ARTICLE IN PRESS

Available online at www.sciencedirect.com
ScienceDirect
Computer Networks xxx (2007) xxx–xxx
Computer Networks
www.elsevier.com/locate/comnet

SPP: An anti-phishing single password protocol ☆

Mohamed G. Gouda [a], Alex X. Liu [b,*], Lok M. Leung [a,1], Mohamed A. Alam [a,1]

[a] Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712-0233, USA
[b] Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824-1266, USA

Received 20 August 2006; received in revised form 19 March 2007; accepted 24 March 2007

Responsible Editor: D. Frincke

Abstract

Most users have multiple accounts on the Internet where each account is protected by a password. To avoid the headache in remembering and managing a long list of different and unrelated passwords, most users simply use the same password for multiple accounts. Unfortunately, the predominant HTTP basic authentication protocol (even over SSL) makes this common practice remarkably dangerous: an attacker can effectively steal users' passwords for high-security servers (such as an online banking website) by setting up a malicious server or breaking into a low-security server (such as a high-school alumni website). Furthermore, the HTTP basic authentication protocol is vulnerable to phishing attacks because a client needs to reveal his password to the server that the client wants to login.

In this paper, we propose a protocol that allows a client to securely use a single password across multiple servers, and also prevents phishing attacks. Our protocol achieves client authentication without the client revealing his password to the server at any point. Therefore, a compromised server cannot steal a client's password and replay it to another server.

Our protocol is simple, secure, efficient and user-friendly. In terms of *simplicity*, it only involves three messages. In terms of *security*, the protocol is secure against the attacks that have been discovered so far including the ones that are difficult to defend, such as the malicious server attacks described above and the recent phishing attacks. Essentially our protocol is an anti-phishing password protocol. In terms of *efficiency*, each run of our protocol only involves a total of four computations of a one-way hash function. In terms of *usability*, the protocol requires a user to remember only one password consisting of eight (or more) random characters, and this password can be used for all of his accounts.

Keywords: Password protocols; Network security; Protocol design; Authentication; Phishing attacks

1. Introduction

Authentication of an entity is usually done by verifying one or more of the followings:

1. Something the entity *is* (by biometric techniques, such as fingerprint or voiceprint identification).

☆ This material is based upon work supported by the National Science Foundation under Grant No. 0520250.
* Corresponding author. Tel.: +1 517 353 5152; fax: +1 517 432 1061.
E-mail addresses: gouda@cs.utexas.edu (M.G. Gouda), alexliu@cse.msu.edu (A.X. Liu), raymond1@cs.utexas.edu (L.M. Leung), malam@cs.utexas.edu (M.A. Alam).
[1] Lok M. Leung and Mohamed A. Alam participated in this work while they were undergraduate students in The University of Texas at Austin.

1389-1286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.03.007

2. Something the entity *has* (by PKI certificate, ID cards, smart cards).
3. Something the entity *knows* (by passwords, PINs).

Of these three authentication methodologies, only the last two are suitable for remote authentication. On today's Internet, server authentication is usually done by SSL [5] using PKI certificates, which is something the server has, while client authentication is usually done using passwords, which is something the client knows.

1.1. The problem

Many people today have multiple accounts on the Internet. For example, one may have an email account on www.yahoo.com, a travel account on www.travelocity.com, a credit card account on www.discovercard.com, a banking account on www.chase.com, an online stock trading account on www.fidelity.com, etc. Forrester Research reports that a typical web user manages an average of 15 passwords on a daily basis [10]. Most of these accounts are protected by passwords. As more services move to the Internet, the number of accounts a user needs to manage is expected only to grow. If one uses different and unrelated passwords for each account, then remembering all these unique passwords is a daunting task. It has been observed in [1] that a typical user can only remember 4 or 5 passwords effectively. Because of this, the common practice is to use the same password for multiple accounts. However, the predominant HTTP basic authentication protocol (even over SSL) [4] makes this practice remarkably dangerous because the protocol allows a server to know the password of each of its clients.

Let us first examine how the HTTP basic authentication protocol works. A client C first registers with a server (such as a web site) S using password P. The registration results in S storing in its password file the pair of user name C and password verification information $MD(P)$. Here MD denotes a message digest (one-way hash) function such as $MD5$ [16] and SHA-1 [18]. Later on when C wants to login on S, C sends his user name C and password P to S. Then S applies the message digest function MD to the received password and compares the result with the stored password verification information $MD(P)$. If they are equal, then the authentication of C is successful; otherwise it is unsuccessful. The HTTP basic authentication protocol usually runs on top of SSL [5], which allows the client to authenticate the server using certificate mechanisms, and provides an encrypted channel between the client and the server.

Allowing a server to know the passwords of its clients like the HTTP basic authentication protocol does is incredibly dangerous because a server may be untrustworthy. An attacker can set up a malicious server (at cost as low as $100 US dollars), and allures people to register using passwords by offering free goods or services. The attacker can reasonably estimate that some of his clients use the same password for their financial accounts. After gathering those passwords from the registered clients, the attacker can impersonate them to login on some financial servers such as an online banking or stock trading server, which could cause significant loss to the clients. Furthermore, a server may be compromised. Instead of directly trying to break into a high-security server such as an online financial server to gain unauthorized access, which would be difficult, an attacker can first try to break a low-security server such as an art bulletin board set up by an amateur. Once a low-security server is broken, which is relatively easy, since some clients of the broken poorly-defended server may use the same password for their accounts on high-security servers, the attacker can then use the captured passwords to access those high-security servers.

