative to real-world currencies for payment-enabled systems.

LCP as defined by [15] is a simple, secure protocol that enables nodes with a network connection to issue currencies through public key identifiers. This means that entities (persons or organizations) can issue their own currencies as a medium of exchange without reliance on certificate authorities or other centralized systems.

When an entity – such as a mail service provider – issues currency, the entity imbues value into the currency by making it redeemable for a service that it provides, such as email delivery. However, entities do not need to issue currency in order to operate in the market, because they can rely on the currencies issued by other entities. The motivation to issue a currency is to profit through the collection of transaction fees. Under the LCP, each transaction processed by the currency issuer is subject to a transaction fee.

A currency issuer is responsible to control its supply of currency in the marketplace. When an entity redeems currency for service provided by the issuer, the total amount of currency in circulation decreases. Likewise, when an entity transfers currency to an entity other than the issuer for the purchase of service, the issuer collects a transaction fee, which also decreases the total amount of currency in circulation. To increase the amount of currency in circulation, the issuer purchases services with newly issued currency. It is also possible for a currency issuer to issue new currency in exchange for real-world dollars.

One benefit of LCP is that it is a generic currency not tied to any specific application. Thus, entities can acquire currency through the sale of a service in one market, and spend those currencies in other markets. Potential services include peer-

to-peer media delivery, storage contracts, email delivery, proxy and lookup services, electronic documents, and audio and video media. See [14] for a detailed example of a P2P content distribution application that relies on LCP.

LCP is ideal for payment-based email, because it is easy for email service providers to issue their own currencies to make delivery payments. By transacting in their own currencies, email service providers can reduce the loss of value from transaction fees charged by currency issuers. In fact, they can collect transaction fees on their own currency to offset the fees paid to other issuers.

4 The PAYMENT Command

RFC 1869 defines a framework for extending the SMTP email delivery protocol [9]; all service extensions published by the IETF as standards comply with this framework. The payment-based delivery system advocated in this paper is an SMTP protocol extension that conforms to RFC 1869. We propose to extend SMPT through the use of an additional command called *PAYMENT*.

The use of payments for email delivery raises additional security concerns that go beyond privacy. A plain text protocol will encourage man-in-the-middle attacks designed to steal payments. To remedy this problem, transacting systems should use the STARTTLS service extension [5] to authenticate each end, and establish a secure, reliable communication channel. Justification for this requirement is given in Section 5.

Fig. 1 contains an example exchange between two mail systems, illustrating the main success scenario. In this example, user Alice (with email address *alice@system_a.com*) is sending an email to user Bob (with email address *bob@system_b.com*). In the example, Alice's system is represented with letter C, indicating its role as TCP client. Likewise, Bob's system is represented by the letter S, indicating its role as TCP server. In the example, C obtains the IP address for a mail transfer agent for domain system_b.com by querying DNS. After establishing a TCP connection, S sends a greeting message to C. The client then sends the EHLO command to indicate that it is an ESMTP client. The server replies with the domain name it claims to accept mail for, followed by a list of service extensions it supports, namely, STARTTLS and PAYMENT.

Figure 1: Main Success Scenario

```
S: <wait for connection on TCP port 25>
C: <open connection to server>
S: 220 system_b.net Service ready
C: EHLO system_a.net
S: 250-system_b.net
   250-STARTTLS
   250 PAYMENT
C: STARTTLS
S: 220 Go ahead
<client and server establish a TLS session>
C: EHLO system_a.net
S: 250-system_b.net Service ready
   250 PAYMENT
C: MAIL FROM: alice@system_a.net
S: 250 Address OK
C: RCPT TO: bob@system_b.net
S: 250-LCP/1.0 money.com 0.10 1234
   250-LCP/1.0 system_b.net 0.25 1234
   250 LCP/1.0 system_z.net 0.95 1234
<client makes out-of-band payment>
C: PAYMENT LCP/1.0 money.com 0.10 1234
<server verifies payment>
S: 250 OK
C: DATA
S: 354 Enter mail, end with a single "."
C: Bob,
C: How are you?
C: Sincerely, Alice
C: .
S: 250 OK
C: QUIT
S: 221 Goodbye.
```

The client initiates establishment of a TLS session by sending the STARTTLS command, and then initiating a TLS handshake. After the TLS session is established, the client resends the EHLO command, to which the server replies with a restatement of the service extensions it supports (minus the STARTTLS command).

