

Authentication Testing



Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu



Introduction

- Authentication is the act of establishing or confirming something (or someone) as authentic, that is, that claims made by or about the thing are true.
- Authenticating an object may mean confirming its provenance, whereas authenticating a person often consists of verifying his/her identity.
- Authentication depends upon one or more authentication factors.



Introduction

- In computer security, authentication is the process of attempting to verify the *digital identity* of the sender of a communication.
- A common example of such a process is the log on process.
- Testing the authentication schema means understanding how the authentication process works and using that information to circumvent the authentication mechanism.

Testing for Credentials Transported over an Encrypted Channel: Summary

- Testing for credentials transport means verifying that the user's authentication data are transferred via an *encrypted channel* to avoid being intercepted by malicious users.
- The analysis focuses simply on trying to *understand if the data travels unencrypted from the web browser to the server*, or if the web application takes the appropriate security measures using a protocol like HTTPS.
- The HTTPS protocol is built on TLS/SSL to *encrypt the data* that is transmitted and to ensure that user is being sent towards the desired site.
- Clearly, the fact that traffic is encrypted does not necessarily mean that it's completely safe.
- The security also depends on the encryption algorithm used and the robustness of the keys that the application is using, but this particular topic will not be addressed.

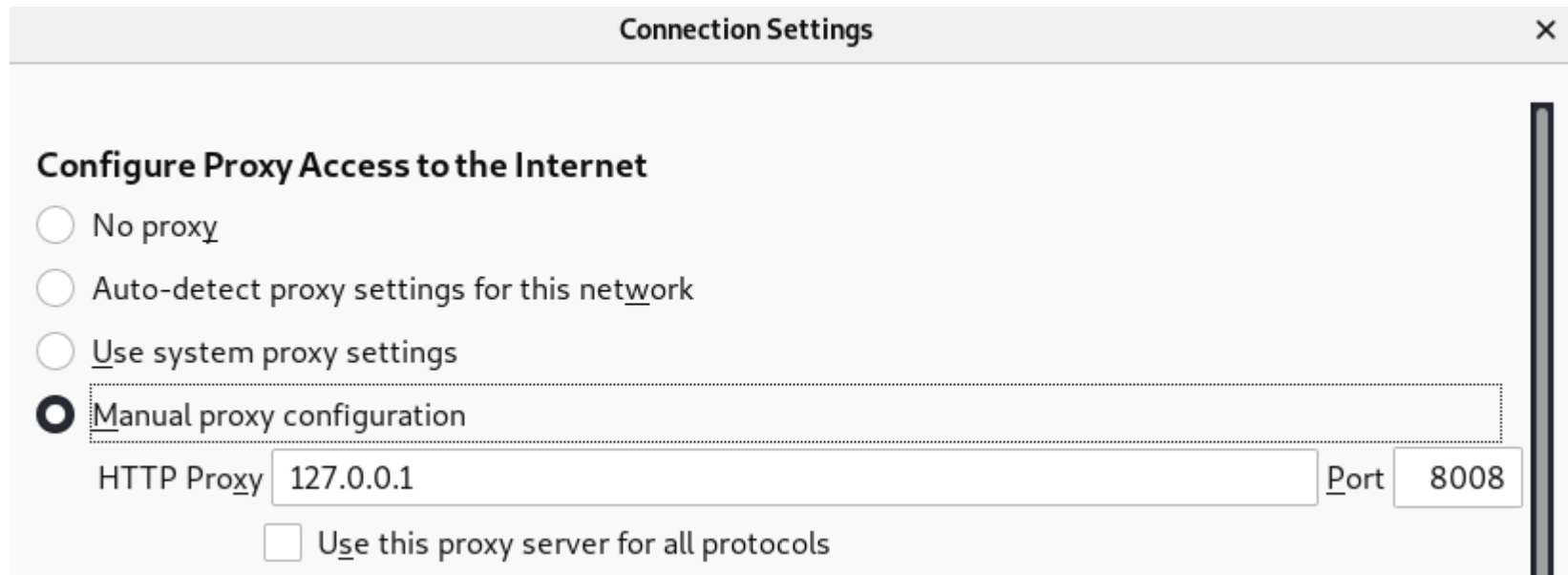


Testing for Credentials Transported over an Encrypted Channel: Summary

- Nowadays, the most common example of this issue is the log in page of a web application.
- The tester should verify that *user's credentials are transmitted via an encrypted channel*.
- In order to log in to a web site, the user usually has to fill a simple form that transmits the inserted data to the web application with the POST method.
- What is less obvious is that this data can be passed using the HTTP protocol, which transmits the data in a non-secure, clear text form, or using the HTTPS protocol, which encrypts the data during the transmission.
- To further complicate things, there is the possibility that the site has the login page accessible via HTTP (making us believe that the transmission is insecure), but then it actually sends data via HTTPS.
- This test is done to be sure that an attacker cannot retrieve sensitive information by simply sniffing the network with a sniffer tool.

Testing for Credentials Transported over an Encrypted Channel: How to Test

- WebScarab: capture packet headers and to inspect them. You can use any web proxy that you prefer.
