Blocking Foxy Phishing Emails with Historical Information

Yongdong Wu, Zhigang Zhao, Ying Qiu, and Feng Bao
Institute for Infocomm Research, Singapore
{wydong,zzhao,qiuying, baofeng}@i2r.a-star.edu.sg

Abstract—Unlike most of the spams targeting for advertisements only, phishing spams try to cheat the email recipients with bogus sender addresses so as to obtain confidential information of the recipients. This paper presents a Sender Authentication Protocol (SAP) which aims to filter out this kind of crafty spoofing emails. To this end, SAP challenges the claimed-sender with the historical emails so as to verify the authenticity of the sender. As it does not change the email protocol, and is able to be embedded into the off-the-shelf email software such as Microsoft Outlook. SAP is not only easy to be deployed, but also fully compatible with other anti-spam technologies. To illustrate its feasibility, we develop a SAP add-in for enhancing Microsoft Outlook. The SAP add-in will be started automatically as long as Outlook is started. As the enhanced Outlook has the same user interface as the original one, the add-in is transparent and friendly to the users.

I. INTRODUCTION

Electronic mail, or email for short, has been one of major applications of Internet since the standard SMTP (Simple Message Transfer Protocol) [1] was published in 1982. Because it costs almost nothing to deliver an email, email systems have been heavily misused in the 21st century. For instance, it has been estimated that 3.8 billion spam messages sent to Hotmail [2] daily, over 120 billion spam messages in the world each day [3], approximately 86% of e-mail traffic being spam in 2006[4], or nearly $22 billion for deleting junk email per year [5]. Even worse, there were at least 47,324 phishing attacks in the first half of 2008 [6], [7] albeit most of the spams are merely for advertisement and harmless[8]. Those phishing emails tried to steal the confidential information of the email recipients by hiding the email origins.

Historically, email or SMTP protocol was used to deliver messages among researchers who were trustworthy, hence it was unnecessary to authenticate the sender addresses in the delivered emails at that time. Nowadays, anyone is able to have an arbitrary number of email accounts and send emails freely, the non-authenticity of source address opens the door for massively distributing spoofing emails. Specifically, by changing certain properties of the e-mail, such as the From field in the message header, an adversary can make the e-mail appear to be from someone rather than the actual sender. On the surface, the spoofing email may appear to be from a legitimate company or individual, but it attempts to fraudulently acquire personal information from the recipient, such as account password or credit card information.

Although a number of anti-spam mechanisms have been proposed and deployed to foil spammers, spam messages continue swarming into users’ mailboxes, and hence blocking spam emails is one of 7 hot projects [9]. Of all the spams, a foxy email sent from an impersonated friend or colleague may be harmful but accepted by the recipients. To block this kind of hazard spams, this paper presents a Sender Authentication Protocol (SAP for short) based on the standard SMTP mechanism. Concretely, when a user Bob receives an email from a claimed-sender Alice, he will send a second email to the claimed-sender for confirmation. If Alice is really the claimed-sender of the first email, she will send a third email which is related to the first email, otherwise the claimed-sender is not able to do so. Based on the third email, Bob can detect the fraud even though the adversary sits between Alice and Bob. The above 3-round SAP can be simplified as 1-round without significantly sacrificing security. Because the present SAP employs the standard protocols, it is compatible with the commercial email systems such that it can be easily deployed. In addition, SAP is fully compatible with other anti-spam technologies. To demonstrate the applicability and user-friendliness, SAP is implemented as an add-in of Microsoft Outlook and tested successfully.

The remainder of this paper is as follows. Section II elaborates our SAP protocol. The SAP implementation and experiments are presented in Section III. The related work are described in Section IV. The last Section concludes the paper.

II. THE SENDER AUTHENTICATION PROTOCOL

A. Security model

In an email system, both sender Alice and receiver Bob have email accounts but they may be in different domains. Suppose an attacker Eve is a stateless adversary to cheat Bob. That is to say, Eve likes to impersonate Alice to send phishing emails to Bob, and eavesdrop their communication. Moreover, Eve maybe sit between Alice and Bob to tamper their communication without recording all the history of their network traffic.

