



FishNet Security White Paper

Best Practices for Your "Forgot Password" Feature

by Dave Ferguson, CISSP, CSSLP
Principal Consultant
FishNet Security



Corporate Headquarters 1710 Walnut Street, Kansas City, MO 64108 www.fishnetsecurity.com

focused INFORMATION ASSURANCE • SECURITY TECHNOLOGY • SECURITY INTEGRATION • 24X7 SUPPORT • TRAINING

Introduction

As an application security consultant, I've had the opportunity to assess a wide variety of web applications. If I were to count the number of applications I've reviewed, there'd probably be the same number of different techniques for helping users with forgotten passwords. It seems everybody has their own idea how it should be done. Some web applications offer no help at all when you've forgotten your password. Some development teams have bolted on the functionality as an afterthought. Others ask you jump through various hoops involving emails, special URLs, temporary passwords, and so on. But which is most secure? In this paper, I hope to describe the most secure practices for a forgot password feature, extracted from my experience in assessing the security of web applications.

Note: The recommendations presented in this paper are most appropriate for organizations that have a business relationship with users. Applications targeting the general public (social networking, free email sites, etc.) are fundamentally different and some concepts presented may not be feasible in those situations.

A Tricky Proposition

For a web application, authentication is typically done via username and password inputs from the user. The authentication process often receives plenty of attention during the design and development phases, and much has been written about how to secure the process. Enforce strong user passwords. Display a generic error message after a failed login. Disable the account after a certain number of login attempts. Store passwords in the database as salted hash values. All of these carefully thought-out security measures often go out the window when a user forgets his password.

The expectation as a user is that an application will assist you when you've forgotten your password. But there's something fundamentally insecure about a forgot password feature. The application is essentially providing a second, entirely separate procedure to authenticate users. It's this secondary method of authentication that often gets short shrift by designers and developers. Security weaknesses creep in. Let's explore some existing techniques being used.

Review of Existing Techniques

Below are some techniques web applications currently use to help users with forgotten passwords.

- Email the current password
- Email a temporary password
- Email instructions on how to reset the password
- Display the current password after answering questions
- Provide a password hint or password reminder

There are other techniques, but the above five are probably the most common. Let's review these in more detail. For the sake of this exercise, let's say that a user named "Joe" can't remember his password.

Weak Technique A – Email the current password

With this technique, Joe enters either his username or email address. The application looks up the email address for the username provided (or verifies that the email address is valid) and proceeds to send an email to Joe with his current password. This is one of the simplest and most common techniques used by web applications, but it is also one of the most insecure. The fundamental weakness here is that email is not a secure form of communication. Sensitive data is transmitted unencrypted over the Internet, meaning that someone who is sniffing the network

traffic could steal Joe's password. The same could happen if another person has access to Joe's emails or has compromised his email account.

Some applications actually make the situation worse by including Joe's username in the email. Another concern is the storage of Joe's password. Since the current password is retrievable such that it can be sent via email, it's obviously stored in the database either unencrypted or encrypted with a reversible, symmetric algorithm. Joe's password is not being stored as a one-way hash value, which is a standard best practice to help keep passwords safe.

An example of Technique A from the real world is presented below. In this case, the application makes a poor decision to send both the username (called "handle") and the password in the same email.

