Book Title
Book Editors
IOS Press, 2003

On Automatically Detecting Malicious Impostor Emails[1]

Erhan J. Kartaltepe [a], Shouhuai Xu [a]
[a] *Department of Computer Science, University of Texas at San Antonio, {ekartalt,shxu}@cs.utsa.edu*

Abstract. In this paper we explore the problem we call "malicious impostor emails." Compared with the fairly well-known abuses such as spam and email worms, malicious impostor emails could be much more catastrophic because their payloads may directly target at the victim users' cryptographic keys (via whatever means) and their content—except the malicious payload as an attachment—could look perfectly like a legitimate one. As a first step in dealing with malicious impostor emails, we present a partial solution that mitigates their damage without forcing the involvement of the users.

Keywords. malicious emails, malicious worms, automatic detection, email security, public key infrastructure (PKI) robustness

1. Introduction

Emails have become an indispensable part of most people's daily routines. However, emails were not originally designed as a utility in an adversarial environment, which may explain why there have been so many incidents related to the abuse of emails. Two fairly well-known abuses are spam/phishing emails and email worms. Spam emails are often commercially-motivated messages that are sent to innocent users. Although they have wasted a significant amount of human and machine resources, in general spam emails are benign in the sense that they do not carry harmful payloads. The so-called "phishing" emails could do more harm with the participation of an innocent user who may be fooled into trusting, for instance, a bogus web link and entering his username/password which is thus gleaned by the adversary. Recent email worms could do more damage *automatically* (i.e., without the participation of the victim users), because an infected machine could self-duplicate the email worms to all the users in the address book on the computer, and thus the adversary could launch more advanced attacks such as coordinated Distributed Denial of Service (DDoS).

In this paper we envision and characterize (in Section 2) a class of potentially even more catastrophic attacks implemented via what we call "malicious impostor emails" that would directly target the cryptographic keys stored on the victim machines (via whatever means). As a first step in dealing malicious impostor emails, we present a solution that can mitigate their damage. The solution is called MAUDE, which stands for "Mutiserver Authentic User DEtector" (see Section 3). We discuss the integration of

[1]Supported in part by a grant from the UTSA Center for Infrastructure Assurance and Security.

MAUDE into real-life email systems, and present the preliminary performance analysis result in Section 4. We discuss the related works in Section 5 and conclude the paper in Section 6.

2. The Malicious Impostor Emails Problem

The authors have often received emails that falsely claim to be from some colleagues or friends. These emails were easy (for them) to recognize because the non-attachment content was not something the claimed sender should/would write, and the attachment was quite obviously something they should not execute (or "double click"). This is just a simple example of how easy the email headers (e.g., the sender address) can be faked by even a not-so-sophisticated adversary. This was an alert and inspired us to ask the following question:

> What if the email header and non-attachment content of an email looks perfectly like something the claimed sender would write, while the malicious attachment also looks legitimate so that it will likely be "double-clicked" by the recipient?

Informally, we call such emails "malicious impostor emails." Before we give a formal definition, we need to establish a good understanding of the possibility of the attack and its potential damage.

How are "malicious impostor emails" possible? We believe that several causes make malicious impostor emails possible. First, email headers (e.g., the `From` and `Subject` fields) can be faked easily. Since the `From` field typically plays an important role in a recipient's reaction to an email (e.g., the tendency to trust or distrust it), it would make an attack succeed relatively easily if the adversary faked the `From` field appropriately. What could make this strategy effective from an adversary's perspective is that it is now very easy for an adversary to glean information such as "who would send whom emails" and "who would be in one's email address book" so that the appropriately faked emails might be able to bypass some filters (e.g., those that target spam). The incidents we experienced suggest that email addresses that have appeared on our department's webpage have been abused by the adversary to send an email to Bob while claiming it was sent by Alice. As another example of this type of social-engineering based attack, we observe that it is tremendously easy for an adversary to collect the information about "who have co-authored papers with whom" (e.g., via the DBLP website). This has a severe consequence because it is likely that emails claiming to be from some co-author would pass any countermeasures without much difficulty.

Second, it is possible for an adversary to send a recipient a faked email such that non-attachment content looks perfectly meaningful. This is so because email communications are typically in cleartext, and thus an eavesdropper can have access to legitimate emails. As a toy example, suppose Alice just sent Bob a legitimate email with an email content denoted by α. Then an adversary who has eavesdropped on the communication channel could simply send an email to Bob by faking the header and making the content α followed by something like "PS: attached is a recent picture taken in Hawaii." It is likely that Bob will "double-click" the attachment.

What would be the payload of malicious impostor emails? We are concerned with malicious impostor emails that target at critical information such as cryptographic keys.