Some users classify servers into high-security servers and low-security servers, and use one password for high-security servers and another different password for low-security servers. This practice is remarkably insecure. First, this classification is highly subjective and a typical user may not do it right. This would cause the same password being used on both a high-security server and a low-security server. Second, the two different passwords are possibly related, such as by being formed according to the same pattern. Thus knowing one password might enable an attacker to discover the other one. Third, not every high-security server is trustworthy. For example, an employee of a high-security server may be bribed to breach the passwords of its clients to attackers. In addition, gathering a user's personal information from multiple low-security servers could be helpful in discovering the other password for high-security servers.

We call the attacks of stealing passwords by setting up a malicious server or by compromising a

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007

benign server "*malicious server attacks*". We use the term "malicious servers" to denote the servers that are either set up or compromised by an attacker. When using the HTTP basic authentication protocol, a user has to use a different and unrelated password for every different server to prevent malicious server attacks. Remembering many totally different and unrelated passwords is certainly not viable for most users. And writing down all the user names and passwords on a piece of paper is certainly not a good idea because compromising of this list could cause serious loss.

1.2. Our solution

In this paper, we propose a new password protocol, named Single Password Protocol (short for SPP), which allows a user to use one single password (and one single user name) for all of his accounts while defeating malicious server attacks. SPP uses two basic techniques: challenge/response and one-time server-specific tickets.

SPP works basically as follows. Let P be the single password that a client C remembers. When C registers with a server S, C generates a challenge and ticket verification information, then sends them to S, and S stores them in its password file. Later on, when C tries to login on S, S prompts C with the stored challenge. Then C uses the challenge, the server's name S, and his password P to mint a one-time server-specific ticket, together with a new challenge and new ticket verification information, and sends them to S. Then, S verifies the received ticket using the stored ticket verification information. If the ticket is valid, then the authentication of C is successful, and S subsequently replaces the stored challenge and ticket verification information by the new challenge and new ticket verification information that S received along with the one-time server-specific ticket from C.

SPP allows a client to use the same user name and password for all of his accounts while defeating malicious server attacks because of the following two reasons:

1. *A server never knows a client's password at any time.* In SPP, a client uses the challenge received from a server, the server's name, and his password, to mint a one-time server-specific ticket using a one-way hash function, and sends the one-time server-specific ticket, instead of his password, to the server for authentication. The password of a client is used for minting a one-time server-specific ticket, and it is the ticket that is used for authenticating the client. The server cannot feasibly compute the user's password based on the one-time server-specific ticket and the ticket verification information stored in the server due to the use of one-way hash functions.
2. *Each ticket can only be used once.* At any moment, for one client, ticket verification information in different servers are different. Therefore, a malicious server cannot replay a received ticket to other servers to gain unauthorized access. At any server, for one client, ticket verification information are changed unpredictably after each successful login.

The rest of this paper proceeds as follows. In Section 2, we present the single password protocol, while in Section 3, we give detailed security analysis of this protocol. In Section 4, we review and examine existing password protocols and compare them with our protocol. Section 5 concludes.

2. Single password protocol

In this section, we present our Single Password Protocol (SPP for short). For ease of understanding, we first present four intermediate versions of it starting from the HTTP basic authentication protocol. We show that each intermediate version is vulnerable to a particular attack, and each attack is countered by the following versions.

The notations used in this section is listed in Table 1.

Note that the message digest (one-way hash) function $MD()$ used in this paper is assumed to have the property that a polynomial-bounded adversary should not be able to gain any information about the input by examining the output of such a function. Example viable candidates include $MD5$ [16] and SHA-1 [18].

Table 1
Notations

C	Client
S	Server
P	Password remembered by client
n, n_i	Random number
$MD()$	Message digest (one-way hash) function
$MD^2()$	$MD(MD())$
$\|$	Concatenation

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007

2.1. Version I

The first version is the HTTP basic authentication protocol, which is shown in Fig. 1. In this protocol, for each registered client C, whose password is P, server S stores $MD(P)$ as password verification information in a password file. We assume server S has some out-band means of authenticating the initial registration, discussion of which is out the scope of this paper. Each time client C tries to login on server S, C sends his user name C and password P to S. Then S uses its stored password verification information $MD(P)$ to authenticate P. More precisely, S first computes the message digest of the received password, and then compares the result with its stored password verification information $MD(P)$. If they are equal, then the authentication is successful.

Despite the wide use of the HTTP basic authentication protocol, it suffers from *malicious server attacks*. To launch this type of attack, an attacker first sets up a malicious server and allures people to register with him using a user name and a password. Because people often use the same user name and password on multiple servers, the attacker is able to henceforth impersonate such clients to other servers such as an online banking website. Clearly, malicious server attacks can be extremely damaging.