 - Configure proxy



The image shows a screenshot of the Windows 'Connection Settings' dialog box. The title bar reads 'Connection Settings' with a close button (X) on the right. The main content area is titled 'Configure Proxy Access to the Internet'. There are four radio button options: 'No proxy', 'Auto-detect proxy settings for this network', 'Use system proxy settings', and 'Manual proxy configuration'. The 'Manual proxy configuration' option is selected. Below this, there are two input fields: 'HTTP Proxy' with the value '127.0.0.1' and 'Port' with the value '8008'. At the bottom, there is a checkbox labeled 'Use this proxy server for all protocols' which is currently unchecked.

Testing for Credentials Transported over an Encrypted Channel: How to Test

- WebScarab: capture packet headers and to inspect them. You can use any web proxy that you prefer.
 - Sending data with POST method through **HTTP**
 - Suppose that the login page presents a form with fields User, Password, and the Submit button to authenticate and give access to the application.
 - If we look at the headers of our request with WebScarab, we can get something like this:

```
Parsed Raw
POST http://testfire.net:80/doLogin HTTP/1.1
Host: testfire.net
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://testfire.net/login.jsp
Content-Type: application/x-www-form-urlencoded
Content-length: 37
Connection: keep-alive
Cookie: JSESSIONID=E04185FD9A77FE435AD9CF4D8E9CD0EF; AltoroAccounts=ODAwMDAwfkNvcnBvcnF0ZX41LjJzNzMyMjI2MUU3fDgwMDAwMX5DaGVja2luZ34xMDQ3ODEuNDR8
Upgrade-Insecure-Requests: 1
uid=admin&passw=admin&btnSubmit=Login
```

Testing for Credentials Transported over an Encrypted Channel: How to Test

- WebScarab: capture packet headers and to inspect them. You can use any web proxy that you prefer.
 - Sending data with POST method through **HTTPS**
 - Suppose that our web application uses the HTTPS protocol to encrypt the data we are sending (or at least for transmitting sensitive data like credentials).
 - In this case, when logging on to the web application the header of our POST request would be similar to the following:

```
Parsed Raw
POST http://ocsp.pki.goog:80/gts1o1 HTTP/1.1
Host: ocsp.pki.goog
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/ocsp-request
Content-length: 83
Connection: keep-alive
OQ0O0M0K0I0 [REDACTED]
```




Testing for Weak Lock Out Mechanism: Summary

- Account lockout mechanisms are used to mitigate *brute force password guessing attacks*.
- Accounts are typically locked after 3 to 5 *unsuccessful login attempts* and can only be unlocked
 - after a predetermined period of time
 - via a self-service unlock mechanism
 - or intervention by an administrator
- Account lockout mechanisms require a balance between protecting accounts from unauthorized access and protecting users from being denied authorized access.