In spite of the exciting progress in cryptography in the past years (e.g., threshold cryptography following [6]), it is still crucial to ensure that average users'cryptographic keys be appropriately protected. This is so because, in order for cryptography (or a public key infrastructure) to be widely utilized in real life activities, the large population of users should be assured that their cryptographic keys are nearly as secure as, for instance, their biometrics (such as fingerprints). Otherwise, large-scale deployment of cryptography may become useless, if not doing more harm than good. Towards this end, we believe that malicious impostor emails are an effective means that can be exploited to compromise cryptographic keys (e.g., by embedding a Trojan Horse or Cryptovirus [19]), and that systematic investigation should be conducted so that their damage can be mitigated, if not prevented altogether.

Malicious impostor emails vs. spam/phishing emails: Here we highlight some issues regarding the relationship between malicious impostor emails and spam/phishing emails; we will discuss it in more detail in Section 5. As mentioned before, spam messages are unsolicited emails that typically do not exploit the innocent machines' vulnerabilities, as they are motivated by economical benefit and thus the spammers are still rational. Phishing emails could lead to severe consequences, but only after the innocent users are fooled into clicking embedded web links and entering their usernames and passwords. In contrast, malicious impostor emails are launched by an adversary that does not have to be rational. As a consequence, solutions to spam and phishing emails do not necessarily apply to the problem of the malicious impostor emails. As an extreme example, the idea of a whitelist, which may be effective in mitigating spam, is doomed in dealing with malicious impostor emails, because the adversary can perfectly fake the `From` field so that the email looks perfectly like one from the legitimate sender.

Malicious impostor emails vs. email worms: We also highlight the potential difference between email worms and malicious impostor emails; more details are discussed in Section 5. Email worms often launch destructive attacks such as Distributed Denial of Service (DDoS), and typically try to infect as many machines as soon as possible. On the other hand, malicious impostor emails may only be interested in breaking certain victim machines pre-selected by the adversary. To this end, malicious impostor emails may corrupt some users' computers as their "step stones" and may not actually do any harm until they have reached the targeted victims, and could adopt even more advanced and crafty strategies to bypass or compromise the deployed countermeasures (e.g., they may spread slowly in a steganographic way).

Summary: Suppose an email consists of three parts: a *header* (including the `From` field), a *non-attachment content* (i.e., the email body that is not part of the attachment), and an *attachment*. For a given email, denoted by EMAIL, let $\mathsf{sender}(\text{EMAIL})$ be the sender id (i.e., the value in the `From` field of the header). For each email user U, let $\mathsf{whitelist}_U$ be the whitelist of email addresses that will be accepted by U, Filter_U be a filter that takes as input an email, analyzes its non-attachment content, and outputs a decision on whether it thinks the email is suspicious (e.g., a spam or phishing email), $\mathsf{wormScanner}_U$ be a worm/virus scanner that takes as input an email, analyzes its attachment, and outputs a decision on whether the attachment is suspicious. We assume that Filter_U is perfect, meaning that any spam or phishing email will be detected. However, $\mathsf{wormScanner}_U$ is not perfect, meaning that it can only detect the malicious attachments that possess known

signatures; they have limited success in dealing with polymorphism worms/virus and cannot deal with unknown (zero-day) worms.

Definition 1 (malicious impostor email) *A malicious impostor email is an email, denoted by* EMAIL*, sent to a recipient* U *with* $(\mathsf{whitelist}_U, \mathsf{Filter}_U, \mathsf{wormScanner}_U)$ *such that*

1. $\Pr[\mathsf{sender}(\text{EMAIL}) \in \mathsf{whitelist}_U] = 1$*, meaning that the email can always perfectly fake the email header information including the sender's email address and perhaps the IP addresses incorporated in the email header.*
2. $\Pr[\mathsf{Filter}_U(\text{EMAIL})$ *outputs "suspicious"*$] = 0$*, meaning that the non-attachment content is perfect and cannot even be detected by a human being.*
3. $\Pr[\mathsf{wormScanner}_U(\text{EMAIL})$ *outputs "suspicious"*$] = \theta$ *for some* $0 \leq \theta < 1$.

The ultimate goal is to allow automatic detection of *all* malicious impostor emails so that they can be appropriately processed, although in what follows we are only able to present a partial solution called MAUDE.

3. MAUDE: Multiserver Authentic User DEtection

3.1. Model

We consider a system consisting of multiple email servers. Each server has a set of legitimate users that can utilize its service. We call the *outgoing* email server the *physical* machine which actually initiated the sending of a sender's email, and the *incoming* email server the *physical* machine which delivers a message to the receipient. Typically, an email takes a path consisting of at least one email server. That is, a path starts with the sender's physical email server and ends at the recipient's physical email server; the servers on the path between may be called relays). Then, the *first* server on the path is the outgoing server, and the *last* server on the path is the incoming server. Therefore, we are only interested in the outgoing and incoming servers, and will ignore the relays.

We assume that (a subset of) the servers have some established trust, which may be fulfilled by each pair of email servers sharing a common secret (i.e., a cryptographic key). This would be feasible if the system of interest is under the same administrative domain (e.g., a campus network or the network of a state university system); We will discuss more on this assumption below.

We consider a probabilistic polynomial-time adversary that intends to send *malicious impostor emails* with the intent that the recipients will "double-click" the attachment.