Next, the client issues the MAIL command, providing the email address of the sending party. The server replies with a status code of 250, indicating success. The client issues the RCPT command,

which identifies the end user to receive the email. The server replies with a status code of 250 to indicate success, and uses the SMTP line continuation mechanism to enumerate the payment options for the client, which is simply a list of payment instructions.

The payment instructions contain enough information for the client to determine how payment should be made, which necessarily includes the payment system to use, the currency (in case the payment system supports multiple currencies), the amount of currency, and possibly other information needed by the payment system. In the example in Fig. 1, the server provides three payment instructions. The components of a payment instruction are separated by one or more space characters (ASCII 32). The first component identifies the payment system. In the example, all payment instructions are for payments using version 1.0 of the Lightweight Currency Protocol (LCP). Each payment instructions contains a currency issuer identifier, the amount of currency required, and a transaction identifier. Because LCP relies on public keys for identifiers, the recipient of payment for all three of these instructions is given by the public key used by the server to establish the TLS session.

The client executes one of the payment instructions, and then issues the PAYMENT command, which identifies which payment option was chosen. In the case of LCP, payment is made out-of-band. However, if a coin-based or cheque-based payment system is used, then the payment token is returned as an argument to the PAYMENT command.

After the server verifies payment, it replies with status code 250 to indicate success. Now, the client issues the DATA command and completes email delivery in the normal manner.

The basic success scenario in Fig. 1 illustrates how an email is sent to a single recipient. If the email is intended for multiple recipients, than it can be handled in two ways. The server allows the client to issue the RCPT command repeatedly to identify each recipient to which it provides a possibly different list of payment options. After the client has identified each recipient of the email, it makes one or more payments that satisfy the server's payment requirements, and then it issues one or more corresponding PAYMENT commands.

The rationale for multiple payments for multiple recipients is that it is possible that the server requires different types payments for different recipients. This may be the result of a system that allows users to specify their own charges, and therefore specify the currencies they are willing to accept. Two recipients of a single email may thus require payment in different currencies.

However, if payments for separate recipients can be aggregated into a single payment, the protocol enables this. In this case, if payments were being made using LCP, the transaction identifiers in the separately enumerated payment instructions would need to be identical. The client would then make a single aggregated payment with the common transaction identifier.

5 Security Issues

Many currency protocols, such as the Lightweight Currency Protocol, are fully secure methods of payments. However, possible security vulnerabilities emerge when payment instructions are transported in plain text over TCP, which is typically the case with email delivery.

Because payment is transferred from the sender to the recipient, the danger is for a man-in-the-middle to impersonate the recipient system. In this case, the attacker provides payment information that results in payments going to the attacker. This situation can arise under two circumstances:

- TLS is not used to establish an authenticated, secure communication channel between sender and receiver.
- The sender does not have – or can not obtain – a reliable copy of the recipient's public key.

We identify three attack strategies based on a man in the middle impersonating the recipient. In the following, we describe these attacks by way examples.

The first attack that we describe is called *simple theft*. Simple theft occurs when the attacker provides payment instructions that cause the sender to transfer currency to the attacker. After the attacker obtains payment, he may allow the email to pass to him, but if he does, he either discards it or saves it for some other purpose. Under simple theft, the recipient never receives the email.

The second attack we describe is called the *spam attack*. The spam attack occurs when an attacker replaces an email with spam. The attacker accomplishes this by acting as a proxy between between sender and receiver, letting commands and replies pass through him unchanged until after DATA command is accepted. After the DATA command is accepted, the attacker accepts the senders message data, but transmits to the recipient a different message. Under the spam attack, the recipient receives payment from the sender, but receives the spam email rather than the sender's email; and the attacker obtains no currency.

The third attack we describe is called *payment skimming*. Under this attack, the attacker delivers the email message, but creates a price differential between the price paid by the sender and the price required by the recipient. The sender makes an inflated payment to the attacker, and the attacker makes the required payment to the recipient, and delivers the email. The result is that the email deliver is accomplished, but the attacker has surreptitiously extracted (or skimmed) a profit from the transaction without being detected.