Testing for Weak Lock Out Mechanism: Summary



- Without a strong lockout mechanism, the application may be susceptible to brute force attacks.
- After a successful brute force attack, a malicious user could have access to:
 - Confidential information or data:
 - Confidential documents, users' profile data, financial information, bank details, users' relationships, etc.
 - Administration panels:
 - Webmasters to manage (modify, delete, add) web application content, manage user provisioning, assign different privileges to the users, etc.
 - Opportunities for further attacks:
 - Authenticated sections of a web application could contain vulnerabilities that are not present in the public section of the web application and could contain advanced functionality that is not available to public users.



Testing for Weak lock out mechanism: Test Objectives

- Evaluate the account lockout mechanism's ability to mitigate brute force password guessing.
- Evaluate the unlock mechanism's resistance to unauthorized account unlocking.



Testing for Weak lock out mechanism: How to Test

- Typically, to test the strength of lockout mechanisms, you will need access to an account that you are willing or can afford to lock.
- If you have only one account with which you can log on to the web application, perform this test at the end of your test plan to avoid that you cannot continue your testing due to a locked account.
- To evaluate the account lockout mechanism's ability to mitigate brute force password guessing, attempt an invalid log in by using the incorrect password a number of times, before using the correct password to verify that the account was locked out.

Testing for Weak lock out mechanism: Example



- Attempt to log in with an incorrect password 3 times.
- Successfully log in with the correct password, thereby showing that the lockout mechanism doesn't trigger after 3 incorrect authentication attempts.

- Attempt to log in with an incorrect password 4 times.
- Successfully log in with the correct password, thereby showing that the lockout mechanism doesn't trigger after 4 incorrect authentication attempts.

- Attempt to log in with an incorrect password 5 times.
- Attempt to log in with the correct password. The application returns "Your account is locked out.", thereby confirming that the account is locked out after 5 incorrect authentication attempts.

Testing for Weak lock out mechanism: Example



- Attempt to log in with the correct password 5 minutes later. The application returns “Your account is locked out.”, thereby showing that the lockout mechanism does not automatically unlock after 5 minutes.
- Attempt to log in with the correct password 10 minutes later. The application returns “Your account is locked out.”, thereby showing that the lockout mechanism does not automatically unlock after 10 minutes.
- Successfully log in with the correct password 15 minutes later, thereby showing that the lockout mechanism automatically unlocks after a 10 to 15 minute period.



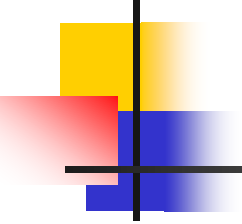
Testing for Bypassing Authentication Schema: Summary

- While most applications require authentication to gain access to private information or to execute tasks, not every authentication method is able to provide adequate security.
- Negligence, ignorance, or simple understatement of security threats often result in authentication schemes that can be bypassed by simply *skipping the log in page and directly calling an internal page* that is supposed to be accessed only after authentication has been performed.



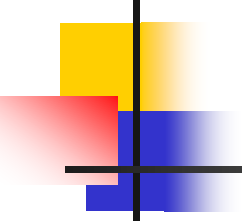
Testing for Bypassing Authentication Schema: Summary

- In addition, it is often possible to bypass authentication measures by *tampering with requests* and tricking the application into thinking that the user is already authenticated.
- This can be accomplished either by *modifying the given URL parameter*, by *manipulating the form*, or by *counterfeiting sessions*.



Testing for Bypassing Authentication Schema: How to Test

- There are several methods of bypassing the authentication schema that is used by a web application:
 - Direct page request (forced browsing)
 - Parameter modification



Testing for Bypassing Authentication Schema: How to Test

- Direct Page Request
 - If a web application implements access control *only on the log in page*, the authentication schema could be bypassed.
 - For example, if a user directly requests a different page via forced browsing, that page may not check the credentials of the user before granting access.
 - Attempt to directly access a protected page through the address bar in your browser to test using this method.

