SECURING SELF-SERVICE PASSWORD RESET FUNCTIONALITY IN WEB APPLICATIONS

David A. Shpritz

July, 2010

INTRODUCTION

Many web applications requiring user authentication also provide self-service password reset functionality to help reduce helpdesk calls and in turn support costs. However, sometimes this functionality can introduce new vulnerabilities into an application, allowing brute forcing, username harvesting email harvesting or information disclosure and in some cases even exposing the password via web page or email.

This document aims to provide web application developers with a "how to" of sorts, providing a process which developers can use to implement password reset functionality into their web applications in a safe and secure manner by incorporating a series of best practices, as well as helping developers avoid pitfalls which have resulted in vulnerabilities in countless web applications.

The first aspect to examine is what information will be required from the user. I will cover information needed from the user during the registration process, as well as the information the user will need to initiate the reset of their password.

A NOTE ON THE RELIANCE ON EMAIL

The insecurity of email is well known and while it is used in this process, it can also be excluded depending on the security posture of the application and organization.

Email is used in this process as described below as a way of confirming the identity of the user as well as for transmission of temporary password reset links, but never the password itself, as the goal is to reset the password, not expose it. As many users will use the same password for multiple web applications, the potential exists that exposure of the existing password, via email or display in a web page could compromise not only the user's account for your application, but other applications as well.

An additional benefit of using the email address currently stored for the user is that it helps form an additional factor for authentication and alerting for the user.

As an alternative SMS messages could also be used to provide a token for the user to verify their identity, if the phone number for the SMS message is already being stored by the application.

Secret Questions

At registration an application can request that a user answer certain questions to which the answer would not be publicly available. As the answers to these questions are behaving as "backup passwords" it is important that the same care be used in selecting the questions as one would take in selecting passwords. Some systems will allow a user to craft their own questions, but in my experience this may not be a good idea as the user may select questions to which the answers are easily found. If the questions are hard coded into the application (I would suggest at least three, however it is important to keep in mind the security posture of the organization and application) then the selection of these questions must be taken into consideration. One option is to supply a number of questions which the user can select a few for use with their account.

The goal is to make the answer to the question very precise, but at the same time make it very hard to guess. As an example, "what is your favorite color", while very personal, does not have very many answers (there are about 100 "common" colors, and most likely the user would answer in the simplest form, so "green", "red", "blue", knocking that range down even more), making it a poor choice for a question. On the other hand a question like "What was the name of your first stuffed animal?" has a great many answers, and is not easily guessed at.

Additionally, the audience for the application should be taken into account, as a question may work for a user in the United States, but may not be answerable to users in Europe or Asia, and as such cultural differences should be taken into account.

Another consideration is that you want to avoid questions which may result in storage of information which may contravene compliance mandates (such as questions relating to Social Security Numbers or Credit Card Numbers) under which your application may fall. In the "Resources" section you will find links to more information on selecting good security questions.

As I mentioned above, these questions and answers are being used as a backup password, so the storage of them should be given the same level of care as the username and passwords for the application.

What does the user need to start the reset process?

In some situations the user can provide a username to start the process, while others will require the email address for the account. In some situations a user may not remember the username they have on file, so an extra step may be added at the beginning to allow the user to enter an email address to have the username on file for the email address sent to them.

A danger here is that this type of mechanism can turn into a method of harvesting email addresses which are valid users of the system. To avoid this do not provide error messages which indicate that an email address has a valid user or does not have a valid user. You should indicate that a message has been sent, and provide the error in the email instead of on the web page. These types of notifications can also help act as an early warning as users may alert you that they have received notifications of attempts to retrieve their username, and as such it may be useful to also

include the IP address used for the request or a request ID which may allow you to track down requests which generated them. Another possible mitigation to this threat is using a CAPTCHA to make automated attacks against the system more difficult, however, CAPTCHA implementations and effectiveness vary. It should also be noted that CAPTCHAs are generally disliked by end users, so usability should be taken into consideration.

Once a user knows the username for an account, the process to reset the password can begin. Notice that this is resetting the password, not providing it back to the user. In most web applications this may not even be possible if a one way hash is being used. A link is provided in the "Resources" section below for the OWASP Cryptographic Storage Cheat Sheet which contains guidance for the storage of passwords.

