

Protecting a Multiuser Web Application against Online Password-Guessing Attacks

Francisco Corella

June 2007

Patent Pending

Abstract

This white paper presents a method for protecting a Web application against online password-guessing attacks. A user logs in with three credentials: the name of the application instance, a user ID, and a password, where the instance name is a secret known only to the instance users, the user ID is a secret known only to the instance administrators, and the password is a secret known only to the user. After five consecutive bad guesses against a password, the user is locked out and the password must be reset; furthermore, after thirty bad guesses (not including five consecutive ones), the user is forced to change her password. A denial-of-service attack that repeatedly locks out one user is thwarted by changing the user ID of the victim, while such an attack against multiple users is thwarted by changing the instance name. Changing the instance name can also be used to preempt attacks by ex-users. This method, together with a technique for securely resetting a password, has been used in the Pomcor file-repository application to provide the convenience of password-based authentication without compromising security.

1 Introduction

Passwords have a bad reputation among security professionals, but they are nevertheless the most common means of user authentication on the Web. There are alternatives to passwords that are deemed more secure, such as hardware tokens that generate time-dependent passcodes [7], or SSL/TLS client certificates [4, 8]. These alternatives, however, require the deployment and maintenance of cumbersome infrastructures. Tokens have to be distributed to users and sometimes resynchronized. Client certificates have to be signed by a certificate authority (CA), and certificate revocation lists have to be maintained. No alternative can compete with passwords in terms of convenience, especially for short-lived on-demand applications.

In this white paper we examine the security challenges that arise from the use of passwords in the context of a Web application. We show that they are substantially different from those that arise in other computing contexts. Two-way authentication, online password-guessing, user lockout, and password reset are the main concerns, while offline dictionary attacks are less of a threat than in traditional computer systems.

We show that the security challenges that arise in the Web context can be met by a combination of well-known security techniques and a few novel security features. Novel security features for protection against online password-guessing and user lockout are presented in this white paper, while a novel method for secure password reset is presented in a companion white paper [3]. We conclude that it is possible to enjoy the convenience and ease-of-use of password-based authentication without compromising security.

2 Password security and the Web

It is important to recognize that passwords present different security challenges in different computing environments. Securing login access to a local timesharing machine or a file server, for example, is a different problem than securing remote access to a Web application.

One challenge of securing local login access is protecting the passwords, or hashes of passwords, stored in the local machine. Early Unix systems used a password file containing salted hashes of passwords, which was world-readable [6, 5]. This flaw made it trivial to mount offline dictionary attacks against the passwords, and greatly contributed to the bad reputation of password security.¹

Modern operating systems, whether derived from Unix or not, do not expose the password file to public scrutiny. Still, the password file, or some equivalent data structure containing hashes or salted hashes of passwords, often resides on the same machine to which users have login access, and sometimes physical access. A user who manages to acquire administrative privileges may thus gain access to the data structure, and mount a dictionary attack. Worse, administrators usually have entirely legal access to the data structure, and a rogue administrator may easily mount a dictionary attack to obtain user passwords.

Protecting the data structure containing the passwords or their hashes is easier in the context of a Web application. Users access the application through a Web server. They do not have physical access to the Web server, and they do not have user accounts with the operating system of the Web server; they can only interact with the Web server through the HTTP protocol. Furthermore, the password data structure may reside in a database server or other back-end server rather than in a front-end Web server. Both servers may be protected by firewalls. There are thus many more barriers between a user and the password data structure.

Moreover, in the case of a user-administered application, such as the Pomcor file-repository application, administrators perform their administrative duties through a Web interface, an extension of the Web interface through which all users access the application. Thus administrators do not have access to the password data structure. Indeed, they are separated from the password data structure by as many barriers as any other user.

On the other hand, passwords face new security challenges in the context of a Web application. An obvious one is the need to carry the password from the user's browser to a Web server duly authenticated as belonging to the application front-end.

To address this challenge, cryptographers have designed protocols that combine password-based authentication and cryptographic key exchange over an insecure network such as the Internet [1, 10]. But there is a simpler solution, viz. to use SSL. Today, SSL (a.k.a. TLS [4]) is well established as the protocol of choice for key exchange and encrypted communication on the Web. SSL is universally supported by browsers and servers, and it is deemed to provide effective security [9], if used properly [2]. It is thus easiest to use SSL to exchange keys, authenticate the server (using a public key certificate signed by a CA known to the browser), and establish an encrypted connection. Once the connection is established, it can be used by the browser to send the user's password securely to the server.