The basic idea underlying MAUDE is very simple. When a recipient's email server (i.e., the incoming email server) receives an email with an attachment that claims to be from some server (i.e., the alleged outgoing email server), the incoming email server will contact the alleged outgoing email server to verify that it sent that specific email. If so, the email is processed as in the original email system; otherwise, the email is suspicious and appropriate action is taken (e.g., thrown into some special folder with an explicit alert or even deleted).

3.2. Detailed Description

At an abstracted level, MAUDE consists of several (distributed) algorithms. Before we present the algorithms, we need to introduce some notations. Let $sender$ be the sender of an email and $recipient$ be the recipient of an email, both in the form of the string "$user@domain$", where $user$ is the username and $domain$ is the hostname concatenated with "." and the top-level domain name. Let $subject$ be the subject of the email and $body$ be the email content itself, containing both the text message (if present) and the attachment.

The above values can be obtained from the real-life email system, namely Sendmail in our case study, via the following algorithm getMaudeMessage. This algorithm's design is based on the observation that Sendmail prepares the SMTP payload as a list of headers followed by the email message. A header has two members $field$ and $value$, where $field$ holds the type of header and $value$ holds its content. Let $Headers$ be the set of headers in the email and $email$ be the email message itself. The following algorithm takes as input $(Headers, email)$, which is provided by Sendmail, and returns the desired $(sender, recipient, subject, body)$.

getMaudeMessage($Headers, email$)
  FOREACH $header \in Headers$
    IF ($header.field =$"From")
      $sender \leftarrow header.value$
    IF ($header.field =$"To")
      $recipient \leftarrow header.value$
    IF ($header.field =$"Subject")
      $subject \leftarrow header.value$
  $body \leftarrow email$
  Return $(sender, recipient, subject, body)$

Let each outgoing email server maintain a database DB for the emails it has sent. An entry of DB is of the format $(sender, recipient, \mathsf{H}(sender, recipient, subject, body))$. Each outgoing or incoming email server maintains a key tables of entry format $(peer, key)$, where $peer$ is another server which shares its common secret key. Each server also runs a MAUDE process which will operate over the DB.

In order fulfill its purpose, MAUDE will treat the following building-block algorithms as black-box in nature, but whose functionalities are well-understood:

- host($emailAdd$) returns the hostname of the email address.
- domain($emailAdd$) returns the domain name of the email address.
- getKey($machine$) returns the key shared between the server that is running this algorithm and the remote sender/recipient server called $machine$.
- oldSend($sender, recipient, subject, body$) is the original email sending functionality (e.g., the one implemented by Sendmail).
- oldReceive($sender, recipient, subject, body$) is the original email receiving functionality (e.g., the one implemented by Sendmail).
- impostor($sender, recipient, subject, body$) handles the impostor emails according to a policy (e.g., placing them in a special folder or deleting them).
- H is a collision-resistant hash function.
- HMAC is a secure message authentication code such as HMAC-SHA1 [4].

MAUDE consists of the following algorithms.

- addMaude$(sender, recipient, \alpha)$ updates DB with a new entry. This algorithm is initiated by the outgoing server and executed by its MAUDE.
- queryMaude1$(sender, recipient, subject, body)$ determines whether the entry $(sender, recipient, subject, body)$ appears in DB. This algorithm is initiated by the incoming server and executed by its MAUDE.
- queryMaude2$(sender, recipient, subject, body)$ asks the local MAUDE to check if the email was sent by the claimed server. This algorithm is initiated by the incoming server and executed by its MAUDE.
- queryPeerMaude$(sender, recipient, subject, body, \alpha, r, \beta)$ is an interactive algorithm between an incoming MAUDE and an outgoing MAUDE. It is initiated by the incoming server's MAUDE and executed by the remote outgoing server's MAUDE.
- newSend$(sender, \{recipient_i\}_{1\leq i\leq n}, subject, body)$ is the extended email-sending algorithm run by the outgoing server.
- newReceive$(sender, recipient, subject, body)$ is the extended email-receiving algorithm run by the incoming email server.

Their precise functionalities are described below.

addMaude$(sender, recipient, \alpha)$
  DB $\leftarrow$ DB $\cup \{(sender, recipient, \alpha)\}$

queryMaude1$(sender, recipient, \alpha)$
  IF $((sender, recipient, \alpha) \in$ DB) THEN
    DB $\leftarrow$ DB $\setminus \{(sender, recipient, \alpha)\}$
    Return TRUE
  ELSE Return FALSE

queryMaude2$(sender, recipient, \alpha)$
  $k \leftarrow$ getKey(host$(sender)$)
  select a random r
  $\beta \leftarrow$ HMAC$(k; sender, recipient, \alpha, r, 0)$
  $\gamma \leftarrow$ queryPeerMaude$(sender, recipient, \alpha, r, \beta)$
  IF $(\gamma =$ HMAC$(k; sender, recipient, \alpha, r, 1))$ THEN
    Return TRUE
  ELSE Return FALSE