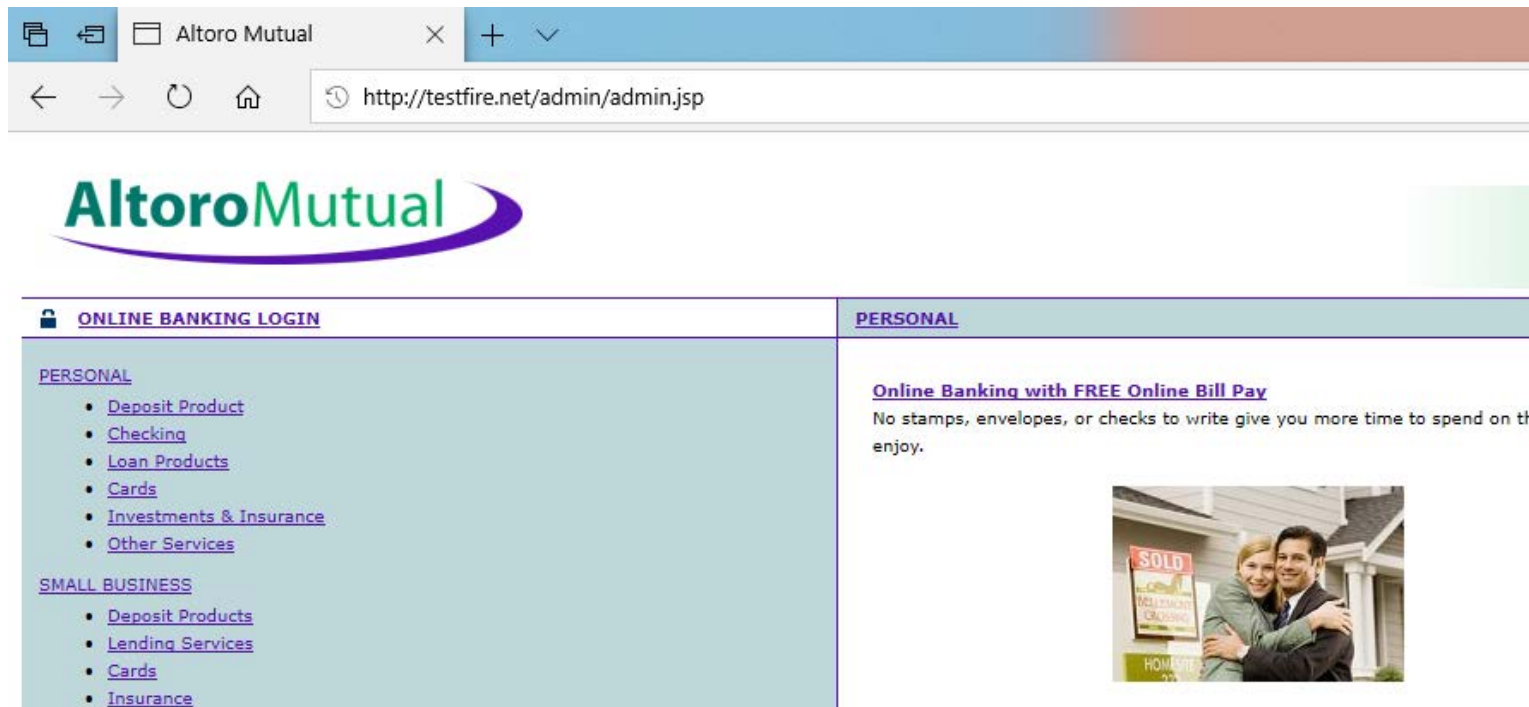
Testing for Bypassing Authentication Schema: How to Test