2.2. Version II

To counter malicious server attacks, we use the challenge/response technique. When client C registers with server S, he first generates a random number n, computes $MD(n|P)$, and then sends them to S. Server S stores n as a challenge and $MD(n|P)$ as password verification information in its password file for client C. Note that the party who remembers n is server S, not client C, although n is generated by C. Client C still only remembers his password P. Since C generates a random number as a challenge independently for each registration, the possibility that the two challenges on two different servers are equal is negligible. Each time client C tries to login on server S, S sends n as the challenge back to C. Then C computes the message digest of $(n|P)$ and sends the result as the response to S. Server S compares this response with the stored password verification information $MD(n|P)$. If they are equal, then the authentication is successful. Note that client C does not send his password P to S, even in the initial registration. Knowing $MD(n|P)$ does not allow server S to gain any information about the password P, and does not allow server S to impersonate client C to other servers. Therefore, this protocol is secure against malicious server attacks. This password protocol as version II is shown in Fig. 2.

Compared to Version I, in Version II, the number of messages is increased by one, the computational cost of the client is increased by the cost of computing a one-way hash function, and the computational cost of the server is decreased by the cost of computing a one-way hash function.

Unfortunately, this password protocol is vulnerable to *password file compromise attacks*. To launch this type of attack, an attacker first steals the password file of a server by some means such as breaking into the server or colluding with insiders of the server; second, the attacker tries to discover either the password of the client or the valid response that the client would use to login on the server using the information in the appropriated password file. According to the above version II protocol, stealing the password file of server S enables the attacker to impersonate any of its clients because the response is the same as the password verification information.

2.3. Version III

To counter password file compromise attacks, the response should not be computable from the password verification information. Accordingly, we change the password verification information from $MD(n|P)$ to $MD^2(n|P)$. Note that an attacker cannot deduce the response $MD(n|P)$ from the password verification information $MD^2(n|P)$ in polynomial time. This password protocol of version III is shown in Fig. 3.

C knows	P
S stores	$MD(P)$
$C \rightarrow S$:	$C\|P$

Fig. 1. Version I (vulnerable to malicious server attacks).

C knows	P
S stores	$n, MD(n\|P)$
$C \rightarrow S$:	C
$C \leftarrow S$:	n
$C \rightarrow S$:	$MD(n\|P)$

Fig. 2. Version II (vulnerable to password file attacks).

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007

C knows	P
S stores	$n, MD^2(n\|P)$
$C \rightarrow S$:	C
$C \leftarrow S$:	n
$C \rightarrow S$:	$MD(n\|P)$

Fig. 3. Version III (vulnerable to message log attacks).

C knows	P
S stores	$n_i, MD^2(n_i\|P)$
$C \rightarrow S$:	C
$C \leftarrow S$:	n_i
$C \rightarrow S$:	$MD(n_i\|P)\|n_{i+1}\|MD^2(n_{i+1}\|P)$

Fig. 4. Version IV (vulnerable to server spoofing attacks).

Compared to Version II, in Version III, the number of messages remains the same, the computational cost of the client remains the same, but the computational cost of the server is increased by the cost of computing a one-way hash function.

Unfortunately, this password protocol is vulnerable to *message log compromise attacks*. Some servers with stringent security requirements record every message that they send or receive into a message log. To launch message log compromise attacks, an attacker first steals the message log of a server just like stealing the password file in the password file compromise attack; second, the attacker tries to gain unauthorized access using the information in the appropriated message log. In the above protocol of version III, the response that client C uses to authenticate himself to server S, namely $MD(n|P)$, may be stored in the message log of S. Stealing such a message log of server S enables the attacker to impersonate any of its clients because the response that a client uses to authenticate himself to a server is always the same.

2.4. Version IV

To counter message log compromise attacks, we use the technique of one-time tickets. Most tickets in real life, such as movie tickets, are one-time tickets in the sense that they can only be used once. Here, we want the response from a client to contain a one-time ticket, which can be used to authenticate the client only once. Therefore, the response from a client in each login needs to be unique every time. To achieve this uniqueness, we change the random number n involved in the protocol accordingly. Let n_i be the challenge and $MD^2(n_i|P)$ be the ticket verification information stored in server S for client C. Every time when C tries to login on server S, S sends the challenge n_i back to C. Then, C first computes the one-time ticket $MD(n_i|P)$; second, C chooses a new challenge n_{i+1} and computes new ticket verification information $MD^2(n_{i+1}|P)$; third, C sends all of them $(MD(n_i|P)|n_{i+1}|MD^2(n_{i+1}|P))$ to S. When S receives this message, S first verifies the received one-time ticket $MD(n_i|P)$ using the stored ticket verification information $MD^2(n_i|P)$. If the one-time ticket is valid, then the authentication is successful and subsequently S replaces the stored n_i and $MD^2(n_i|P)$ by n_{i+1} and $MD^2(n_{i+1}|P)$. This password protocol of version IV is shown in Fig. 4.

Compared to Version III, in Version IV, the number of messages remains the same, the cost of the client is increased by the cost of computing two more hash functions, and the cost of the server is increased by updating one database record.

Unfortunately, this password protocol is vulnerable to *server spoofing attacks*. To launch server spoofing attacks, a malicious server S first pretends to be client C and tries to login on another benign server S', who has C as a client and stores n'_i as the challenge and $MD^2(n'_i|P)$ as the ticket verification information for client C. Responding to the login request from S, S' sends the challenge n'_i to the malicious server S. After malicious server S obtains n'_i from server S', S stops communicating with S'. Later on, when C tries to login on S, S sends to C the stolen challenge n'_i, instead of the correct one n_i. Since C only remembers his password P, C assumes that this challenge is the one stored in S. Consequently, C sends to S the response $MD(n'_i|P) \mid n'_{i+1}|MD^2(n'_{i+1}|P)$, which can be used by the malicious server S to login on S' as client C.