queryPeerMaude$(sender, recipient, \alpha, r, \beta)$
  IF $((sender, recipient, \alpha) \notin$ DB) OR (getKey(host$(recipient)) = \perp$)THEN
    Return $\perp$
  ELSE
    $k \leftarrow$ getKey(host$(recipient)$)
    IF (HMAC$(k; sender, recipient, \alpha, r, 0) = \beta$) THEN
      DB $\leftarrow$ DB $\setminus \{(sender, recipient, \alpha)\}$
      Return HMAC$(k; sender, recipient, \alpha, r, 1)$
    Return $\perp$

```
newSend(sender, {recipient_i}_{1≤i≤n}, subject, body)
    FOR i = 1 to n
        IF (getKey(host(recipient_i)) ≠⊥) THEN
            addMaude(sender, recipient_i, H(sender, recipient_i, subject, body))
    oldSend(sender, {recipient_i}_{1≤i≤n}, subject, body)

newReceive(sender, recipient, subject, body)
    IF (host(sender) = host(recipient)) THEN
        IF (queryMaude1(sender, recipient, H(sender, recipient, subject, body)))
            THEN oldReceive(sender, recipient, subject, body)
            ELSE impostor(sender, recipient, subject, body)
    ELSE  IF (getKey(host(sender)) ≠⊥) THEN
                IF (queryMaude2(sender, recipient, H(sender, recipient, subject, body))
                    THEN oldReceive(sender, recipient, subject, body)
                    ELSE impostor(sender, recipient, subject, body)
    ELSE oldReceive(sender, recipient, subject, body)
```

3.3. Analysis

Security of MAUDE against a probabilistic polynomial-time adversary is clear (based on the security of the message authentication codes).

The extra computational overhead imposed on the servers is small (i.e., computing some message authentication tags). The size of the database DB grows proportionally to the number of email-sending requests, and decreases as the emails have been confirmed. Thus, on average the size of DB is relative to the difference between the email-sending requests and the email-confirmation requests.

3.4. Discussion on Assumptions and Design Rationale

We observe that MAUDE's utility applies when there is some well-established trust between the email servers. This is reasonable for email servers that are under the same administrative jurisdiction, or emails servers that are administered by some collaborative partners (e.g., all the schools of a state university system). This already suffices to solve the malicious impostor email problem in the setting that the "claimed" senders (i.e., the senders listed in the headers of the malicious impostor emails) are indeed affiliated with the email servers that are administered by the collaborative partners.

What if the servers do not have the above-mentioned initial trust relationship? An immediate approach is to assume the existence of a (potentially small-scale) PKI that certifies the public keys of the email servers in one way or another. Based on this, there are numerous cryptographic protocols that suffice to establish initial trust and common keys between the servers. If it is possible to establish such a "meta" PKI, we can utilize it to achieve a robust large-scale PKI so that the large population of average users' (rather than the small-population of servers') keys can be well-protected from malicious impostor emails.

But if we assume the existence of the initial trust, why don't we simply let each outgoing server associate with every email a digital signature or message authentication tag? The reasons against this are threefold. First, we believe that an ultimate solution should not be solely based on the existence of such an established, though small-scale, trust

structure. Under certain circumstances, it would still be useful even if we can achieve *best effort* security in the following sense: An incoming server still contacts an outgoing email server to check if a claimed email was indeed sent by it, even if this communication is *not* protected by a cryptographic means (due to the absence of a common key). In this case, the communication initiated by the incoming email server could become another factor of authentication (or the adversary has to use some real email server with real domain name and IP address, which would help forensics investigation, or the adversary has to hijack many TCP sessions, which may significantly limits its effectiveness).

Second, we are investigating methods that can help a pair of servers that can establish a certain degree of trust without relying on any trusted third parties. Of course, the established trust is weaker than the one established via some trusted third parties; however, this is still promising because the trust may now be exclusive between the two servers.

Third, the way the real-life email systems operate is quite complicated. For example, it is rather difficult for an outgoing email server to figure out which *physical* machine is the real destination machine; on the other hand, it is relatively easy for the real destination machine to determine which *physical* machine was the sender (initiator) of the email. Therefore, the design makes it possible for the destination server to *only* contact the initial sending server; this avoids involving the numerous email relays that are indicated in the email headers (which could be quite long), and makes the scheme practical.