This security model distinguishes man-in-the-middle adversary from stateless adversary. Albeit both of the adversaries sit between the sender and the receiver as a corrupted router, their difference is that a man-in-the-middle attacker will record all the traffics in the network, but a stateless adversary will not.

1An email header has a From field which states the source of the email. We call that source as claimed-sender. Note that claimed-sender may be not the real sender due to email spoofing.
Thus a man-in-the-middle adversary is more powerful than a stateless adversary. In reality, schemes against man-in-the-middle attack are designed for applications of top security (e.g., e-banking). In the email spoofing case of interest, an adversary is likely to send spoofing emails to many people and wait for random victims, and will not be patient to monitor some target users all the time. That is to say, the adversary does not hunt for target victims in advance and is unlikely to spend resource to record all the user’s traffic. Therefore man-in-the-middle adversaries do not massively exist in email spoofing activities, and are beyond our scope. Nonetheless, a stateless adversary is still powerful, and is able to defeat many existing protection methods addressed in Section IV because he can intercept the traffic and manipulate the communication messages at will.

B. Sender authentication protocol

As mentioned in Subsection II-A, Eve is able to impersonate Alice to send foxy emails to Bob so as to take advantage of Bob given that Bob trusts Alice. In order to prevent Eve from sending fraudulent emails in name of Alice, the standard email process is revised as a new SAP process in Fig. 1. Specifically, the SAP includes 3 conventional emails sequentially:

- **Normal email**: Both sender Alice and receiver Bob carry on the conventional email process. The MTAa (mail transfer agent) of Alice sends the first message with Message_ID ID1 to MTAb of Bob, but the message field in the email is the encryption id = $E(ID_1, ID_{old})$ of ID1 instead of ID1 itself, where IDold, the message identifier of an old email (e.g., the most recent email), is reused as an encryption key, where $E(\cdot, k)$ is an encryption algorithm whose corresponding decryption algorithm is $D(\cdot, k)$.

- **Email for proof request**: After the MTAb receives the first email, it replies to the sending MTAa a second email with a special Message_ID ID2=0. If the second email is not reachable, the first email is identified to be bogus. For the sake of higher security, Bob may send to Alice a challenge c which states a request for another old designated email 2.

- **Email of proof**: After receiving the special email ID2, MTAb generates a third email with message_ID ID3 = $H(IM_1, ID_{old}, c)$, and sends the third email to the MTAb, where $H(\cdot)$ is a one-way hash function.

Finally, MTAb searches all the emails ID1 in Bob’s mailbox, and then calculates $ID_1' = D(id, ID_1)$ one by one. If there is an email ID1 such that $ID_3 = H(ID_1, ID_1', c)$, MTAb will mark the first email as authentic and replace its Message_ID field with ID1 for verifying future emails, otherwise, it will identify the first email as spoofing, and signal a warning. Note that the second email and third email will be destroyed after the protocol is complete.

2If the receiver does not send challenge c, SAP can be simplified into a one-round protocol. That is, the 1st email and 3rd email is merged into one email, and the second mail is deleted. Thus, the SAP protocol can be carried out by MUA (Mail User Agent), and hence can be deployed easily.

C. Security analysis

In order to send to Bob a spoofing email successfully, an adversary Eve can launch the phishing attack as any of the following roles:

1. **Non-existing sender**: Eve sends to Bob an email with an invalid sender address. Clearly, the present scheme can block this kind of emails easily because Bob can not find a matching old email at the verification step.

2. **External sender**: Suppose that Alice and Bob have email accounts from different email domains, and Eve sits between their MTAs by compromising a route for manipulating their emails at will. With the assumption that Eve will not record all the old emails, Eve can not forge the proof email to MTAb in Fig. 1, hence, Eve fails.