2.5. Final version

To counter server spoofing attacks, we make the one-time ticket from a client to be specific to the server that the ticket is intended for. More precisely, we change the one-time ticket from $MD(n_i|P)$ to $MD(n_i|P|S)$, and accordingly change the ticket verification information from $MD^2(n_i|P)$ to $MD^2(n_i|P|S)$. In this way, when malicious server S sends to client C the challenge n'_i, which he steals from benign server S', C will send to S the corresponding response

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007

C knows	P
S stores	$n_i, MD^2(n_i\|P\|S)$
$C \rightarrow S$:	C
$C \leftarrow S$:	n_i
$C \rightarrow S$:	$MD(n_i\|P\|S)\|n_{i+1}\|MD^2(n_{i+1}\|P\|S)$

Fig. 5. Single Password Protocol (SPP).

$$MD(n'_i|P|S)|n'_{i+1}|MD^2(n'_{i+1}|P|S),$$

which cannot be replayed as a valid response to S'. The valid response to S' should be

$$MD(n'_i|P|S')|n'_{i+1}|MD^2(n'_{i+1}|P|S').$$

These modifications complete our single password protocol, which is shown in Fig. 5.

Compared to Version IV, in this final version, the number of messages remains the same, the cost of the client and the server also remain the same.

Note that in this paper, we assume that SPP is running on top of SSL. A client authenticates a server using SSL before SPP starts. Therefore, a malicious server cannot forge its identity.

3. Security analysis

In this section, we discuss the security of SPP. We start our discussion by the assumptions we need to make for SPP to be secure. Then we analyze the security properties of SPP.

3.1. Assumptions

For SPP to be secure, we have two reasonable assumptions. First, we assume that SPP is used with the Secure Sockets Layer (SSL for short). Second, we assume that a user is capable of remembering one password of eight (or more) random characters for all his accounts.

3.1.1. Using SSL

The current industry standard for securing communication over the Internet is SSL, which was developed by Netscape in 1994 [5]. SSL has been built into all major web browsers, web servers, email clients, email servers, etc. The SSL protocol runs mainly in three steps:

1. *Server authentication*: The server sends its certificate to the client. The client authenticates the server using the certificate. Note that the certificate contains the public key of the server.
2. *Key establishment*: The client generates a session key, encrypts it with the public key of the server, and sends the result to the server. The server then decrypts the message using its private key and obtains the session key. Henceforth the session key between the client and the server is established.
3. *Secure communication*: All the subsequent communication is encrypted with the session key.

A higher level protocol can easily layer on top of the SSL protocol transparently because SSL is application protocol independent. Based on the wide availability of SSL implementations, we assume that SPP is layered on top of SSL. In other words, SPP runs in the third step (secure communication) of SSL. All the messages involved in our protocol are encrypted with a session key. Note that the client authenticates the server using the server's certificate when the SSL connection is being initiated and the server authenticates the client using SPP after the SSL connection is established. In the sense of mutual authentication, SPP is complementary to SSL.

It is worth noting that SSL only provides server authentication if the server's certificate is carefully checked. This is usually done by a web browser. A web browser, such as Internet Explorer and NetScape, is usually preloaded with a list of trusted certificate authorities. When a client tries to access a web site whose certificate has expired or is not signed by a trusted certificate authority, the web browser usually prompts a warning message to the client saying that the certificate should not be trusted. In this paper, we assume most clients will stop accessing such web sites, or at least refrain from giving out any confidential information to such web sites.

3.1.2. Using strong passwords

Using SPP, a user can safely use the same pair of user name and password for all the servers with which he registers. We assume that the single password that a user remembers consists of eight (or more) random characters. The characters are the ones available on standard keyboards. Remembering random characters seems intimidating, but remembering only a single string of eight random characters in one's life time for all accounts does not. How to generate random passwords, either by software or by humans according to some high

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007

quality password policies, is out the scope of this paper. Next, we will see that the above length and randomness requirements of one's single password defeat brute force attacks and online/off-line dictionary attacks.

3.2. Security properties

Here, we discuss a list of 16 types of attacks that can be launched on password protocols. This list covers all the password attacks that appeared on previous literatures. We show that SPP is secure against each of them.