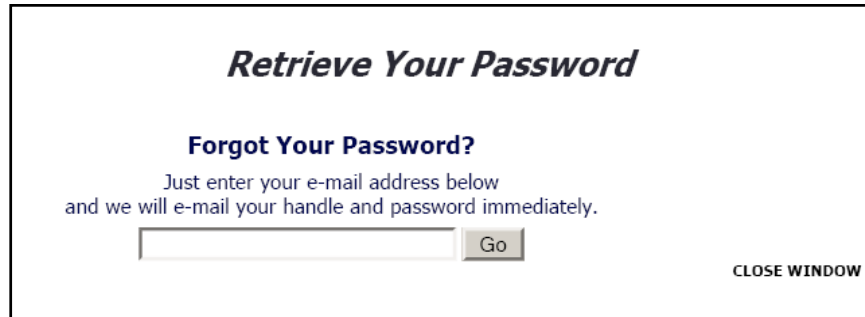


Figure A1 – You're asked to enter your email address

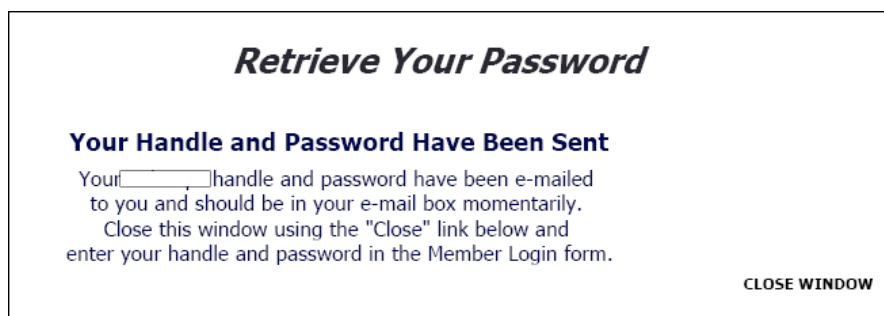


Figure A2 – A message indicates that an email has been sent to you

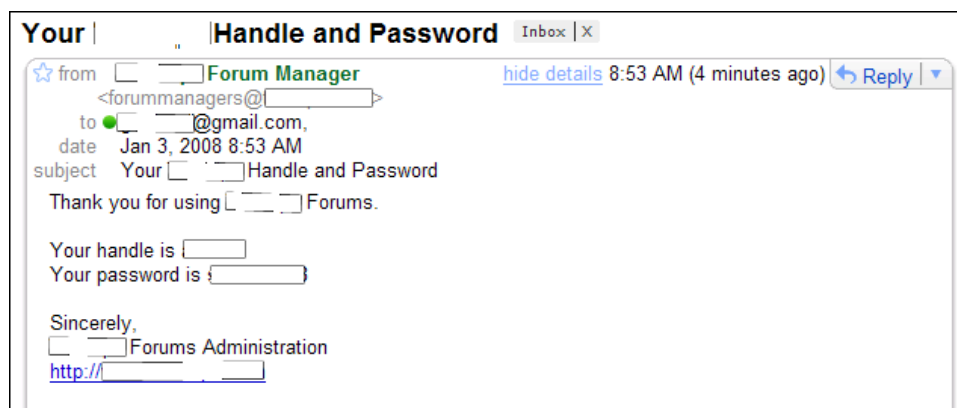


Figure A3 – The email contains your current password (and username too in this case)

Weak Technique B - Email a temporary password

This technique is often used when the application is correctly storing user passwords as one-way hash values. That is good, but things go downhill from there. Joe in this case is asked to enter either his username or email address. The application proceeds to reset Joe's password to a temporary value and sends an email containing his temporary password. An application

may recognize the temporary password only for a few hours or days. Sometimes there is no time limit. Regardless, Joe is forced to change his password after logging in with the temporary password.

These steps make the whole process of Technique B *seem* secure, but is it? Not really. The problem here again is that a password is sent via email. A malicious person sniffing network traffic could compromise the account. Creating a temporary password and forcing Joe to change it upon login are not effective countermeasures against network sniffing or someone who has compromised Joe's email account.

An example of Technique B taken from the real world is presented below.

