A Password Stretching Method using User Specific Salts

Changhee Lee
TSLab
INITECH
Seoul, Korea
changhee.lee@initech.com

Heejo Lee
Dept. of Computer Science and Engineering
Korea University
Seoul, Korea
heejo@korea.ac.kr

ABSTRACT
In this paper, we present a password stretching method using user specific salts. Our scheme takes similar time to stretch a password as recent password stretching algorithms, but the complexity of a pre-computation attack increases by 10^8 times and the storage required to store the pre-computation result increases by 10^8 times.

Categories and Subject Descriptors
K.6.5 [**Security and Protection**]: Authentication

General Terms
Security

Keywords
Phishing, Password stretching, user-specific salting.

1. INTRODUCTION
Generally, people use the same password on many web sites because of the difficultly in remembering different passwords. As a result, once an attacker obtains a password from one faked site or site with weak security, he can use it to enter other sites. This type of attack is known as phishing. The password stretching method is a way to create a strong password from a weak password. It can be used to create a strong site-specific password from a weak password, using a site-specific salt (e.g. domain name).

2. RELATED WORKS

2.1 PASSWORD STRETCHING METHOD
The password stretching algorithm is defined as

$$K_{long} = F(K_{short}, Salt),$$

where K_{short} is a weak password, K_{long} is a strong password, *Salt* is a variable, and F is a password stretching function [1]. Salt can be a system-specific, time-specific or message-specific variable and F() can be a hash based or block cipher based function.

Ross *et al.* proposed an algorithm that uses a domain name as a salt and HMAC as F() [2]. Their algorithm is defined as

$K_{long} = HMAC(K_{short}, dom)$, where dom is the domain name.

Halderman *et al*. proposed a method, which needs more time to generate a strong password than Ross et al. [4], defined as

$$V = f^{k1}(salt1, K_{short}),\ K_{long} = f^{k2}(K_{short}, salt2, V),$$

where $f^k()$ is to repeat k times f(), *salt1* is user name, and *salt2* is domain name.

2.2 PRE-COMPUTATION ATTACK
The traditional attack method of password stretching algorithm is the brute force attack, which attempts all possible weak passwords one by one. As a result, it is a very time consuming way to retrieve a password. In [3], Oechslin proposed a way of pre-computation attack which is able to crack all possible MS-windows password hashes in 13.6s.

3. PROBLEM AND GOAL
In previous password stretching methods, username and domain name are known variables, therefore unknown variable is only a weak password. An attacker can obtain K_{long} from a faked site or a site with weak security and get a weak password from K_{long} using a rainbow table, pre-computation results of all possible weak passwords.

In many sites, the user uses a 6-8 alphanumeric character password. If a user stretches the password using Ross's method, an attacker can obtain the original password within 13.6s, via a pre-computation attack. In the case of Halderman's method, the password stretching time takes (k1+k2) times longer than Ross's, and consequently an attacker needs more time to generate a rainbow table. However, after generating rainbow table, an attacker can retrieve the original weak password within a similar time period as Ross's.

Of course, we can change the parameter, (k1+k2), so that it has a very long computation time, but password stretching also takes a very long time. This makes the algorithm unusable. We need to find an algorithm which has a similar computation time as Halderman's, but has strong resistance to pre-computation attacks. This is the goal of this paper.

4. PROPOSED METHOD
In this paper, we propose a method to stretch a password using a user-specific salt, to prevent pre-computation attacks using a rainbow table. It has two steps as shown in Figure 1, the first step is to generate a user-specific salt via challenge-response with the user, and the second step is to stretch the password using a user-specific salt. A user-specific salt can be secret numeric data from a user, such as a credit card number or debit card number.

We changed the k1 parameter of Halderman's algorithm into a user-specific salt, which enabled the proposed algorithm to be executed differently for each user. It is defined as