1. *Eavesdropping attacks*: In this type of attack, an attacker listens to all the communication between a client and a server, and tries to discover the client's password or tickets. Since SPP runs on top of SSL and all the communication between a client and a server is encrypted with a session key established by SSL, an eavesdropper cannot discover the client's password or tickets. Note that without SSL, an eavesdropper can discover the one-time ticket, but still cannot discover the client's password due to the use of a one-way hash function.
2. *Message replay attacks*: In this type of attack, an attacker first listens to all of the communication between a client and a server, then tries to login on the server by replaying some messages that the attacker captured previously. SPP is secure against message replay attacks for two reasons. First, SPP messages are encrypted by different SSL session keys in different runs of SPP. Replaying a message of one SSL session to another SSL session is useless. Second, even without the help of SSL, SPP itself is secure against message replay attacks. If an attacker replays an old challenge to the client, the client will create the old ticket that corresponds to the old challenge, but the server cannot verify the old ticket because every ticket can be used only once. Similarly, replaying an old ticket is useless. Note that the ticket verification information stored in the server is modified if and only if the ticket from the client is successfully verified. Replaying any previous message cannot cause the ticket verification information to be changed because the replayed messages cannot be verified successfully.
3. *Message loss attacks*: In this type of attack, an attacker drops some messages between a client and a server with the hope of discovering the client's password or tickets. Launching this type of attack on SPP can only cause the authentication between the client and the server to fail, but cannot enable an attacker to gain any information about the client's password or tickets.
4. *Message modification attacks*: In this type of attack, an attacker modifies some messages between a client and a server on the fly with the hope of discovering the client's password or tickets, or gaining unauthorized access. Since every message in SPP is encrypted by a session key established by SSL, launching message modification attacks on SPP has the same effect as launching message loss attacks on SPP. In other words, modifying the messages in SPP can only cause the authentication between the client and the server to fail, but cannot enable an attacker to gain any information about the client's password or tickets, or gain unauthorized access.
5. *Message insertion attacks*: In this type of attack, an attacker inserts some messages in the channel between a client and a server on the fly with the hope of discovering the client's password or tickets, or gaining unauthorized access. Since every message in SPP is encrypted by a session key established by SSL, inserting invalid messages into SPP can only cause the authentication between the client and the server to fail, but cannot enable an attacker to gain any information about the client's password or tickets, or gain unauthorized access. For example, if an attacker inserts a message in the channel from a server to a client as the challenge, the client will use its SSL session key to decrypt the message and will use the result, whatever it maybe, to generate the one-time ticket, which consequently will fail to be verified by the server.
6. *Brute force attacks*: To launch brute force attacks, an attacker first obtains some password verification information such as the message digest of a password; second, for every possible combination of characters, the attacker tests whether it is the correct password according to the obtained password verification information. In practice, a pass-

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007

word that consists of eight characters is long enough to be secure against brute force attacks because there are at least 68^8 combinations and testing all of them takes years. Since in SPP the single password that a user remembers consists of eight (or more) characters, SPP is secure against brute force attacks.

7. *Online dictionary attacks*: In this type of attack, an attacker pretends to be someone else and attempts to login on a server by trying every word in a dictionary which contains commonly used passwords. Since in SPP the single password that a user remembers consists of random characters, such a password cannot be found in password dictionaries and cannot be derived from any word in password dictionaries. Therefore, SPP is secure against online dictionary attacks.
8. *Off-line dictionary attacks*: In this type of attack, an attacker first obtains some password verification information such as the message digest of a password, and then, for every word in password dictionaries, the attacker tests whether it is the correct password according to the obtained password verification information. Similarly, because in SPP the single password that a user remembers consists of random characters, SPP is secure against off-line dictionary attacks.
9. *Password file compromise attacks*: In this type of attack, an attacker first steals a server's password file, which stores the password verification information of every client; then the attacker tries to discover either the password of a client using off-line dictionary attacks or the next ticket that a client will use to login on the server. Because SPP is secure against off-line dictionary attacks, the attacker cannot discover the password of any client. Because the message digest function used in SPP is a secure one-way hash function, from the password verification information $MD^2(n_i|P|S)$ stored in the password file, it is not computationally feasible to compute the next ticket $MD(n_i|P|S)$. So, the attacker cannot discover the next ticket that a client will use. Therefore, SPP is secure against password file compromise attacks.
10. *Message log compromise attacks*: This type of attack is similar to the above password file compromise attacks except that the stolen file is not the password file but the message log, which stores all the messages exchanged between a client and a server. In addition, these messages are not encrypted by the session key established by SSL. This is a strong attack since we assume that the attacker could get all the messages of past sessions in plain text. However, SPP is secure against this type of attack for the following two reasons:
    (a) Tickets cannot be reused: In SPP, each ticket is unique and nonpredictable, and each ticket can only be used once. Although an attacker can obtain the tickets that has been used, he cannot use any of these used tickets to gain unauthorized access.
    (b) Password cannot be computed: Similarly, it is not computationally feasible to obtain the password P from the ticket $MD(n_i|P|S)$ and the ticket verification information $MD^2(n_{i+1}|P|S)$.

    Note that the ticket verification information stored in the server is modified if and only if the ticket from the client is successfully verified. Any old ticket in the message log file cannot be successfully verified. Therefore, knowing old tickets does not enable an attacker to be able to modify the ticket verification information in the server.

    In this paper, we assume that the ticket verification data stored in a server is not compromised. Otherwise, an attacker can replace the ticket verification data stored in a server by some previously used ticket verification data, and henceforth the attacker can login to the server using previously used ticket. Such attack scenarios are very unlikely to happen. If an attacker is capable to modify the ticket verification data stored in a server, it means that the attacker has gain the control over the server, and therefore does not need to login on the server as a client.
11. *Malicious server attacks*: In this type of attack, an attacker first sets up a malicious server and allures people to register with the server; second, tries to impersonate one of his clients to login on another server. SPP is secure against this type of attack because of two reasons:
    (a) A client never releases his password to a server, and a server is never able to compute a client's password based on the

password verification information that the client gives to the server.

(b) Replaying a used ticket to any server cannot be successful. Although a client uses the same password on multiple servers, a ticket is valid for one particular server and is valid for only one time.