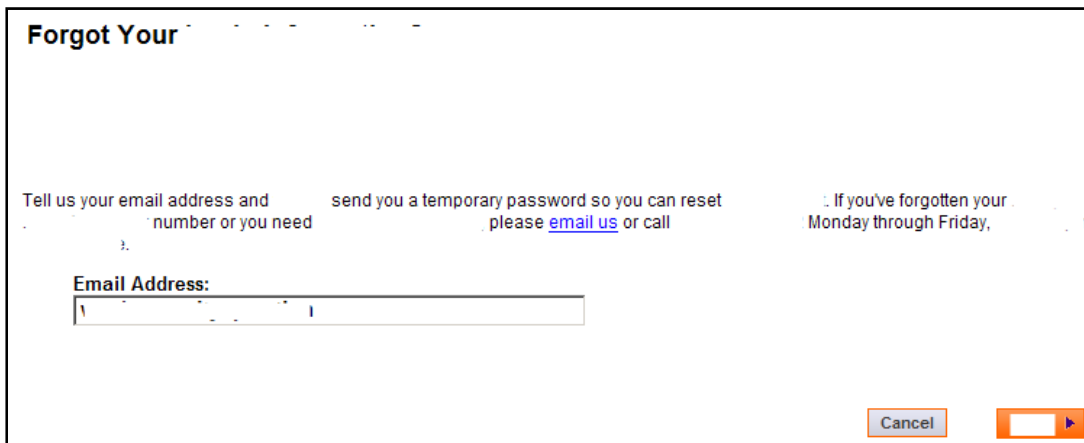


Figure B1 – You're asked to enter your email address

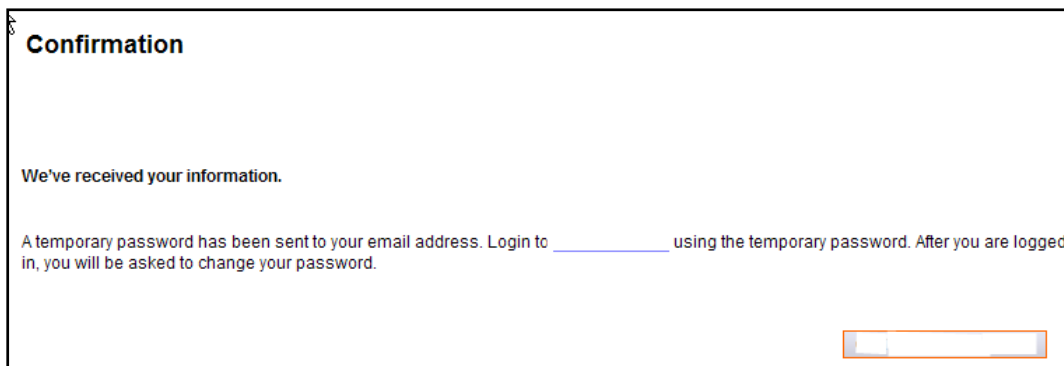


Figure B2 – A message indicates that an email has been sent to you

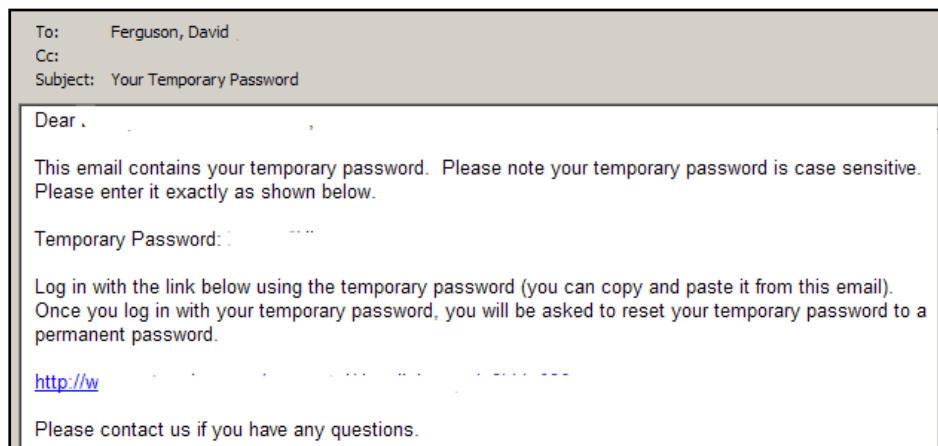


Figure B3 – The email provides you with a temporary password




Figure B4 – After logging in with the temporary password, you immediately must change your password