Another challenge is how to reset a password securely. This problem is discussed, and a new solution is proposed, in a companion white paper [3].

¹In an *offline dictionary attack* the attacker is in possession of hashes of current passwords and uses a computer program to generate a large number of password candidates, some of them derived from words found in dictionaries. The attacker computes the hashes of the password candidates and compares them to the hashes known to the attacker. (A *salted hash* is a hash of a password combined with a random value referred to as *salt*; if hashes are salted, the attacker must compute the hash of each candidate password combined with each salt.) In an *online password-guessing attack*, by contrast, the attacker does not have access to hashes or salted hashes of passwords, and does not compute any such hashes. The attacker simply tries raw credentials by attempting to log in with them.

A third challenge is how to defend against online password-guessing attacks, including password capture on a different site where the user uses the same password, and denial-of-service by user lockout. This is the focus of this white paper.

3 Online password-guessing attacks over the Internet

Online password-guessing attacks present a greater challenge on the Web environment for several reasons:

1. A Web application (or a Web site) is accessible over the Internet from anywhere in the world. This affords a high degree of anonymity and impunity to an attacker.
2. If the application takes no precautions, passwords can be submitted by a computer program running on a client machine at a rate of thousands per second. Moreover, armies of thousands of bots (hijacked computers) are readily available to an attacker for a massively parallel attack. An attacker may thus be able to try millions of passwords per second in the absence of countermeasures.
3. Web applications are vulnerable to password capture on a different site. Many users, faced with the problem of having to remember passwords for a large number of Web sites and Web applications, simply use the same password over and over again. An attacker can exploit this by setting up a malicious site with the purpose of collecting user credentials, which the attacker can then use to log in to a target application.

Most Web applications ask for a user ID and a password as login credentials, and protect themselves against online password guessing locking out the user account identified by a given user ID after a small number of consecutive login attempts with that user ID but an incorrect password. The password must then be reset, or a timeout period must elapse, before logins to the account are permitted again.

This countermeasure, however, has serious drawbacks. First, it can turn an online password-guessing attack into a denial-of-service attack, by repeatedly locking out the user. The denial of service may be a side effect of the guessing attack, or it may be an intentional attack that exploits the countermeasure.

Secondly, although it limits the number of consecutive bad guesses, it allows an unlimited number of bad guesses, as long as they are interleaved with valid logins by the legitimate user. An attacker who is able to watch the user, or who can otherwise guess the time-pattern of logins, may be able to time the password guesses so as to avoid locking out the user.

An application that relies on this countermeasure is particularly vulnerable to an attack by an ex-user. An ex-user may know the user IDs of all or most current users, and may launch a denial-of-service attack against all of them at once. Alternatively, the ex-user may know the time-pattern of logins of many current users, and may be able to mount a sustained guessing attack against their passwords without locking them out; furthermore the chances of success are increased if each password can be tried out against multiple user accounts.

Finally, the countermeasure does nothing to protect the application against password capture on a different site.

4 Proposed defense against online password guessing

Our method of protection against online password-guessing attacks and related denial-of-service attacks, implemented in the Pomcor repository application, is a combination of the following application features and countermeasures.