4. Integrating MAUDE into Email Systems: Experiments, Performance, and Effectiveness

4.1. Methodology

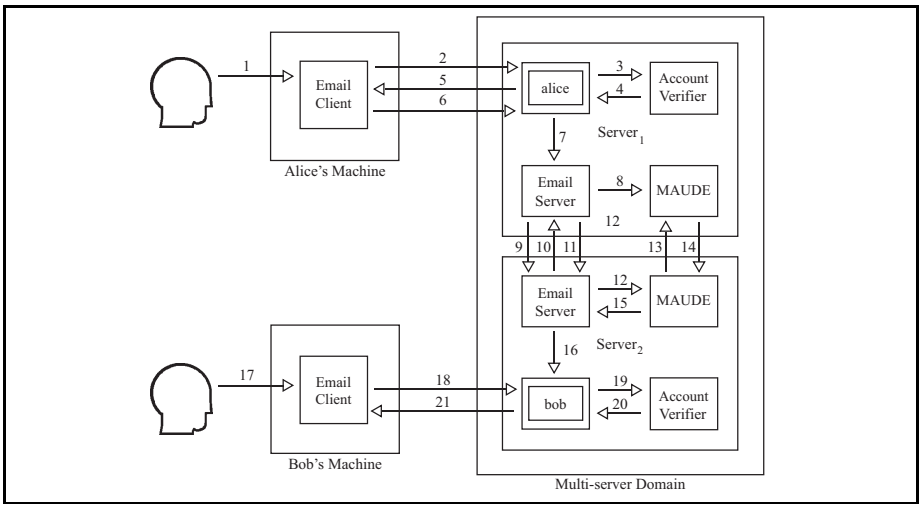

Figure 1. A low-level view of MAUDE

In order to test the effectiveness of MAUDE, we implemented it into Sendmail, a widely deployed email system. To avoid a modification of the SMTP protocol or delay the communication between the two email servers, the modified Sendmail server waits until the SMTP protocol has completed before extracting the headers and constructing the message it sends to its local MAUDE. In other words, we devise a scheme to modify Sendmail while encapsulating the SMTP protocol from MAUDE. As a result, an email system with MAUDE operates as follows (see also Figure 1).

1. Alice logs into her local email client.
2. After composing a message, Alice's email client logs in to the server.
3. Alice's password is sent to an account verifier to determine her authenticity.
4. The account verifier grants Alice access to her account.
5. The server contacts the email client to send the message.
6. Alice's email client sends the message to the outbox in her account.
7. The system delivers the message to the email server for delivery.
8. Alice's email server executes getMaudeMessage$(Headers, email)$ which returns $(sender, recipient, subject, body)$. Immediately after, Alice's email server executes addMaude$(sender, recipient,$ H$(sender, recipient, subject, body))$.
9. Alice's email server server contacts Bob's email server to transmit the message.
10. Bob's email server accepts the message, recording the other server's hostname and IP address.
11. Alice's email server delivers the message.
12. Bob's email server executes getMaudeMessage$(Headers, email)$ which returns $(sender, recipient, subject, body)$. Immediately after, Bob's email server executes queryMaude2$(sender, repicient,$ H$(sender, recipient, subject, body))$.
13. Bob's MAUDE activates queryPeerMaude$(sender, recipient, \alpha, r, \beta)$.
14. Alice's MAUDE executes queryPeerMaude$(sender, recipient, \alpha, r, \beta)$ and returns γ.
15. Bob's MAUDE returns TRUE (supposing queryPeerMaude returns the correct value).
16. The email server routes the message to Bob's inbox.
17. Bob logs into his local email client.
18. To retrieve his email, Bob's email client logs in to the server.
19. Bob's password is sent to an account verifier to determine his authenticity.
20. The account verifier grants Bob access to his account.
21. The server delivers the email in Bob's inbox to the email client.

4.2. Metrics

The metrics we are interested in are the following:

- The delay incurred by MAUDE on the outgoing email server. The delay is defined as the time period between the point that Sendmail has accepted the message for delivery and is about to contact its local MAUDE and the point that the local MAUDE has updated the database.
- The delay incurred by MAUDE on the incoming email server. The delay is defined to be the time period between the point that the SMTP protocol has begun and the point that the email is thrown into the email box (or discarded).
- The effectiveness and accuracy of MAUDE's decisions.

4.3. Experimental System Settings

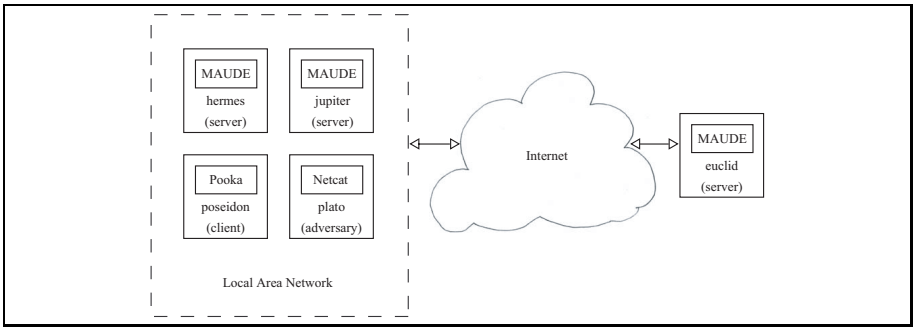

Figure 2. Integrating MAUDE into real-life email systems

The system environment is depicted in Figure 2. The email servers are within a university campus network, and the email clients are both within and outside the campus network. The email servers are called `hermes` and `jupiter`, respectively. There are two email client machines: `poseidon` acted as a friendly external computer within the LAN with authorized access to `hermes` through an email client, and `plato` was an adversary client machine within the campus network. The purpose of `plato` is to send faked emails but make them look as though they were sent by legitimate users through a tool like Netcat. A fifth machine, `euclid`, tested the performance of MAUDE on a non-dedicated internet connection. The three servers, `hermes`, `jupiter`, and `euclid` recognized each other by sharing some pair-wise keys. Figure 3 reviews the concrete configurations of the machines and networks.