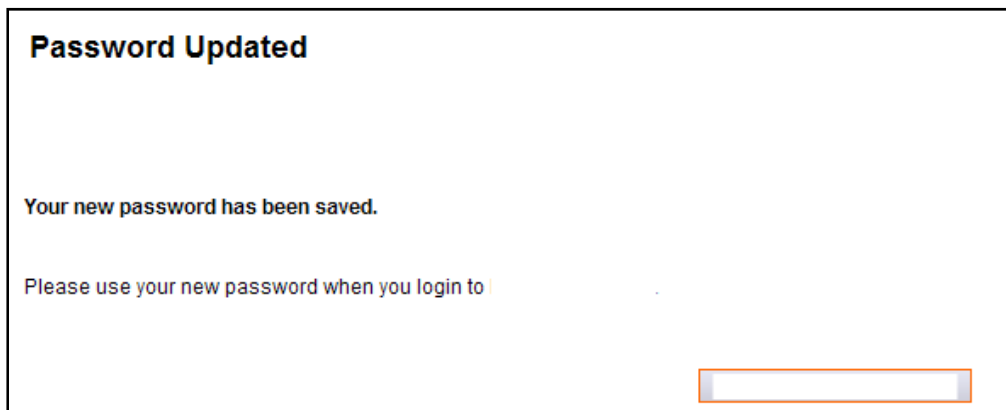


Figure B5 – A message indicates that your new password is in effect

Weak Technique C – Email instructions on how to reset the password

This technique is similar to B in that Joe is asked to enter either his username or email address. The application then emails “instructions” on what to do next. A bit of obfuscation is at work because the web application doesn’t at first explain what the next steps are. Indeed, the instructions are often different from one application to another. A temporary password may or may not be involved. Often, the email simply contains a URL containing one or more parameters and asks Joe to click on the link. The ensuing web page then allows Joe to reset his password. Unfortunately, Joe’s username may be shown on the web page, or it may have been in the email itself. This technique is really no more secure than any of the other methods because email is again the decisively weak link.

A real-world example of Technique C is presented below. Note that the application displays the username on the last step, which is very helpful to a hacker who has been able to extract the URL contained in the email.