- Direct page request

The screenshot shows a web browser window with the address bar containing 'testfire.net/bank/main.jsp'. The page displays the Altoro Mutual logo and a navigation menu with 'MY ACCOUNT' and 'PERSONAL' tabs. The 'PERSONAL' tab is active, showing a personalized greeting: 'Hello Admin User'. Below the greeting, there is a welcome message: 'Welcome to Altoro Mutual Online.' and a 'View Account Details:' section with a dropdown menu showing '800000 Corporate' and a 'GO' button. A 'Congratulations!' message follows, stating: 'You have been pre-approved for an Altoro Gold Visa with a credit limit of \$10000!' and a link to 'Click Here to apply.' The footer contains links for 'Privacy Policy', 'Security Statement', 'Server Status Check', 'REST API', and '© 2019 Altoro Mutual, Inc.'

Testing for Bypassing Authentication Schema: How to Test

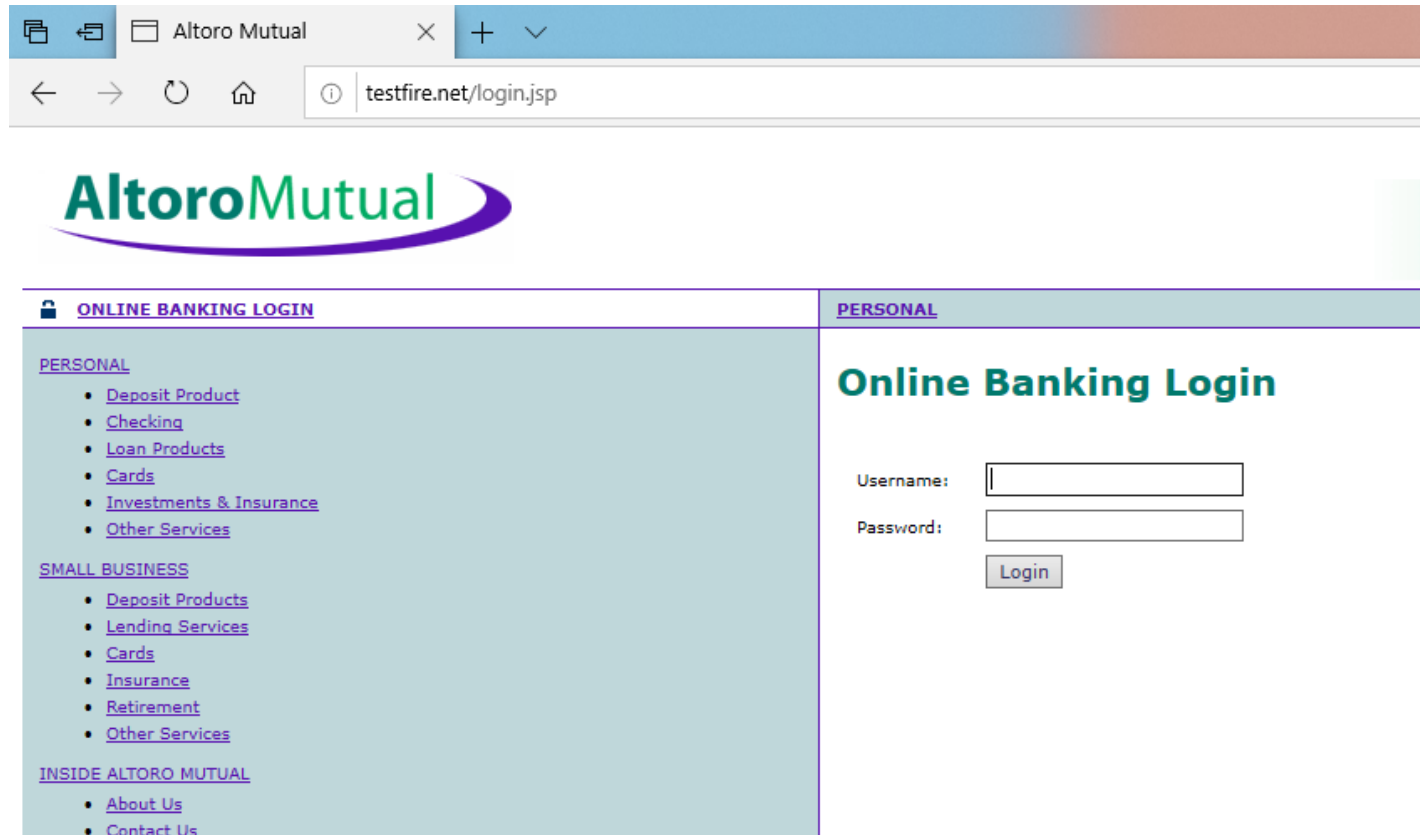
- Direct page request



The screenshot shows a web browser window with the address bar containing the URL `http://testfire.net/admin/admin.jsp`. The page displays the Altoro Mutual logo at the top. Below the logo, there is a navigation menu with two main sections: **ONLINE BANKING LOGIN** and **PERSONAL**. The **PERSONAL** section is expanded, showing a list of services: [Deposit Product](#), [Checking](#), [Loan Products](#), [Cards](#), [Investments & Insurance](#), and [Other Services](#). Below this, there is a **SMALL BUSINESS** section with links for [Deposit Products](#), [Lending Services](#), [Cards](#), and [Insurance](#). On the right side of the page, there is a promotional banner for **Online Banking with FREE Online Bill Pay**, which includes the text: "No stamps, envelopes, or checks to write give you more time to spend on the things you enjoy." Below the text is an image of a smiling couple standing in front of a house with a "SOLD" sign.