The attacks described in this paper do not represent an exhaustive list, but they do illustrate the need for an end-to-end security mechanism that protects against man-in-the-middle attacks.

6 User Requirements

We describe three possible approaches to user interfaces that range in complexity from the simplest to the most complex. In all three of these approaches, the underlying payment mechanism executed between mail transfer agents remains the same; only the manner in which the end user is involved in decisions varies.

The simplest approach to the user interface is to completely hide the payment mechanism. We refer to this approach as the fully automated system. The advantage of this approach is that it frees users from making payment decisions. This approach could be considered ideal, because it places no additional requirements on users.

It is easy to see that the fully automated approach would work well for companies or organizations, because the company's system administrators would set company-wide payment policies and acquire currency whenever necessary.

However, a fully automated system would not work well for the free email services, such as yahoo or hotmail. If users of the large free email services were not made accountable for the emails they send, then users would send email from their accounts soley for the purpose of collecting revenues. Because of this likelihood, the free email service providers would need to hold their users accountable for payments needed to deliver their emails. The two approaches described in the following would satisfy this need.

The opposite approach to hiding all payment decisions from the user is to place all payment decisions on the user. We refer to this approach as the full-control approach. In this approach, users set the payment options required for sending emails into their inboxes. Additionally, when sending an email to another user, they are asked to select one of the payment options provided by the recipient system.

It is possible to hold users accountable for the email they send without burdening them with the complexity that comes with multiple currencies. In the system we call the single currency model, users are presented prices in a single currency, and the email system is responsible for translating the multiple currencies being used to the single currency presented to the user. While this approach simplifies the user's decisions, it adds complexity to the email system, and may even require the presence of a currency exchange market.

7 Future Work

We are currently developing a prototype web-based email system that supports a minimal set of SMTP commands (as defined in [4]) in addition to the standard extension STARTTLS and the draft extension PAYMENT described in this paper. In this prototype system, we will explore the intricacies of the single currency model user interface described in the previous section.

Other projects we are working on investigate the possibility of using LCP currencies as a basis for a peer-to-peer resource market. We are also investigating derivative services, such as banks and currency exchanges. These projects will test our conjecture that LCP or another Internet currency pro-

tocol will enable the development of various electronic marketplaces comprised of low-value transactions, such as the payment-based email system presented in this paper.

References

[1] Cachette, inc. Payment-based email service.

[2] A. Back. Hashcash: A denial of service counter-measure. Tech Report, August 2002. http://www.cypherspace.org/hashcash/hashcash.pdf.

[3] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246, January 1999.

[4] J. Klensin Ed. Simple mail transfer protocol. RFC 2821, April 2001.

[5] P. Hoffman. SMTP service extension for secure SMTP over Transport Layer Security. RFC 3207, February 2002.

[6] Paul Hoffman and Dave Crocker. Unsolicited bulk email: Mechanisms for control. Technical report.

[7] Gary Howland. Development of an open and flexible payment system, 1998. http://www.systemics.com/docs/sox/overview.html.

[8] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security*, 1999.

[9] J. Klensin, N. Freed, M. Rose, E. Stefferud, and D. Crocker. SMTP service extensions. RFC 1869, November 1995.

[10] R. Rivest and A. Shamir. Payworld and MicroMint: Two Simple Micropayment Schemes. In *Proceedings of Security Protocols Workshop*, 1997.

[11] A. Robbins. You spam, you pay. *PC Magazine*, April 2003.

[12] Brad Templeton. E-Stamps. http://www.templetons.com/brad/spam/estamps.html.

[13] David A. Turner and Daniel M. Havey. Controlling spam through lightweight currency. In *Proceedings of the Hawaii International Conference on Computer Sciences*, January 2004.

[14] David A. Turner and Keith W. Ross. A lightweight currency paradigm for the P2P resource market, 2003. Submitted.

[15] David A. Turner and Keith W. Ross. The Lightweight Currency Protocol (LCP). IETF Internet draft (a work in progress), September 2003. http://www.ietf.org/internet-drafts/draft-turner-lcp-00.txt.

[16] Erwin van der Koogh and Ian Grigg. The XML-X project. RFC 2246, January 1999. http://xml-x.org/.

[17] B. Yang and H. Garcia-Molina. PPay: Micropayments for Peer-to-Peer Systems. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, October 2003.