3. **Internal sender**: Assume that Alice and Bob are in the same email server domain. Eve will impersonate Alice to send a spoofing email to Bob. In principle, this spoofing activity can be detected easily by the MTA server with local user authentication. Technically, whenever Eve likes to send an email via the server in name of Alice, the server will check the sender address by a request for email account/password pair. Unfortunately, this simple and effective defense is not always deployed in practice for the sake of user friendliness. For example, Microsoft Outlook™ merely authenticates the user identity when he/she logs in, rather than when he/she sends emails. Luckily, SAP can detect the internal adversary without sacrificing user-friendliness. The reason is that Eve does not have the old emails sent from Alice to Bob even though Bob has some authentic emails sent from Eve, and hence Eve can not create a reply email ID3 which can pass the verification of Bob.

In all, SAP reuses the previous Message_ID as a shared secret between the sender and the receiver so as to authenticate the email source. If the adversary Eve does not record the very first email, he can not forge a valid proof email and fails to cheat the recipient. Furthermore, if an adversary knows a previous Message_ID (secret) at some time but misses one email later, he can not continue to cheat the recipient any more assume that the most recent email is used as a shared secret.
III. IMPLEMENTATION AND EXPERIMENT

A. Design and development

In the implementation, MUA is used to perform SAP protocol because it is easy to deploy. We selected Windows XP operating system and Outlook™ as an implementation platform because it is the most popular email client or MUA, and developed an add-in which implements SAP with Microsoft Visual Studio. We adopt the 3-round protocol for sake of easy explanation\(^3\).

According to the article [10], Microsoft Office supports a uniform design architecture for building Office application (e.g., Word, Excel, and Outlook) add-ins. Typically, such add-ins are housed in ActiveX dlls and can be dynamically loaded and unloaded by the user through the Office application. To implement SAP add-in as a COM Automation component for extending Outlook™, the development process takes 3 steps:

1) Building interface with Outlook:: As an Office COM add-in, SAP code must implement the _IDTextensibility2 interface defined in the MSAdd-in Designer typelibrary (MSADDNDR.dll/MSADDNDR.tlb) file. It has the following five methods:

- OnConnection: called when the SAP add-in is loaded into computer memory. This method initializes event explorer, event adviser etc.
- OnDisconnection: called when the add-in is unloaded from memory. It releases the resource allocated in OnConnection.
- Onadd-insUpdate: called when a set of other add-ins changes;
- OnStartupComplete: called only if the add-in is loaded during application startup. It creates a Temporary folder so as to hold the incoming email tentatively. Only if the email is authentic, it will be moved to the trusted email folder Inbox.
- OnBeginShutdown: called if SAP add-in is disconnected when Outlook™ quits.

2) Enabling event handler:: In Office add-in, there are a lot of events and corresponding handling functions. For example, in Microsoft Outlook™, when an authenticated user sends a request to the Outlook Exchange Server, the server will send back the emails to Outlook™ such that Outlook™ generates an incoming email event. The event will trigger a notification to the add-ins which care about the event. Hence, SAP should set up a map between each event and its corresponding member function. The code for building Message MAP is

```
BEGIN_SINK_MAP(COadd-in)
SINK_ENTRY_INFO
    (ID noopof(Outlook::ApplicationEvents),
     OnNewMail,&OnInMailInfo)
END_SINK_MAP()
```

3-round protocol requires that the receiver’s Outlook™ be on line, but 1-round protocol does not have this requirement.

According to the map, whenever a new email arrives, the event handler OnNewMail() will be activated so as to perform the following tasks:

1) **Analyzing incoming email**: From the header of the incoming email, the SAP add-in reads the claimed-sender (e.g. Alice) address and message identity

\[ id = \mathcal{E}(ID_1, ID_{old}) \]

and moves the email into Temporary folder.

2) **Requesting for proof**: In accordance to Fig.1, SAP add-in creates a second email with Message ID\(_2\) =0, and replies it to the claimed-sender Alice. If the second email is not reachable, the first email is identified as spam and removed. After receiving the second email, Alice will generate a message identity

\[ ID_3 = \mathcal{H}(ID_1, ID_{old}), \]

and send a third email with the new message identity ID\(_3\) to Bob again.