Testing for Bypassing Authentication Schema: How to Test

- Direct page request



The screenshot displays a web browser window with the following details:

- Browser Tab:** Altoro Mutual
- Address Bar:** testfire.net/login.jsp
- Page Header:** AltoroMutual logo
- Navigation:** ONLINE BANKING LOGIN (with a lock icon) and PERSONAL (highlighted in a light blue bar)
- Left Sidebar:**
 - PERSONAL**
 - [Deposit Product](#)
 - [Checking](#)
 - [Loan Products](#)
 - [Cards](#)
 - [Investments & Insurance](#)
 - [Other Services](#)
 - SMALL BUSINESS**
 - [Deposit Products](#)
 - [Lending Services](#)
 - [Cards](#)
 - [Insurance](#)
 - [Retirement](#)
 - [Other Services](#)
 - INSIDE ALTORO MUTUAL**
 - [About Us](#)
 - [Contact Us](#)
- Main Content Area:**
 - PERSONAL** (highlighted)
 - Online Banking Login**
 - Username:
 - Password:
 -



Testing for Bypassing Authentication Schema: How to Test

- Parameter Modification
 - Another problem related to authentication design is when the application verifies a successful log in on the basis of a fixed value parameters.
 - A user could modify these parameters to gain access to the protected areas without providing valid credentials.

Testing for Bypassing Authentication Schema: How to Test

- Parameter Modification

- In this example, the parameter is in the URL, but a proxy could also be used to modify the parameter, especially when the parameters are sent as form elements in a POST request or when the parameters are stored in a cookie.

<http://www.site.com/page.asp?authenticated=no>

```
raven@blackbox /home $nc www.site.com 80
GET /page.asp?authenticated=yes HTTP/1.0
```

```
HTTP/1.1 200 OK
Date: Sat, 11 Nov 2006 10:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