$$V = f^{salt}(K_{short}, username),\ K_{long} = f^{k}(K_{short}, dom, V).$$

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005

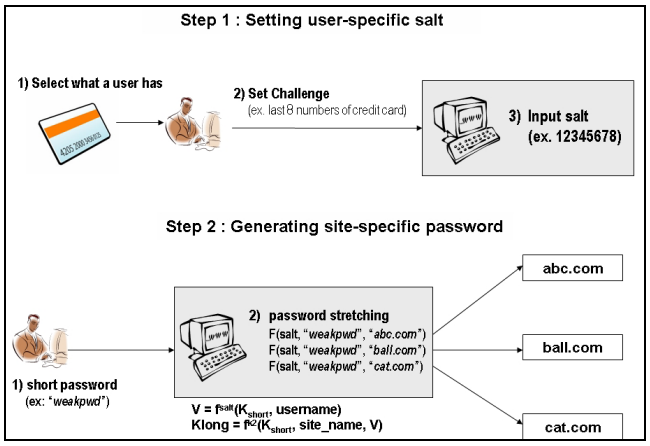

Figure 1. Proposed password stretching method

Because the user-specific salt differs for each user, the algorithm has a different parameter for each user. Therefore an attacker has to generate a rainbow table for each possible salt, store it and use it to look up values. In [3], we need 1.4GB to store the rainbow table for a 7 characters password. If the salt is 8 numeric characters (eg. the last 8 characters of a credit card number), the possible number of salts is 10^8 and an attacker needs $10^8 \times$ 1.4GB, about 0.1 million TB, to store the rainbow table for all possible salts. It takes about $10^8 \times$ 10s, or 31 years, to find the password from all rainbow tables. The greater the number of characters of the salt, the greater the storage and table lookup time required. We compare Ross's algorithm, Halderman's algorithm and our proposed algorithm in Table1.

Table 1. Resistance to a pre-computation attack (7 alphanumeric characters password and 8 numeric characters salts)

Algorithm	Stretching Time	Rainbow Table Size	Attack Time
Ross	10^{-6}s	1.4G	≤ 10s
Halderman	0.1s	1.4G	≤ 10s
ours	0.1s	$10^8 \times$ 1.4G	31 years

5. IMPLEMENTATION

We implemented our method as a Firefox extension by modifying passwordMaker [5]. PasswordMaker is implemented in JavaScript, to allow execution on any platform, and supports the multiple password stretching algorithm and multiple user accounts. Our program is available at:

http://www.gbtn.org/~chlee/research/passwdmaker-cr-1.6.xpi .

A user has to set a challenge and response for password stretching. We use the response as the salt. Our program caches the value V so that the user does not need to enter a salt each time. We set the k value such that K_{long} generation takes about 0.1s. There are three pre-defined challenges in our program, the last 8 numbers of a credit card number, 8 numbers selected from an ID card, and 8 numbers selected from an item owned by the user.

After the user sets answers to the challenge, and master password, the program generates a V and caches it to disk. Then, the program stretches the master password using the domain name from the address bar of the browser. From this point on, a user needs to input only the master password, because the program will use a cached V. In addition, if a user stores the master password on disk in encrypted format, there is nothing required to input to generate a strong password. Figure 2 shows a screenshot of our program.

6. CONCLUSTION AND FUTURE WORK

In this paper, we discuss a password stretching algorithm that creates a strong password from a weak password and provides protection against a pre-computation attack. Using a pre-computation attack, a password stretched using Ross's or Halderman's algorithm can be revealed within about 10s.

We propose a novel password stretching algorithm that operates with a user-specific salt. It takes similar time to stretch a password as Halderman's algorithm, but an attacker requires 10^8 times more rainbow tables than Halderman's and 10^8 times longer to obtain the original weak password using a rainbow table. This result makes a pre-computation attack infeasible.

In the future, we will increase our scheme's resistance against malware and enable it to be used with Microsoft Internet Explorer (MSIE).

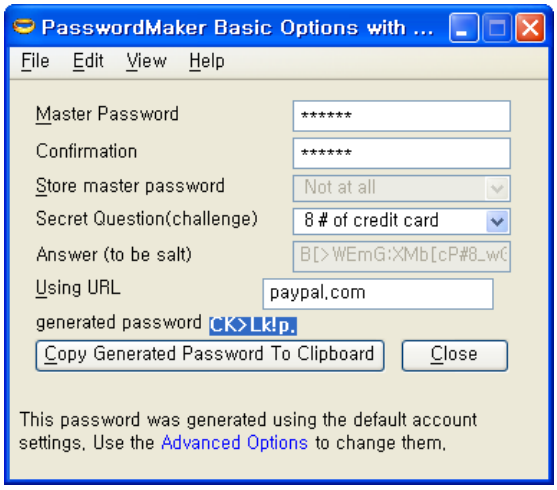

Figure 2. Program Screenshot

7. REFERENCES

[1] J. Kelsey, B. Schneier, C. Hall, and D. Wagner. Secure applications of low-entropy keys. Lecture Notes in Computer Science, 1396:121–134, 1998.

[2] Blake Ross, Collin Jackson, Nicholas Miyake, Dan Boneh, and John C. Mitchell. Stronger Password Authentication Using Browser Extensions, Proeedings of the 14th Usenix Security Symposium, 2005

[3] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off, Proceedings of Crypto'03

[4] J.A.Halderman, B.Waters, and E.Felten. A Convenient method for securely managing passwords. Proceedings of the 14th International World Wide Web Conference (WWW 2005), 2005

[5] LEAHSCAPE, Inc. passwordMaker, http://passwordmaker.org/, 2006