3) **Phishing email identifying**: Bob searches his emails sent from Alice, if he can find an email with message identity ID\(_1\) such that

\[ \mathcal{H}(D(id, ID_i), ID_i) \equiv ID_3, \]

then the first email is authentic and is moved into Inbox from Temporary folder, otherwise, the first email is highlighted as a spam.

3) **Registering add-in::** In order to embed the SAP add-in into Outlook™, we have to register SAP add-in into the correct registry of Windows so that the code can be used in Outlook environment. To this end, define one registry key HKCU\{Software\Microsoft\Office\Outlook\add-ins with attributes

```
‘Oadd-in.Oadd-in’
{
    val FriendlyName = s ‘SAPemail add-in’
    val Description = s ‘SAPemail Outlook add-in’
    val LoadBehavior = d ‘00000003’ // Automatically load
    val CommandLineSafe = d ‘00000000’
}
```

in file Oadd-in.rgs, compile the whole code with Visual studio into SAPemail.dll, and run tool regsvr32 SAPemail.dll.

B. Experiments

The present SAP scheme is totally transparent to the email user. That is to say, when a user runs Microsoft Outlook™, he does not know whether the protection is running or not. To demonstrate the protection function and user friendliness, we perform 2 experiments.

\(^3\)3-round protocol requires that the receiver’s Outlook™ be on line, but 1-round protocol does not have this requirement.
1) Normal email:: As usual, after starting Microsoft Outlook™ embedded with SAP add-in, we observe that the user interface is identical to the original one. Afterwards one author composes an email and sends it to another author. The email is sent successfully, and the receiver can read the email without any difference from the original Outlook™. If we run the SAP code step by step, we know that the email is inserted in the Temporary folder first and then moved into Inbox later.

2) Foxy phishing email:: As shown in Fig.2., with a handy tool SimpleCheck, an adversary Eve can impersonate Alice to send email to Bob. Because the email is compliant with SMTP protocol, the original Outlook™ can not identify this spoofing email. However, the enhanced Outlook™ with SAP add-in is able to identify it as a bogus email because there is no valid proof.

IV. RELATED WORK AND COMPARISON

Up to now, a number of anti-spam schemes have been proposed and deployed in email systems, e.g. spam blockers (www.mytrashmail.com/spam_protect.aspx) and DCC (Distributed Checksum Clearinghouse, http://www.rhyolite.com/dcc/). These defense technologies can be classified into 4 categories.

A. Blacklist of senders

One straightforward way to filter out spamming emails is to check a sender against predefined lists (whitelist or blacklist [11]) or rules. For example, Microsoft Outlook™ defines some rules for classifying emails. If an email does not meet the Junk E-mail Filter’s definition of spam, it will be sent into the Inbox folder. Otherwise, if the email does not meet the definition of a phishing rules, Outlook™ keeps the email in the folder Inbox but disables the links in the email and prevents the user from replying to the mail, puts it into Junk E-mail folder otherwise.

Certified e-mail (e.g., KobeCertified by KobeMail, CertifiedEmail by Goodmail Systems) is an e-mail whitelisting technique by which an internet service provider allows someone to bypass spam filters when sending e-mail messages to its subscribers, in return for paying a fee to the certifying service. A sender can then be sure that his messages have reached their recipients without being blocked, or having links or images stripped out of them, by spam filters. The purpose of certified e-mail is to allow companies to reliably reach their customers by e-mail, while giving recipients certainty that a certified message is legitimate and is not a forged phishing attempt.

Although blacklist/whitelist is effective and simple, it is not trivial to build the lists or rules. One way is to differentiate a human sender from a machine sender [12], [13]. It requires any sender to identify himself/herself in the first time. Technically, the recipient requests the sender to solve an artificial intelligence “puzzle”. When the sender completes the verification request, the address is automatically placed on the recipient’s whitelist.