Machine	Processor	Internet Connection	Relavant Software
`hermes`	2.26 GHz P4	Gigabit LAN	Sendmail 8.12.9, MAUDE
`jupiter`	2.80 GHz P4	Gigabit LAN	Sendmail 8.12.9, MAUDE
`poseidon`	2.26 GHz P4	Gigabit LAN	Pooka Email Client
`plato`	2.80 GHz P4	Gigabit LAN	Netcat, mutt
`euclid`	2.40 GHz P4	100 Megabit Cable	Sendmail 8.12.9, MAUDE

Figure 3. System Settings

MAUDE was written in Java 1.5.0 and ran on the Java Virtual Machine. For processing incoming and outgoing emails, Java's crypto library was used to compute any hash value or HMAC it needed, in both cases using the SHA1 algorithm. For testing, a program simulating Sendmail on each server sent valid messages to its local MAUDE. This program, implemented in C, used Peter Gutmann's cryptlib 3.1 library to compute the SHA1 hash value for the tuples it sent to MAUDE. This simulator measured the extra delay time MAUDE added to the delivery of an email.

4.4. Performance

To examine the delay incurred by MAUDE on the sender's email server, time was marked before and after each transmission over 10000 requests. We repeated this test ten times and took the average over the runs. Figure 4 shows the trends over the 10000 requests.

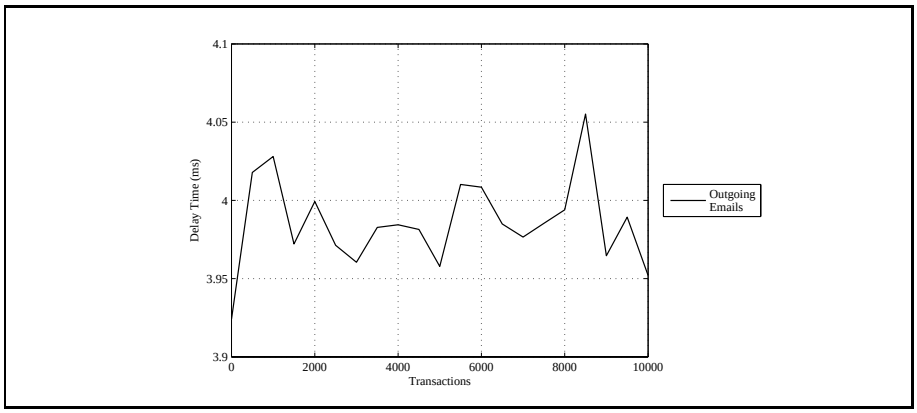

Figure 4. Outgoing Emails over 10000 trials

For incoming emails, we ran three tests of 10000 concurrent requests at the rates of 100, 1000, and 5000 requests per minute, respectively. These stress tests were implemented to show how MAUDE operates with a high-volume incoming email server. For each of the three experiments, we obtain the corresponding delay time by averaging 10 independent runs (Note that a single run of the simulation at the rate of 100 requests per minute requires 100 minutes for all 10000 requests). In plotting the data, we fit the curves by taking a moving average of 500 requests for a clear representation of each trend. We plotted the delay time for the three stress tests for the LAN and WAN MAUDE in Figure 5.

In the LAN experiments, the Sendmail simulator on `jupiter` delivered a valid message to the local MAUDE. In turn, the local MAUDE contacted `hermes` with a valid message to determine if `hermes` sent the message. We recorded the time it took for `hermes` to respond to each request from `jupiter` from the moment Sendmail constructed the message for MAUDE until after it processed MAUDE's return message. In the WAN experiments, the simulator on `euclid` operated in the same fashion.

Delay times taken as an average are summarized in Figure 6. The time for Sendmail to send an email averaged under 177 milliseconds. At the rate of 100, 1000, and 5000 requests per minute, the delay time increase due to a LAN MAUDE is increases by 23.8%, 40.6%, and 81.1%, respectively. However, even via a WAN, MAUDE's delay time is less than the original architecture's processing capability.