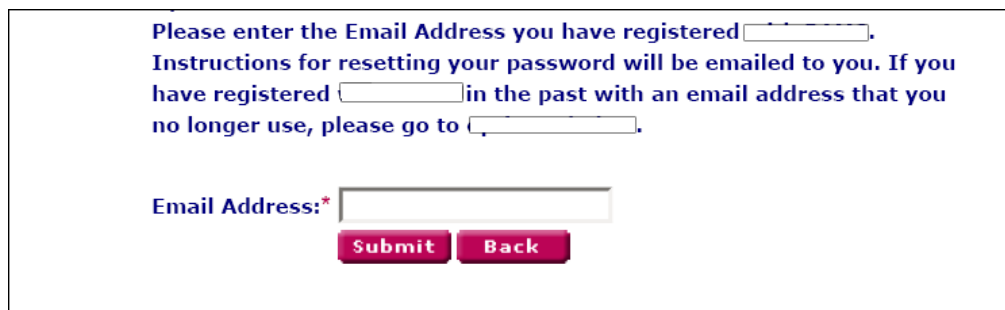


Figure C1 – You’re asked to enter your email address

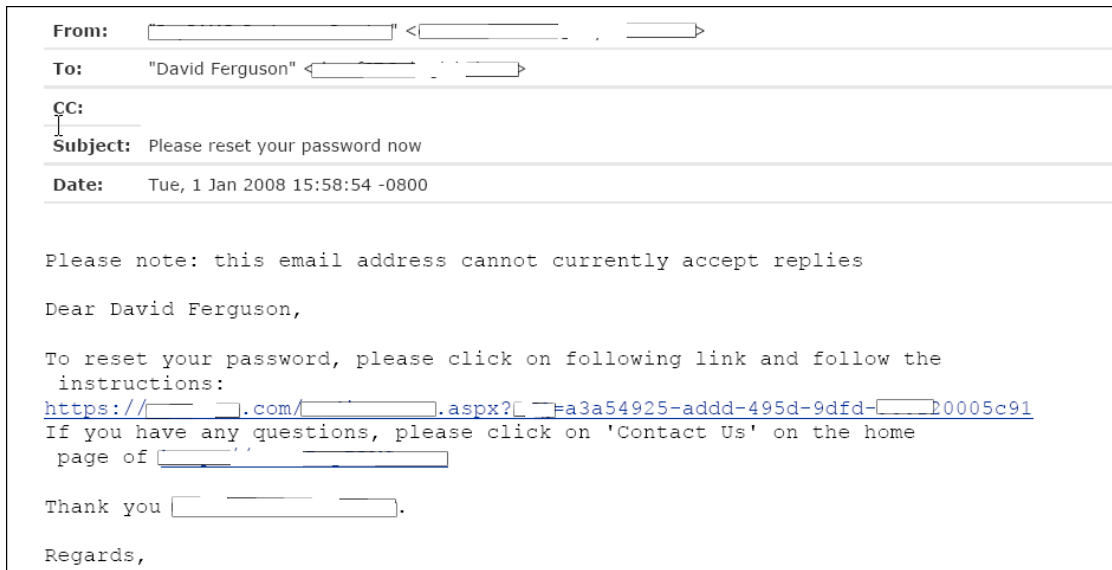


Figure C2 – You receive an email with a URL containing a cryptic parameter

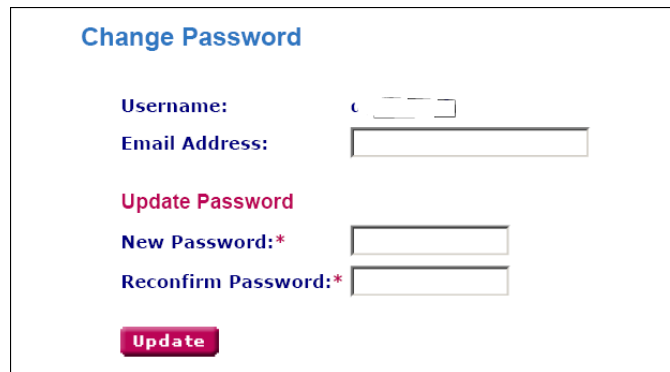


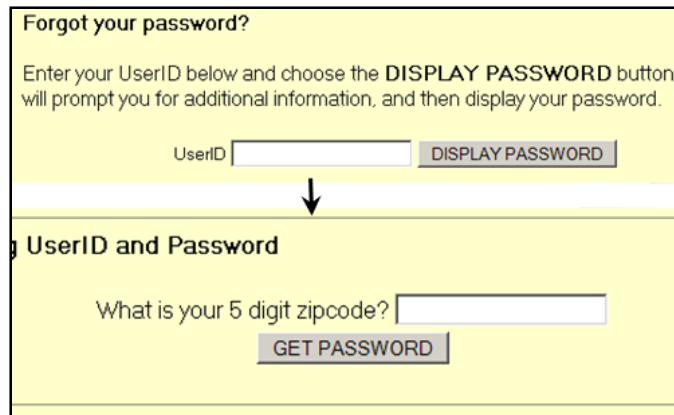
Figure C3 - Clicking on the URL opens a page that displays your username and allows you to reset your password

Weak Technique D – Display the current password after answering questions

This technique is substantially different than the others because it doesn't rely on email. That is a positive aspect. In this case Joe is asked to enter several pieces of information about himself. The information ostensibly consists of shared secrets between the user and the application. Examples are social security number, email address, zip code, account number, and date of birth. Personal security questions might be used as well.

The security flaw with this technique is that a user's password should never be displayed on a web page. The page could be cached by the browser to the local disk where it could be viewed by unauthorized parties. Or, the password could be lifted by a shoulder-surfing co-worker. Furthermore, it reveals that the application doesn't store user passwords in the database as one-way hash values as it should.

Three examples of Technique D taken from the real world are presented below.



Forgot your password?

Enter your UserID below and choose the **DISPLAY PASSWORD** button. will prompt you for additional information, and then display your password.

UserID

↓

UserID and Password

What is your 5 digit zipcode?

Figure D1 – An example where you enter username, then answer a personal question



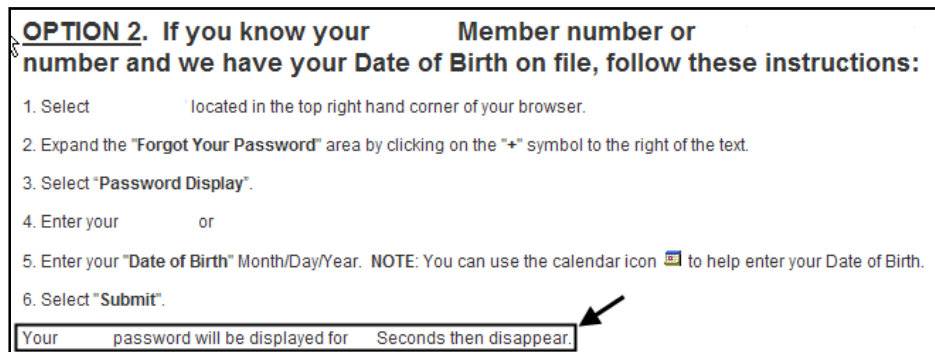
Option 2: Display my password right now

Type in your **email address** (the one used for select which you just purchased. If the information matches, we will display your password right now.) below, and

Your email address:

Select the product

Figure D2 – An example where you enter two pieces of information, after which your password is displayed



OPTION 2. If you know your Member number or number and we have your Date of Birth on file, follow these instructions:

1. Select located in the top right hand corner of your browser.
2. Expand the "Forgot Your Password" area by clicking on the "+" symbol to the right of the text.
3. Select "Password Display".
4. Enter your or
5. Enter your "Date of Birth" Month/Day/Year. NOTE: You can use the calendar icon to help enter your Date of Birth.
6. Select "Submit".