12. *Server spoofing attacks*: Server spoofing attacks could be launched on challenge/response based password protocols. In this type of attack, a malicious server S first pretends to be one of its clients C and tries to login on another benign server S' where C also has an account. By doing this, S steals the challenge that C stores in S'. Later on, when C tries to login on S, S sends the stolen challenge to C, and wishes that C would send the ticket that S can use to login on S'. SPP is secure against this type of attack because each ticket is specific to the server that it is sent to. Note that a malicious server cannot convince a client that it is another server because SSL handles server authentication using certificates mechanisms.
13. *Phishing attacks*: In this type of attack, an attacker sends fraudulent emails to users, pretending to be the system administrator of a benign website such as an online banking website, and fools users to take login actions on a malicious website, which looks very similar to the benign website, but is set up by the attacker. Once a user tries to login on such a malicious website, his user name and password will be recorded and possibly later will be used by the attacker to login on the benign website. Phishing attacks have skyrocketed in 2004 [2]. According to [2], on average, 5% of recipients of such phishing emails are fooled into responding to them. Clearly, phishing attacks are very effective on the HTTP basic authentication protocol because users send their passwords to the server in each authentication. However, such attacks cannot succeed on SPP because what an attacker obtains is a one-time ticket that is specific to the malicious server (i.e., the domain hosting the malicious web page), and this one-time server-specific ticket cannot be successfully played anywhere else.

Here we elaborate how SPP prevents phishing attacks using an example. Suppose you have an account on www.chase.com. You receive a phishing email pretending to be an authentic email from www.chase.com telling you that you need to login on www.chase.com. After you click the link provided in the email, which looks like www.chase.com, your browser actually displays the web page of a malicious website, say www.chase_usa.com, which looks exactly the same as www.chase.com. Suppose the malicious website www.chase_usa.com has known the value of n_i for your account on www.chase.com. After you type in your password P, your browser generates the one-time ticket

$$MD(n_i|P|\text{www.chase\_usa.com})|n_{i+1}|MD^2(n_{i+1}|P|\text{www.chase\_usa.com})$$

and send it to www.chase_usa.com. Note that the malicious website www.chase_usa.com cannot login on www.chase.com because the valid one-time ticket for www.chase.com is

$$MD(n_i|P|\text{www.chase.com})|n_{i+1}|MD^2(n_{i+1}|P|\text{www.chase.com}).$$

In this example, we assume that your browser, the benign website www.chase.com and the malicious website www.chase_usa.com are running SPP. If the malicious website www.chase_usa.com does not run SPP, your browser should alert you that this website could be malicious.

4. Related work

Many password protocols have been proposed, especially in the past decade. In this section, we review these password protocols and compare them with SPP.

4.1. One-time password protocols

In one-time password protocols, a client uses a different password for every authentication with a server. There are mainly two such protocols: Lamport's one-time password protocol [12] (which was implemented in S/KEY [7–9] and OPIE [14]) and Rubin's one-time password protocol [17]. Both one-time password protocols were designed before the wide deployment of SSL. The motivation for both one-time password protocols are preventing eavesdropping attacks, which are now easily defeated by the confidentiality provided by the widely deployed SSL. To make either of the above one-time password protocols secure against mali-

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007

cious server attacks, a client has to remember multiple "seed" passwords or multiple lists of one-time passwords for multiple accounts. In addition, both one-time password protocols need a client to re-register with its server when the client uses up his current list of one-time passwords. This re-registration requirement is extremely inconvenient for clients because each registration costs their significant effort, and it is extremely harmful for e-commerce servers because it can cause annoyed customers to leave. In contrast, SPP only requires a user to remember one single password for all of his accounts.

4.2. Strong password protocols

Strong password protocols often have strong security properties, but they usually require computationally intensive operations such as modular exponentiations, asymmetric encryptions/decryptions, etc. Many such protocols have been proposed, such as EKE [3] and SRP [19].

Most of these strong password protocols were proposed before the wide deployment of SSL. They are mostly designed to achieve the following two goals: (1) To establish a session key between a client and a server after the server authenticates the client using passwords. This is not a must-have functionality for a password protocol any more because a session key is established after the client authenticates the server using SSL. (2) To prevent dictionary attacks. This goal is always expensive to achieve. Using SPP, dictionary attacks are not a problem because a client is required to choose a single strong password to protect all of his accounts.

Those strong, and consequently heavy weight, password protocols are not well suited for Internet authentication because of the heavy computational cost [6]. A password protocol that is desirable for the Internet must be computationally efficient in both the server side and the client side because of two reasons. First, on the server side, a server on the Internet (mostly a web server) often has heavy service load and consequently cannot afford significant computation for authenticating every client. Second, on the client side, the computing device of a client may have limited computational power, for example, a palm or a cell phone. In contrast, SPP is a light weight password protocol that involves negligible amount of processing time for both the server side and the client side.