1. The application is user-administered. Each application instance (i.e. each repository, in the specific case of the Pomcor repository application) has an administrative hierarchy consisting of the Pomcor customer who creates the instance, referred to as the *owner*, and, optionally, users to whom the owner grants administrative privileges. The owner and the users granted administrative privileges are referred to as *administrators*. Only the owner registers with the application provider (Pomcor); other user accounts are created by administrators using a Web interface. The administrative hierarchy, which is also useful in providing a secure password procedure, is described in more detail in [3].
2. Each user logs in with three credentials rather than the usual two:
 - (a) The application instance name (the repository name, in the case of Pomcor repository application), which is considered a secret shared by the users of the application instance. The instance name can be changed by the owner.
 - (b) A user ID, which is known only to the user and the administrators. The user ID is chosen by the administrator who creates the user account, and can be changed by an administrator (by any administrator if the user has no administrative privileges, by the owner if the user is herself an administrator).
 - (c) A password, known only to the user.
3. After a certain number of consecutive bad guesses against a password (5 in the case of the Pomcor application), the user is locked out, i.e. login is disabled for the user's account. Bad guesses are considered to be consecutive if there is no intervening successfully completed login to the user's account. All the consecutive bad guesses must be against the same password; counting starts over if the password is changed.
4. A user who has been locked out is allowed to log in again once her password has been reset. An administrator can reset the password of an unprivileged user, by assigning a temporary password, known to the administrator, to the user's account; the user must change the temporary password to a permanent one only to her when she logs in. The companion white paper [3] describes a secure password reset procedure that protects the transmission of the temporary password from the administrator to the user. The owner can reset the password of an administrator, and can use a rescue code, as described in [3], to reset her own password.
5. After the total number of invalid login attempts against a password (not necessarily consecutive ones) reaches a certain threshold (30 in the case of the Pomcor application), the user is asked to change her password the next time she logs in; she cannot further use her account until she changes her password. Furthermore, any logins with a valid password made after the threshold has been reached do not count as "successfully completed", for the above definition of what it means for bad guesses to be consecutive. (The login is not "successfully completed" until the user changes her password, at which time the counting of bad guesses starts over.)
6. When the user changes her password, she is not allowed to select as the new password a password that has previously been used as a permanent or temporary password on her user account. (On the other hand, there is no such constraint when an administrator resets the user's password; otherwise the administrator could exploit the rejection of previously-used passwords to mount an online password-guessing attack with the purpose of discovering permanent passwords previously chosen by the user; although those passwords cannot be reused for the application, they should only be known to the user, and they must be considered confidential in case the user makes use of them for other purposes. The

Pomcor repository application provides a random password generation facility that the administrator can use to produce a high entropy temporary password that, with very high probability, has not been used previously.)

5 Effectiveness

The following considerations show the effectiveness of the defense against online password-guessing attacks:

- There is an overall limit on the number of guesses that an attacker can make against a permanent password, equal to the sum of the limit on consecutive bad guesses and the threshold on the total number of bad guesses. (The overall limit is $5 + 30 = 35$ for the Pomcor application.) Indeed, after the threshold (30) is crossed, logins by the legitimate user, if any, do not reset the counter of consecutive bad guesses; hence the attacker can make at most a number of guesses equal to the limit of consecutive bad guesses (5) before the logins are disabled for the account or the legitimate user changes the password. The same password cannot be reused as a permanent password. (It can be reused as a temporary password, but that would be an improbable coincidence, since a permanent password for a user account is chosen by the user, whereas a temporary password is chosen by an administrator other than the user.)
- If a user is repeatedly locked out by an attacker, an administrator can change the user ID for that user to thwart the attack.
- If an attacker knows the user IDs of multiple users (e.g. because the attacker is an ex-user) and repeatedly locks them out, the owner of the application instance can change the name of the instance to thwart the attack.
- If an attacker by an ex-user is feared, the owner can preempt it by changing the name of the application instance.
- A rogue Web site set up by an attacker with the purpose of capturing passwords may well obtain a user's password, and may even obtain the user's user ID for the application, but will not obtain the application instance name. Hence the credentials obtained by the rogue site are not sufficient to log in to the application.

Furthermore, the user ID is chosen and controlled by the administrative hierarchy rather than by the user; this makes it less likely that the user ID will be the same that user chooses herself for other Web applications and Web sites, and thus less likely that it will be captured by the rogue site.²

The mere use of the application instance name as an additional credential increases the entropy of the set of credentials, i.e. makes it more difficult to guess them. Of course, the application must ask for all three credentials at once, and must give the same response if no application instance is found with the instance name supplied as the first credential, if no user is found with the user ID entered as second credential, or if the password is incorrect. The Pomcor application goes further by introducing delays to ensure that timing cannot be used to distinguish between the three possible causes of login failure.

²The above assumes that the rogue site is not trying to masquerade as the Pomcor application, as would be the case, for example, in a phishing attack. For protection against such masquerading, the Pomcor repository application relies on proper use of SSL/TLS [2], coupled with user education.

6 Password Databases and Automated Password Entry

This section shows that the proposed defense against online password guessing meshes well with a popular solution to the password proliferation problem.

We discussed above how users are burdened by the need to remember passwords for many Web sites and Web applications, how they may be tempted to use the same password for multiple sites or applications, and how that password is then vulnerable to capture by a rogue site.

A solution to the password proliferation problem that is currently becoming popular is to keep Web passwords stored in an encrypted database instead of remembering them. Several variations on this idea have been proposed. The database may be managed by the browser, or by the operating system, or by a separate application. It may physically reside on the client machine, on a pluggable memory device, on a PDA, or on a different computer reachable over the Internet. Access to the database may be protected by a master password or a biometric sensor or both.