Your password will be displayed for Seconds then disappear

Figure D3 – An example where you can view your password, but only for a certain number of seconds

Weak Technique E – Provide a password hint or password reminder

This technique was very common in the early days of dynamic web applications, but seems to have lost favor in recent years. Like Technique D, it does not rely on email, and passwords are not displayed on the screen (theoretically anyway). In this case Joe has previously entered a word or phrase to help him remember his password.

The real trouble with this technique is that it implicitly assumes that a word or number meaningful to the user has been chosen as the password and that the user just needs a bit of information to help jog his memory. Meaningful words and numbers are typically not strong passwords, so applications of this type fundamentally tend to encourage weak passwords. Furthermore, most users are not security experts. The particular hint that Joe chooses could divulge too much information, making an attacker's job very easy. Joe might even decide to use his actual password as his password hint!

A real-world example of Technique E is presented below.



Figure E1 – You're asked to enter your username to see your password hint

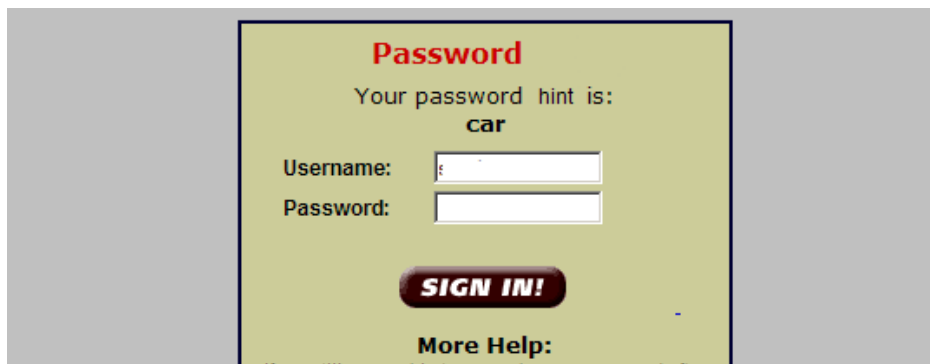


Figure E2 – Your password hint is displayed (in this case the password is probably the make and/or model of a car)

A Secure “Forgot Password” Feature

Enough about weak techniques. What are the elements of a secure forgot password feature? The bottom line is that an application following best practices should allow a user to reset his own password. Personal security questions should be used. The application doesn't need to rely on email, display passwords, nor set any temporary passwords. Below is a step-by-step (or page-by-page if you will) recommendation for a secure forgot password feature.

Step-by-Step Walk Through

Page 1 – Get hard data

The first page of a secure forgot password feature asks the user for multiple pieces of hard data. A single HTML form should be used for all of the inputs. An example is shown below.

Reset Your Password, step 1 of 3

Please enter the following information to begin the process of resetting your password.

Username *

Email Address *

10-digit Account Number *

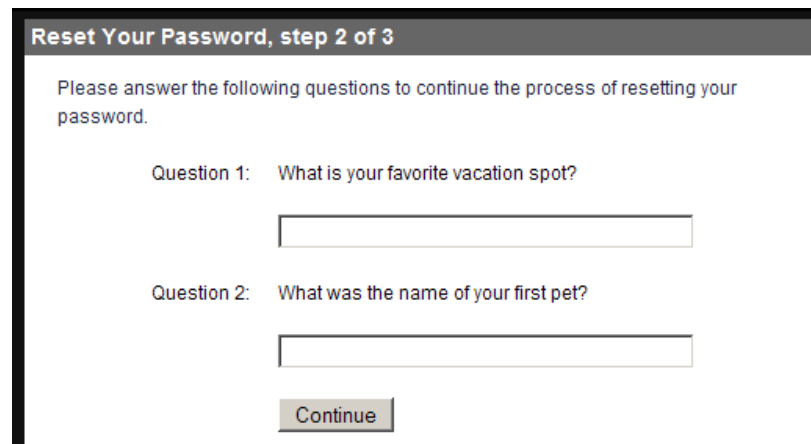
* Required

A minimum of three inputs is recommended, but the more you require, the more secure it will be. One of the inputs, preferably listed first, should be the username. Others should be selected depending on the nature of the data available to the application. Examples include:

- email address
- account number
- last name
- date of birth
- customer number
- last 4 digits of social security number
- zip code for address on file
- street number for address on file

Page 2 – Ask personal security questions

After the form on Page 1 is submitted, the application verifies that each piece of data is correct for the given username. If anything is incorrect, or if the username is not recognized, the second page displays a generic error message such as “Sorry, invalid data”. If all submitted data is correct, Page 2 should display at least two of the user’s pre-established personal security questions, along with input fields for the user to provide answers. It’s important that the answer fields are part of a single HTML form. An example is shown below.



Reset Your Password, step 2 of 3

Please answer the following questions to continue the process of resetting your password.

Question 1: What is your favorite vacation spot?

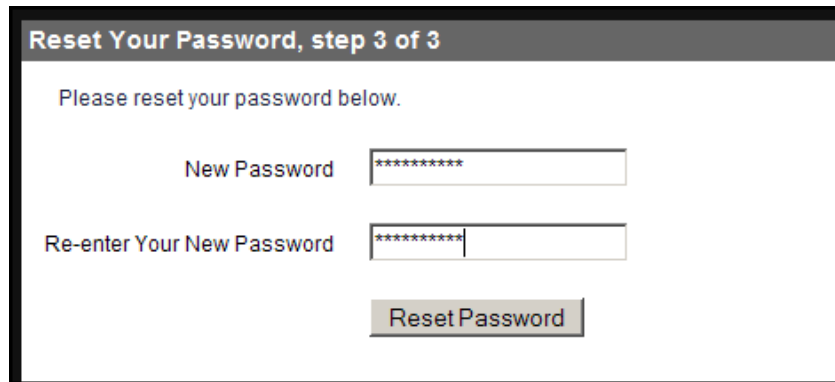
Question 2: What was the name of your first pet?

Continue

Do not provide a drop-down list for the user to select the questions he wants to answer. Be sure to avoid sending the username as a parameter (hidden or otherwise) when the form on this page is submitted. The username should be stored in the server-side session where it can be retrieved as needed.

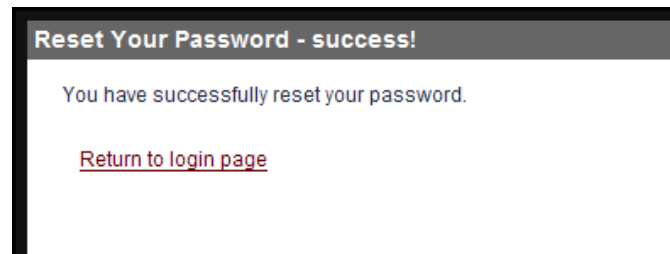
Page 3 – Allow user to reset password

When the answers to the security questions are submitted on Page 2, the application verifies that the answers are correct. If they aren't, the user is returned to the same page where he has another chance to answer. If the submitted answers are correct, Page 3 should (finally) allow the user to reset his password. Display a simple form with one input field for the new password and another field to confirm the new password. Make sure to enforce all password strength requirements that exist in other areas of the application. As before, avoid sending the username as a parameter when the form is submitted. An example is shown below.



Page 4 – Display success message

Display a “success” message. Explain that the user has successfully reset his password. Provide a button or link for the user to return to the normal login page of the application. An example is shown below.



Extra Protection

For applications where security is extremely critical, there's an extra step you can take to raise the bar even further. After Page 3, implement another step that requires the input of a special, randomly-generated code (10 characters or more) that the application sends to the user via email. This step introduces an "out of band" communication channel into the process. If an attacker somehow managed to successfully complete steps 1 to 3, he is unlikely to be able to compromise this secondary channel. The downside with this approach is that it creates a real problem for legitimate users who have an obsolete or invalid email address within the system.

DO's and DON'Ts

Here are some important DO's and DON'Ts to keep in mind when designing a secure forgot password feature.

- **DON'T** allow users to bypass any steps

Hackers use a technique called forced browsing to bypass application logic. Your application should ensure the user has completed every step in the password reset process. For example, implement a Boolean flag in the server-side session to know when a user has correctly answered the personal security questions. Display an error if a user attempts to skip that page by navigating directly to the final password reset page.