4.3. Web-specific password protocols

Except for the HTTP basic authentication protocol mentioned in Section 2, the HTTP specification provides another authentication mechanism: digest authentication protocol [4]. The HTTP digest authentication protocol uses the challenge/response technique, which basically works as follows. When client C registers with server S, S stores C's password P. When C wants to login on S, S generates a nonce n and sends it to C as a challenge. Then C computes $MD(n|P)$ and sends the result to S as a response. Server S verifies the received response using the stored password P and the generated nonce n. Because a server knows the passwords of its clients, the HTTP digest authentication protocol is vulnerable to malicious server attacks and password file compromise attacks. In addition, because the response that a client sends to a server is not specific to that server, the HTTP digest authentication protocol is vulnerable to server spoofing attacks and phishing attacks.

4.4. Single sign-on protocols

The basic idea of single sign-on protocols, such as the Microsoft Passport protocol [15], is to use one central server to authenticate clients for multiple servers, instead of each server authenticating clients by itself. Although single sign-on protocols provide clients the convenience of remembering only one password, which is registered in the single sign-on server, such protocols have the following main disadvantages. First, single sign-on protocols introduces a single point of failure. If the single sign-on server fails working, then all the servers that depend on it fail authenticating their clients, which is extremely destructive. Compromising the single sign-on server has high pay-offs for attackers and thereby makes attack attempts more likely. In fact, Microsoft Passport protocol has already been shown to have security flaws in [11]. Second, as pointed out in [13], single sign-on protocols have high cost of integration because servers need to register with the single sign-on server in order to get the service, and consequently lacks universal adoption. For the servers that do not use single sign-on protocols, a client has to register with them individually using passwords. If a client registers with a malicious server using the same password that he uses for the single sign-on server, then the malicious server can impersonate the client to login on multiple servers.

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007

5. Conclusions

The predominant HTTP basic authentication protocol (over SSL) makes the common practice of using the same password for multiple accounts remarkably dangerous: an attacker can effectively steal users' passwords for high-security servers by setting up a malicious server or breaking into a low-security server. To defeat this type of malicious server attack, we propose the Single Password Protocol (SPP). SPP employs two basic techniques: challenge/response and one-time (server-specific) tickets.

SPP has two important features. One is that SPP does not allow a server to know a client's password at any time. An attacker cannot steal a client's password by compromising a server. Therefore, SPP allows a user to securely use one single password across multiple servers. The other feature is that SPP prevents phishing attacks.

The following four properties of SPP makes it a promising candidate for client authentication across the Internet:

5.1. Simple

SPP only involves three messages: first, client C sends to server S his user name C; second, S sends to C the stored challenge; third, C sends to S a one-time server-specific ticket, together with a new challenge and new ticket verification information. This simple structure makes SPP easy to understand and easy to implement.

5.2. Secure

The security of SPP is based on two reasonable assumptions. First, SPP is layered on top of SSL. This assumption is reasonable because SSL has become the industry standard for securing communication over the Internet and has been built into all major web browsers, web servers, email clients, email servers, etc. Second, the single password that a client uses to protect all of his accounts is a string of eight (or more) random characters, which is not guessable. This assumption is reasonable because using SPP, a user needs to remember only one password for all of his accounts. Under these two assumptions, SPP is secure against a long list of attacks: eavesdropping attacks, message replay attacks, message loss attacks, message modification attacks, message insertion attacks, brute force attacks, online dictionary attacks, off-line dictionary attacks, password file compromise attacks, message log compromise attacks, malicious server attacks, server spoofing attacks, and even phishing attacks. Detailed analysis of the security properties of SPP is available in Section 3.

5.3. Efficient

SPP only involves one type of computation, hashing (i.e., computing message digest), which requires a negligible amount of processing time. In addition, it only involves four executions of this hashing computation: one execution on the server side and three executions on the client side. SPP itself does not involve any sort of encryption/decryption operations, although the messages in SPP are encrypted by the underlying SSL protocol. This property benefits SPP in both performance and availability. In terms of performance, computing a message digest requires orders of magnitude less processing time than performing an encryption/decryption. In terms of availability, if a password protocol stores some encrypted data using some encryption algorithm on the server side, then a client cannot login on his accounts on a machine that the corresponding decryption algorithm is not available.

5.4. User-friendly

Using SPP, a user only needs to remember one password, and this password can be used for all of his accounts. In addition, this password can be changed easily at the user's will during each login without the notice of the server.

Acknowledgements

The authors would like to thank Reza Curtmola for his valuable comments on the preliminary version of this paper. The authors also would like to thank the editor and the anonymous referees for their constructive comments and valuable suggestions in improving the presentation of this paper.

References

[1] A. Adams, M.A. Sasse, User are not the enemy, Communications of the ACM 42 (12) (1999) 40–46.
[2] Anti-Phishing Working Group. http://www.antiphishing.org/. Accessed: January 30, 2005.
[3] S.M. Bellovin, M. Merritt, Encrypted key exchange: password-based protocols secure against dictionary attacks, in: Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy, 1992, pp. 72–84.