4.5. Effectiveness

Using the Pooka email client [15] on `poseidon`, the email client showed no noticeable delay when MAUDE was activated versus when it was absent. Since MAUDE's effects

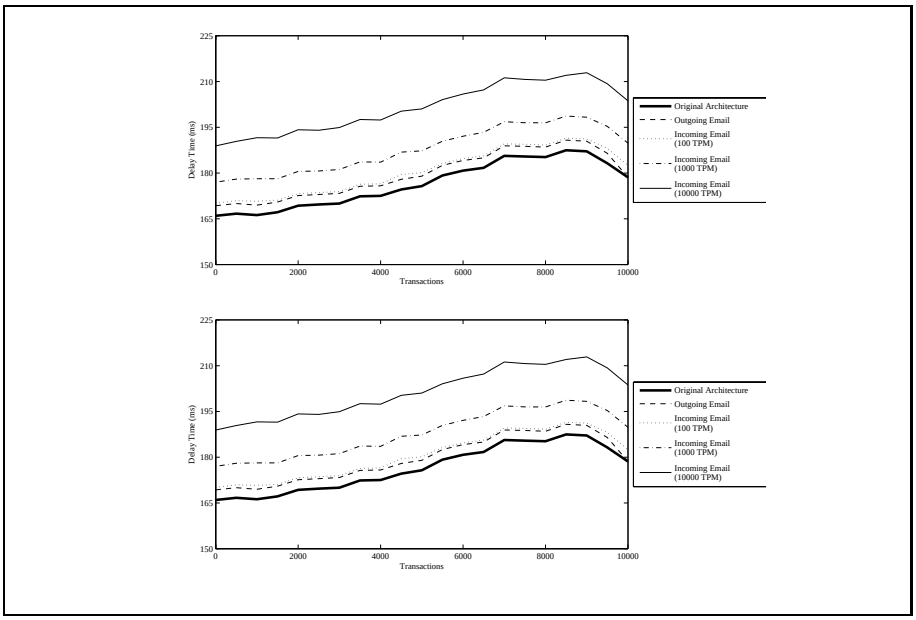

Figure 5. Incoming emails over 10000 trials via a LAN MAUDE (top) and WAN MAUDE (bottom)

Transactions Per Minute	Original Architecture	Incoming Email (LAN Network)	Incoming Email (WAN Network)
100	176.406	217.307	224.061
1000	176.406	248.619	256.440
5000	176.406	319.491	332.351

Figure 6. Average delay times for each experiment (milliseconds)

are encapsulated away from any email agent implementation, no special setup to the client was necessary.

When using Netcat [14] on `plato` to send bogus email messages to `hermes`, every email was checked against the supposed sender's MAUDE, and each came back as fraudulent. This was the case when using the more user-friendly mutt email client [13] to send impostor emails as well. Moreover, every email sent by a legitimate user was not flagged as a possible impostor. Thus, we note that no false positive or false negative is possible.

5. Related Work

On the relationship between malicious impostor emails and PKI. In order for a public key infrastructure (PKI) to achieve its fully-expected utility, it is necessary to ensure that the cryptographic keys are appropriately protected. Advanced cryptographic mechanisms such as threshold cryptography [6] have been invented to protect critical cryptographic functionalities (e.g., the signing function of a certificate authority) so that the damage

incurred by sophisticated attackers could be mitigated. We observe, however, that the protection of average users' cryptographic keys would be an equally important factor in deploying a PKI. Unlike in the setting of the servers, the protection of individual cryptographic keys could be far more challenging (given that smartcards are not widely deployed), particularly because the individual users' machines are always used to fulfill their routing tasks without being administered by professionals (servers are relatively well-protected by professionals who may keep patching the relevant software programs). Moreover, there is a wide spectrum of ways to compromise an individual's machine (and thus the cryptographic keys). One such method is the above described "malicious impostor emails" whose malicious payloads can compromise one's private keys (e.g., [19]). This explains why dealing with malicious impostor emails is an essential part of protecting a PKI. On the other hand, it would be ideal if MAUDE can get support from a small-scale PKI (e.g., one for the email servers of a relatively small population), then a large-scale PKI (involving a large population of average users) built on it can be better protected—this is the case of MAUDE.

More on malicious impostor email vs. email worms. Current worms (e.g., [16,21,18, 11,17,22]) are typically high-speed because they want to infect as many machines *as fast as possible*. Since "high-speed spreading" does not necessarily apply to malicious impostor emails, they need new solutions.

More on malicious impostor emails vs. spam/phishing. Spam emails are quite different from malicious impostor ones, and therefore solutions to countering the former might not be effective at combating the latter. As a consequence, an approach founded on economic incentives that could be effective against spam would not necessarily translate to resolving the problem of malicious imposter emails. Goodman [10] analyzed the problem of preventing outgoing spam while assuming that the adversary is *rational*, but this solution would not work against malicious emails because the adversary launching them does not have to be rational.

Machine learning or data mining based filters [9] for detecting or blocking potential spam are not necessarily effective in detecting or blocking malicious impostor emails on their own. This is so because the email content, besides the offending attachment, could be an otherwise perfectly legitimate email.