Use of a password database introduces the risk that it may be compromised, causing all passwords to be disclosed at once to an attacker. However, if properly designed and used, a password database may be a good solution to the password proliferation problem.

In some variations of the password database solution, passwords are supplied automatically when needed, without the user having to enter them manually. If the user does not type the password, the user cannot mistype it. Therefore the user of an application that implements the proposed defense against online password guessing will never be forced to change her password if she uses a password database that provides automated password entry.

7 Comparison with Password Aging

It is worth comparing the technique for limiting the number of significant negative responses that we have just described with the practice of forcing users to change their passwords after a certain period of time, called *password aging*.

Password aging has problems, some of which were pointed out long ago by Grampp and Morris [5]. Grampp and Morris asserted then that "the aging of passwords is a difficult problem, yet unsolved".

Our technique can be viewed as a form of password aging, where the aging is caused, not by the mere passage of time, but by the accumulation of invalid login attempts that bring an attacker closer to guessing the password by eliminating alternatives. The concept is similar to the "aging" of an encryption key as it produces ciphertext that may be used as input to cryptanalysis.

One problem with traditional password aging is that it is an inconvenience for users. Our technique is also an inconvenience for users. We hope, however, that it will prove less annoying, because it is better motivated, and because the user has some control over the frequency of password changes. Indeed, by being careful when entering her password, or by using an encrypted password database that supplies the password automatically as discussed above, the user may be able to keep a password indefinitely.

A secondary benefit of our technique is that the user is motivated to pay attention to the count of bad guesses, which is shown to the user after login. The user is thus more likely to discover extraneous login attempts indicating a password-guessing attack, and can then thwart the attack by having her user ID changed by an administrator.

A benefit of traditional password aging is that it limits the period of time during which a successful attacker can use a compromised password. Our technique does not address this concern, since a careful user, or a careful attacker who has guessed the user's password, can keep the same password indefinitely.

It would be possible to add a time limit to our scheme. We do not advocate this, however, because the resulting benefit is doubtful. Indeed, an attacker who knows a password will often be able to guess the next password that will be chosen by the user. For example, if the password ends in one or more digits that make up a number, it is likely that the next password will be the result of adding 1 to that number. We do not think it is wise to further inconvenience users for a doubtful security benefit.

8 Conclusion

This white paper has presented a password-based authentication method where the user's credentials include a secret application instance name and a secret user ID in addition to a password, where the user is locked out after a number of consecutive bad guesses, and, furthermore, where the user is forced to change her password after a (typically higher) number of bad guesses with no requirement that those bad guesses be consecutive. This method provides protection against online guessing attacks and related denial-of-service attacks, including attacks by ex-users, and other security benefits. Together with a secure-password-reset technique described in a companion white paper, this technique makes it possible to enjoy the convenience of password-based authentication without compromising security. Both techniques have been implemented in the Pomcor file-repository application.

References

- [1] S.M. Bellare and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attacks. In *IEEE Computer Society Conference on Research in Security and Privacy*, pages 72–84, 1992.
- [2] Francisco Corella. On the Proper Use of SSL to Protect Form Data. http://www.pomcor.com/whitepapers/proper_use_of_ssl.pdf. Pomcor Whitepaper.
- [3] Francisco Corella. Secure Password Reset in a Multiuser Web Application. http://www.pomcor.com/whitepapers/secure_password_reset.pdf. Pomcor Whitepaper.
- [4] T. Dierks and C. Allen. The TLS Protocol Version 1.0. <http://tools.ietf.org/html/rfc2246>, January 1999. IETF RFC 2246.
- [5] F.T. Grampp and R.H. Morris. Unix Operating System Security. *AT&T Bell Laboratories Technical Journal*, 8(63):1649–1672, October 1984.
- [6] R. Morris and K. Thompson. Password Security: A Case History. *Communications of the ACM*, 22(11):594–597, November 1979.
- [7] M. Nystrom. The SecurID(r) SASL Mechanism. <http://tools.ietf.org/html/rfc2808>, April 2000. IETF RFC 2808.
- [8] W. Ford R. Housley, W. Polk and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. <http://tools.ietf.org/html/rfc3280>, April 2002. IETF RFC 3280.
- [9] D. Wagner and B. Schneier. Analysis of the SSL 3.0 Protocol. <http://www.schneier.com/paper-ssl.html>.
- [10] Thomas Wu. The Secure Remote Password Protocol. In *Network and Distributed System Security Symposium*, March 1998.