[4] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, Http authentication: Basic and digest access authentication, RFC 2617, 1999.
[5] A.O. Freier, P. Karlton, P.C. Kocher, The ssl protocol version 3.0 internet draft, March 1996. http://wp.netscape.com/eng/ssl3/draft302.txt.
[6] K. Fu, E. Sit, K. Smith, N. Feamster, Dos and don'ts of client authentication on the web, in: Proceedings of the 10th USENIX Security Symposium, August 2001.
[7] N. Haller, The s/key one-time password system, RFC 1760, 1995.
[8] N. Haller, C. Metz, A one-time password system, RFC 1938, 1996.
[9] N.M. Haller, The S/KEY one-time password system, in: Proceedings of the Symposium on Network and Distributed System Security, 1994, pp. 151–157.
[10] R. Kanaley, Login error trouble keeping track of all your sign-ons? Here's a place to keep your electronic keys, but you'd better remember the password, San Jose Mercury News, February 4, 2001.
[11] D.P. Kormann, A.D. Rubin, Risks of the passport single signon protocol, Computer Networks 33 (2000) 51–58.
[12] L. Lamport, Password authentication with insecure communication, Communications of the ACM 24 (11) (1981) 770–771.
[13] P.D. McDaniel, Handbook of Information Security, chapter Computer and Network Authentication, John Wiley and Sons Inc., 2004.
[14] D.L. McDonald, R.J. Atkinson, C. Metz, One time passwords in everything (opie): experience with building and using stronger authentication, in: Proceedings of the 5th USENIX UNIX Security Symposium, 1995.
[15] Microsoft Passport, http://www.passport.net/. Accessed: November 18, 2004.
[16] R. Rivest, The md5 message-digest algorithm, RFC 1321, 1992.
[17] A.D. Rubin, Independent one-time passwords, in: Proceedings of the 5th USENIX Security Symposium, 1995, pp. 167–175.
[18] US National Institute of Science and Technology. Secure hash standard. Federal Information Processing Standard (FIPS) 180-1, 1993.
[19] T. Wu, The secure remote password protocol, in: Proceedings of the Internet Society Symposium on Network and Distributed System Security, March 1998, pp. 97–111.

Mohamed G. Gouda was born in Egypt. His first B.Sc. was in Engineering and his second was in Mathematics; both are from Cairo University. Later, he obtained M.A. in Mathematics from York University and Masters and Ph.D. in Computer Science from the University of Waterloo. He worked for the Honeywell Corporate Technology Center in Minneapolis 1977–1980. In 1980, he joined the University of Texas at Austin where he currently holds the Mike A. Myers Centennial Professorship in Computer Sciences. He spent one summer at Bell labs in Murray Hill, one summer at MCC in Austin, and one winter at the Eindhoven Technical University in the Netherlands.

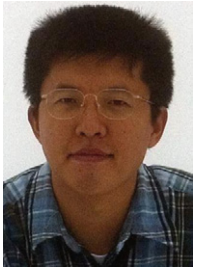
His research areas are distributed and concurrent computing and network protocols. In these areas, he has been working on abstraction, formality, correctness, nondeterminism, atomicity, reliability, security, convergence, and stabilization. He has published over 60 journal papers, and over 80 conference and workshop papers. He has supervised 19 Ph.D. dissertations.

He was the founding Editor-in-Chief of the Springer-Verlag journal Distributed Computing 1985–1989. He served on the editorial board of Information Sciences 1996–1999, and he is currently on the editorial boards of Distributed Computing and the Journal of High Speed Networks.

He was the program committee chairman of ACM SIGCOMM Symposium in 1989. He was the first program committee chairman of IEEE International Conference on Network Protocols in 1993. He was the first program committee chairman of IEEE Symposium on Advances in Computers and Communications, which was held in Egypt in 1995. He was the program committee chairman of IEEE International Conference on Distributed Computing Systems in 1999. He is on the steering committee of the IEEE International Conference on Network Protocols and on the steering committee of the Symposium on Self-Stabilizing Systems, and was a member of the Austin Tuesday Afternoon Club from 1984 till 2001.

He is the author of the textbook "Elements of Network Protocol Design", published by John-Wiley & Sons in 1998. This is the first ever textbook where network protocols are presented in an abstract and formal setting. He also coauthored, with Tommy M. McGuire, the monograph "The Austin Protocol Compiler", published by Springer in 2005.

He is the 1993 winner of the Kuwait Award in Basic Sciences. He was the recipient of an IBM Faculty Partnership Award for the academic year 2000–2001 and again for the academic year 2001–2002 and became a Fellow of the IBM Center for Advanced Studies in Austin in 2002. He won the 2001 IEEE Communication Society William R. Bennet Best Paper Award for his paper "Secure Group Communications Using Key Graphs", coauthored with C. K. Wong and S. S. Lam and published in the February 2000 issue of the IEEE/ACM Transactions on Networking (Volume 8, Number 1, Pages 16–30). In 2004, his paper "Diverse Firewall Design", coauthored with Alex X. Liu and published in the proceedings of the International Conference on Dependable Systems and Networks, won the William C. Carter award.



Alex X. Liu received his Ph.D. degree in computer science from the University of Texas at Austin in 2006. He is currently an assistant professor in the Department of Computer Science and Engineering of Michigan State University. He won the 2004 IEEE& IFIP William C. Carter Award, the 2004 National Outstanding Overseas Students Award sponsored by the Ministry of Education of China, the 2005 George H. Mitchell Award for Excellence in Graduate Research in the University of Texas at Austin, and the 2005 James C.Browne Outstanding Graduate Student Fellowship in the University of Texas at Austin. His research interests include computer and network security, dependable and high-assurance computing, applied cryptography, computer networks, operating systems, and distributed computing.

Please cite this article in press as: M.G. Gouda et al., SPP: An anti-phishing single password protocol, Comput. Netw. (2007), doi:10.1016/j.comnet.2007.03.007