B. Blacklist of network addresses

RFC 4406 or Sender ID [14] seeks to verify that every e-mail message originates from the Internet domain from which it claims to have been sent. This is accomplished by checking the address of the server that sent the mail against a registered list of servers that the domain owner has authorized to send e-mail. This verification is automatically performed by ISP or the recipient’s mail server before the e-mail message is delivered. To employ sender ID to defeat spoofing emails, e-mail senders and domain owners must publish or declare all of the Internet Protocol (IP) addresses used by their outbound e-mail servers, or the IPs authorized to send e-mail on their behalf, in the Domain Name System (DNS). These IPs are included in a Sender Policy Framework (SPF) text file. Tauberer [15] extends the Mozilla Thunderbird email application that reports, when possible, whether the claimed-sender was actually the sender of the email. The extension uses SPF in a nonstandard way and DomainKeys (DK) to verify the sender’s domain.

RFC 4408 [16] validates the HELO domain and the claimed-to-be address of SMTP protocol. It recommends that SPF clients not only check the claim-to-be address, but also separately check the HELO identity by applying the optional check_host(ip, domain, sender) function to the HELO identity as the sender.

In RFC 4871 [17], Domain Keys is an e-mail authentication system designed to verify the DNS domain of an e-mail sender and the message integrity. The Domain Keys specification has adopted aspects of identified Internet mail to create an enhanced protocol called Domain Keys Identified Mail (DKIM).

C. Signed email

Cryptography tools, such as S/MIME (Secure/Multipurpose Internet Mail Extensions)[18], Multisignature[19], or TMTP (Trusted Message Transfer Protocol[20]), enable to authenticate email senders and ensure the integrity of messages in transit. But they are dependent on the public key infrastructure which is not easy to deploy.

D. Proof-carrying email

Howe [21] presented an enhanced Message-ID generator and validator of an email system. The enhanced Message-ID comprises a symbolic name used to identify the presence of an enhanced Message-ID header, a timestamp value indicating when a modification was applied to the header, an original value of the Message-ID header, and a textual representation of a one-way security hash generated from the timestamp value, the original value, and a secret phrase.

George [22] designed a method and system for email sender verification. A tag is included in an email that contains an identity datum for the sender that is associated with their email address in a server. The email is sent via a network, thus associating a sender address with the email. At least the identity datum, taken from the tag, is then provided to the server, to retrieve the email address. The email address and the sender address are then compared, verifying that the email came from the sender if they match.

E. Comparison

Table I compares the performance of anti-phishing technologies in 5 aspects. The 2nd column is the cost of communication, the 3rd column is the computational cost. The 4th column is the compatibility with the standard SMTP. Although SMIME is a standard in itself, not all of email systems implement it because it relies on public key infrastructure. The 5th column is the protection capability the attacker within the same domain as the victim. The last column shows the defense capability against man-in-the-middle attack.

<table>
<thead>
<tr>
<th>Blacklist</th>
<th>Traffic</th>
<th>Compu.</th>
<th>Compa.</th>
<th>DNS</th>
<th>MITA</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC4406[14]</td>
<td>low</td>
<td>low</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RFC 4408[16]</td>
<td>low</td>
<td>low</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RFC 4871[17]</td>
<td>low</td>
<td>low</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SMIME[18]</td>
<td>high</td>
<td>high</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Howe[21]</td>
<td>low</td>
<td>low</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>George[22]</td>
<td>low</td>
<td>low</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Our SAP</td>
<td>medium</td>
<td>low</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

1 Man-in-the-Middle attack;
2 The computation cost for building blacklist;
3 Stateless attacker only.

V. Conclusion

E-mail spoofing is an fraudulent e-mail activity in which the sender address and other parts of the e-mail header are altered to appear as though the e-mail originated from a different source. Although there are many defense tools and products in the market, email spoofing is still prevalent in reality. The present protocol can reduce the amount of spoofing emails based on the historical email records assume that an adversary will not record a common user’s traffic all the time due to financial or other reasons. However, we have to point out that the receiver should keep at least one message identity (e.g., the most recent email) for verification purpose.

We developed a prototype-of-concept as an add-in of Microsoft Outlook™ email system. The behavior of the add-in is totally transparent to the user, i.e., it does not change the user interface but reduces the spoofing emails silently.

References