The puzzle approach, based on computational puzzles [8] or their recent variants called "moderately hard, memory-bound functions" [2,7], could be effective in dealing with spam. Similarly, the related virtual stamps or other payment schemes (e.g., [1,12]) could be successful as well against spam, provided that spammers have only a reasonable budget (recall that spammers are typically motivated by economic incentives). However, they do not solve the problem of malicious impostor emails as they involve the end (or average) users. Moreover, introducing cryptography-based stamps indeed bring in new problems that the end users need to protect their cryptographic keys appropriately. We need a solution that is transparent to end users.

On the relationship with other loosely related works. There are some other works that are loosely related to ours. For example, certified emails [3,5,20] deal with *fair exchange* between a recipient's receipt and a sender's email. Although there may be some resemblance between the certified email protocols and our approach to dealing with impostor emails, the two problems are indeed fundamentally different. First, the ultimate goal of certified emails is to fulfill fair exchange transactions between two *end*

users, not necessarily between the corresponding incoming and outgoing email servers (unless one would like to make the unrealistic assumption that the users fully trust, and thus would even give their private keys to, the email servers). In contrast, our approach to dealing with impostor emails is contained to the interactions between the email servers. Indeed, such an ideal solution, as what we have proposed, should be transparent to the end users. Second, in contrast to the case of certified emails, the sender of malicious impostor emails (i.e., the adversary) never has the intent to conduct a fair exchange with a (victim) recipient. Indeed, the adversary always tries to hide itself so that it will not be held accountable.

6. Conclusion and Future Work

We introduced the problem we call "malicious impostor emails"—emails that can perfectly fake email headers and possess non-attachment content that can flawlessly mimic legitimate ones. We explored a partial solution to this problem as a first step toward countering this powerful attack. We are continuing the investigation in the following directions:

- How can we remove or weaken the reliance on the initial trust between the email servers? This is particularly relevant when we consider the Internet as a single system.
- Our solution is effective provided that every involved email server has MAUDE installed. The hope is that MAUDE can be employed incrementally. So can we establish an analytical model to understand the effectiveness of the incremental deployment of MAUDE?

Acknowledgements

We are grateful to Allen Petersen for the use of the Pooka email client source code and insight into the program. We also thank Carlos Cardenas for his suggestions, and Allen Fouty and Hugh Maynard for their helpful comments and feedback. Finally, we thank the anonymous reviewers for their useful comments.

References

[1] M. Abadi, A. Birrell, M. Burrows, F. Dabek, and T. Wobber. Bankable postage for network services. In *Advances in Computing Science—ASIAN 2003, Programming Languages and Distributed Computation, 8th Asian Computing Science Conference*, pages 72–90.

[2] M. Abadi, M. Burrows, M. Manasse, and E. Wobber. Moderately hard, memory-bound functions. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, pages 25–39.

[3] A. Bahreman and J. D. Tygar. Certified electronic mail. In *Proceedings of the 1994 Network and Distributed Systems Security Conference*, February 1994, pages 3–19.

[4] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Proceedings of Crypto'96*.

[5] R. Deng, L. Gong, A. Lazar and W. Wang. Practical Protocols For Certified Electronic Mail. *Journal of Network and System Management*, vol. 4, no. 3, 1996.
[6] Y. Desmedt and Y. Frankel. Threshold Cryptosystems.*CRYPTO'89*.
[7] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In D. Boneh, editor, *Proc. CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 426–444. Springer-Verlag, 2002.
[8] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Proc. CRYPTO 92*, pages 139–147. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.
[9] J. Goodman and R. Rounthwaite. Smartproof. manuscript, 2004.
[10] J. Goodman and R. Rounthwaite. Stopping outgoing spam. In *Proceedings of ACM E-Commerce 2004*, 2004.
[11] A. Gupta and R. Sekar. An Approach for Detecting Self-Propagating Email Using Anonmly Detection. *Proceedings of RAID'03*.
[12] A. Herzberg. Controlling spam by secure internet content selection. In *Proceedings of the 4th Conference on Security in Communication Networks, Lecture Notes in Computer Science*, 2004.
[13] Michael Elkins The Mutt E-Mail Client `http://www.mutt.org/`
[14] Giovanni Giacobbi The GNU Netcat Project `http://netcat.sourceforge.net`
[15] A. Petersen Pooka: A Java Email Client. `http://suberic.net/pooka`
[16] S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. *Proceedings of Usenix Security 2002*.
[17] C. Wong, S. Bielski, J. McCune, and C. Wang. A Study of Mass-Mailing Worms. *Proceedings of ACM Worm 2004*.
[18] J. Wu, S. Vangala, L. Gao, and K. Kwiat. An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques. *Proceedings of NDSS 2004.*
[19] A. Young and M. Yung. Cryptovirology: Extortion Based Security Threats and Countermeasures. *IEEE Symposium on Security and Privacy'96*.
[20] J. Zhou and D. Gollmann. Certified electronic mail. In *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS 96)*, pages 160–171.
[21] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and Early Warning for Internet Worms. *Proceedings of ACM CCS'03*.
[22] C. Zou, D. Towsley, and W. Gong. Email Worm Modeling and Defense. *Proceedings of International Conference on Computer Communications and Networks (ICCCN'04)*.