Answering questions (Starting the password reset process)

Once the user has obtained the username the reset process can begin. The user enters the username and email address on file and then can be presented with one to three of the security questions selected at registration. The questions can be chosen at random from those stored for the user, and upon an unsuccessful attempt to answer them a different set should be presented. Again there is a danger here of brute forcing usernames or username harvesting, so care must be taken to prevent this.

One strategy to prevent username harvesting is to use a "tar pit" of sorts. When an invalid username is entered, the application presents a set of fake security questions. When submitted, the application always indicates that the answers are incorrect. Keep in mind that while this may frustrate an attacker, it may also frustrate valid users, so developers will need to strike a balance between security and usability appropriate for the application.

If multiple questions are used, it is also important to never indicate which answers are correct if any of the answers are wrong, as it allows an attacker to confirm information and drill down to one particular question. After a number of attempts (usually corresponding to the number of attempts used for account lockout functionality) the account should be locked out in similar fashion as failed login attempts.

After the questions have been answered successfully

After a user has succeeded in answering the security questions correctly, a unique link should be sent to the email address on file for the user. This link should only be sent to the email address on file rather than allowing the user to enter a new address to prevent an attacker from hijacking the account. The link should be unique (for example contain a unique token) to prevent the ability of an attacker to randomly guess at password reset URLs which would allow them to reset the passwords for users without going through the processes above. The link should also

expire after a short amount of time, preferably 48 hours or less to prevent someone from using an old link to reset a password. Once used, the link should expire as well to prevent reuse.

There are some drawbacks to this approach, the main being a reliance on email. If this is a concern, the sending of a reset link can be excluded and the user can move on to the process of selecting a new password. As mentioned above, it is possible to use SMS messages to previously known mobile phone numbers as another form of "out-of-band" verification.

SELECTING A NEW PASSWORD

Once the user has received the link and visited the page in the application for resetting the password, the link should be expired to prevent reuse. The password can then be reset by the user, applying the same controls in place for the initial password selection or password change process, ensuring that complexity and length requirements are met.

AFTER THE PASSWORD HAS BEEN RESET

After a new password has been selected a confirmation telling the user that the password has been reset can be displayed. Additionally a confirmation email should also be sent to the email address on file. This confirmation is used as a way to alert the user in the case that it is an attacker attempting to gain access to their account.

After the password has been reset, do not instantly log the user in to the application but instead send them to the standard login page to attempt a login with their new password, providing a check that the user's password has been reset successfully.

RESOURCES

Below are resources which provide more information on designing and attacking password reset functionality in web applications, as well as suggestions for selecting secret questions for use in the process.

Self-service password reset: http://en.wikipedia.org/wiki/Self-service_password_reset

Good Security Questions: http://www.goodsecurityquestions.com/index.htm

How to Do Password Resets Right:
http://www.csoonline.com/article/205900/How_to_Do_Password_Resets_Right?page=1

Insecure Implementations Of Challenge Questions Answers (CQA) For Password Resets:
http://securesoftware.blogspot.com/2009/03/insecure-implementations-of-challenge.html

Best Practices for Your "Forgot Password" Feature:
http://www.fishnetsecurity.com/sites/com.fishnetsecurity/downloads/Forgot_Password_Best_Practices_v2.0.pdf

Testing for Vulnerable Remember Password and Pwd Reset (OWASP-AT-006): http://www.owasp.org/index.php/Testing_for_Vulnerable_Remember_Password_and_Pwd_Reset_%28OWASP-AT-006%29

Authentication Cheat Sheet: http://www.owasp.org/index.php/Authentication_Cheat_Sheet

Parola secreta?: http://www.owasp.org/index.php/Using_Secret_Questions

Insufficient Password Recovery: http://projects.webappsec.org/Insufficient-Password-Recovery

Username Enumeration Vulnerabilities: http://www.gnucitizen.org/blog/username-enumeration-vulnerabilities/

Cryptographic Storage Cheat Sheet: http://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

Detecting Malice: http://www.detectmalice.com/

ACKNOWLEDGEMENTS

The author would like to thank the following for their contributions and review:

Kees Leune

Harry Woodward-Clarke

LICENSE