- **DO** establish personal security questions with each user

Establish at least three personal security question and answer sets with each user. In general, you'll want to provide five unique questions for every one question that is needed. For three questions required, provide users with a list of 15 questions to choose from. Supply a *pre-defined* list of questions. I don't recommend allowing users to create their own questions. Doing so opens the possibility for weak questions such as “what is your favorite color?”. The additional input fields also serve to increase the attack surface of the application. The additional inputs must be thoroughly validated and encoded or else cross-site scripting or some other type of attack may be possible.

- **DON'T** permit weak security questions

Assess the strength of each of the security questions you make available to users. Weak questions could lead to account compromise, especially when an application inadvertently discloses valid usernames. Case in point: An application I assessed employed a forgot password feature that disclosed valid usernames and asked two personal security questions. Several of the users I found were asked the following two questions:

What month is your wedding anniversary?

In what state were you born?

It wasn't difficult to see that only 600 combinations of answers are possible (assuming the users answered accurately). Compromising these particular accounts would have been fairly trivial. Needless to say, I recommended that our client remove these two questions from their list and replace them with stronger ones.

- **DON'T** disclose valid usernames

Usernames that are valid within the application should be closely held to the vest. Often the main authentication process correctly displays a generic "invalid login" message, but the forgot password feature does not. For example, the initial page of a forgot password feature should not have a single input field for username or email address. With only one input on the form, the next page inevitably tells a potential attacker whether or not the username entered was valid. This type of account enumeration is dangerous. Internet hackers could run a brute force or dictionary attack to pry their way into user accounts. If your application locks accounts after a certain number of failed login attempts, a malevolent denial-of-service attack could be launched against users.

- **DO** implement account lockout

For an extra measure of protection, lock the user's account after a certain number of failed attempts to complete one of the steps. Lockout simply means the application refuses all unauthenticated activity on the account. I suggest allowing between 5 and 10 attempts before triggering lockout (the login page should allow fewer). It's usually best to automatically unlock the account after a certain time period, such as one hour, has passed. Alternatively, you can require users to call in to request an unlock, but this is obviously more expensive and may require planning and coordination with other departments within your organization.

- **DON'T** rely on email

Your forgot password feature should not be dependent on email, because it is an inherently insecure form of communication. You should take the mindset that emails sent by your application will be viewed by unauthorized parties. Furthermore, you'll avoid the whole dilemma that occurs when a user's email address is no longer valid. The only exception to using email is the next item.

- **DO** send an email notification indicating the password was changed

After a successful password reset, the application should automatically send an email notification to the email address on record. The email shouldn't contain any password -- just simply state that the password was changed. Doing so may alert a victim that his or her account was compromised. Otherwise, it's just a harmless, secondary confirmation.

- **DO** store answers to personal security questions as one-way hashes with a salt

Similar to how passwords are treated, answers to personal security questions should be concatenated with a salt and stored in the database as one-way hash. Use a strong, industry-standard algorithm, such as SHA-256. For the salt, consider using the username or user ID from the database. To determine if incoming answers provided by users are correct, the

answers would be combined with the salt, hashed, and then compared to the values in the database.

- **DON'T** allow the HTTP GET method

Your application should accept only POST requests when sensitive data is sent by the client. Parameters sent with a GET request become part of the URL, while those sent with POST are sent in the request body. Requests for your forgot password feature will include sensitive data, so using GET causes sensitive data to appear in URLs, which are recorded in the browser history as well as web server logs. Reject GET requests summarily by returning 405/Method Not Allowed.

- **DON'T** recognize username on the final HTTP request

By the time your application receives the HTTP request that resets a password, you should be fully aware of which user's password is being reset. This information should be stored in the server-side session. Don't accept or recognize a username input on this final request. Attackers might tamper with the value.

About FishNet Security

Headquartered in Kansas City, Missouri, FishNet Security is a national leader in Information Security solutions, integration, and professional and managed services. Founded in 1996, FishNet Security continues to be a market leader in helping businesses, government, educational institutions, and other organizations define the true risks in their environment and deploy the right solutions and technologies to ensure the continued success of day-to-day operations and objectives. With offices spanning coast-to-coast, FishNet Security is able to provide industry leading engineers, consultants, forensic teams, educators, and account managers to all facets of businesses and organizations throughout the United States. We focus on the threat...so you can focus on the opportunity. For more information please visit us at <http://www.fishnetsecurity.com>.