```
<!DOCTYPE HTML PUBLIC "-//IETF/DTD HTML 2.0/EN">
<HTML><HEAD>
</HEAD><BODY>
<H1>You Are Authenticated</H1>
</BODY></HTML>
```



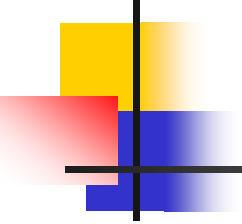
Testing for Vulnerable Remember Password: Summary

- Browsers will sometimes ask a user if they wish to remember the password that they just entered.
- The browser will then store the password, and automatically enter it whenever the same authentication form is visited.
- This is a convenience for the user.
- Additionally some websites will offer custom “*remember me*” *functionality* to allow users to persist log ins on a specific client system.



Testing for Vulnerable Remember Password: Summary

- Having the browser store passwords is not only a convenience for end-users, but also for an attacker.
- If an attacker can gain access to the victim's browser (e.g. through a Cross Site Scripting attack, or through a shared computer), then they can retrieve the stored passwords.
- It is not uncommon for browsers to store these passwords in an easily retrievable manner, but even if the browser were to store the passwords encrypted and only retrievable through the use of a master password, an attacker could retrieve the password by visiting the target web application's authentication form, entering the victim's username, and letting the browser to enter the password.



Testing for Vulnerable Remember Password: How to Test

- Look for passwords being stored in a cookie.
 - Examine the cookies stored by the application.
 - Verify that the credentials are not stored in clear text, but are hashed.
- Examine the hashing mechanism: if it is a common, well-known algorithm, check for its strength; in hash functions, attempt several usernames to check whether the hash function is easily guessable.
- Verify that the credentials are only sent during the log in phase, and not sent together with every request to the application.
- Consider other sensitive form fields
 - e.g. an answer to a secret question that must be entered in a password recovery or account unlock form.



Testing for Weak password policy: Summary

- The most prevalent and most easily administered authentication mechanism is a static password.
- The password represents the keys to the kingdom, but is often subverted by users in the name of usability.
- In each of the recent high profile hacks that have revealed user credentials, it is lamented that most common passwords are still: 123456, password and qwerty.



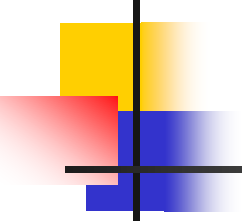
Testing for Weak password policy: Test Objectives

- Determine the resistance of the application against brute force password guessing using available password dictionaries by evaluating the length, complexity, reuse and aging requirements of passwords.



Testing for Weak password policy: How to Test

- What characters are permitted and forbidden for use within a password? Is the user required to use characters from different character sets such as lower and uppercase letters, digits and special symbols?
- How often can a user change their password? How quickly can a user change their password after a previous change? Users may bypass password history requirements by changing their password 5 times in a row so that after the last password change they have configured their initial password again.
- When must a user change their password? After 90 days? After account lockout due to excessive log on attempts?
- How often can a user reuse a password? Does the application maintain a history of the user's previous used 8 passwords?
- How different must the next password be from the last password?
- Is the user prevented from using his username or other account information (such as first or last name) in the password?



Testing for Weak Security

Question/Answer: Summary

- Often called “secret” questions and answers, security questions and answers are often used to recover forgotten passwords, or as extra security on top of the password.
- They are typically generated upon account creation and require the user to select from some pre-generated questions and supply an appropriate answer.
- They may allow the user to generate their own question and answer pairs.
- Both methods are prone to insecurities.
- Ideally, security questions should generate answers that are only known by the user, and not guessable or discoverable by anybody else.
- This is harder than it sounds.



Testing for Weak Security Question/Answer: Summary

- Security questions and answers rely on the secrecy of the answer.
- Questions and answers should be chosen so that the answers are only known by the account holder.
- However, although a lot of answers may not be publicly known, most of the questions that websites implement promote answers that are pseudo-private.



Testing for Weak Security Question/Answer

- Pre-generated questions:
 - The majority of pre-generated questions are fairly simplistic in nature and can lead to insecure answers.
 - The answers may be known to family members or close friends of the user, e.g. “What is your mother’s maiden name?”, “What is your date of birth?”
 - The answers may be easily guessable, e.g. “What is your favorite color?”, “What is your favorite baseball team?”
 - The answers may be brute forcible, e.g. “What is the first name of your favorite high school teacher?” - the answer is probably on some easily downloadable lists of popular first names, and therefore a simple brute force attack can be scripted.
 - The answers may be publicly discoverable, e.g. “What is your favorite movie?” - the answer may easily be found on the user’s social media profile page.



Testing for Weak Security Question/Answer

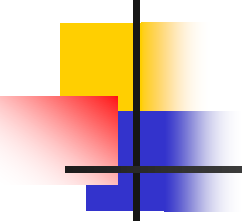
- Self-generated questions:
 - The problem with having users to generate their own questions is that it allows them to generate very insecure questions, or even bypass the whole point of having a security question in the first place.
 - Here are some real world examples that illustrate this point:
 - “What is 1+1?”
 - “What is your username?”
 - “My password is M3@t\$p|N”



Testing for Weak Security

Question/Answer: How to Test:

- Testing for weak pre-generated questions:
 - Try to obtain a list of security questions by creating a new account or by following the “I don’t remember my password”-process.
 - Try to generate as many questions as possible to get a good idea of the type of security questions that are asked.
 - If any of the security questions fall in the categories described above, they are vulnerable to being attacked (guessed, brute-forced, available on social media, etc.).



Testing for Weak Security

Question/Answer: How to Test:

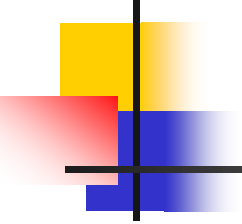
- Testing for weak self-generated questions:
 - Try to create security questions by creating a new account or by configuring your existing account's password recovery properties.
 - If the system allows the user to generate their own security questions, it is vulnerable to having insecure questions created.
 - If the system uses the self-generated security questions during the forgotten password functionality and if usernames can be enumerated, then it should be easy for the tester to enumerate a number of self-generated questions.
 - It should be expected to find several weak self-generated questions using this method.



Testing for Weak Security

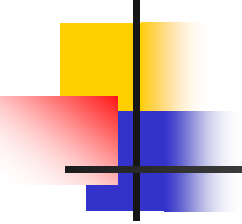
Question/Answer: How to Test:

- Testing for brute-forcible answers:
 - Use the methods described in Testing for Weak lock out mechanism to determine if a number of incorrectly supplied security answers trigger a lockout mechanism.



Testing for Weak Security Question/Answer: How to Test:

- The first thing to take into consideration when trying to exploit security questions is the number of questions that need to be answered.
- The majority of applications only need the user to answer a single question, whereas some critical applications may require the user to answer two or even more questions.
- The next step is to assess the strength of the security questions.
- Could the answers be obtained by a simple Google search or with social engineering attack?



Testing for Weak Security Question/Answer: How to Test:

- As a penetration tester, here is a step-by-step walk-through of exploiting a security question scheme:
 - Does the application allow the end-user to choose the question that needs to be answered?
 - Determine how many guesses you have if possible.
 - Pick the appropriate question based on analysis from the above points, and do research to determine the most likely answers.



Testing for Weak Password Change or Reset Functionalities: Summary

- The password change and reset function of an application is a self-service password change or reset mechanism for users.
 - This self-service mechanism allows users to quickly change or reset their password without an administrator intervening.
- When passwords are changed they are typically changed within the application.
- When passwords are reset they are either rendered within the application or emailed to the user.
- This may indicate that the passwords are stored in plain text or in a decryptable format.



Testing for Weak Password Change or Reset Functionalities: Test Objectives

- Determine the resistance of the application to subversion of the account change process allowing someone to change the password of an account.
- Determine the resistance of the passwords reset functionality against guessing or bypassing.



Testing for Weak Password Change or Reset Functionalities: How to Test:

- For both password change and password reset it is important to check:
 - if users, other than administrators, can change or reset passwords for accounts other than their own.
 - if users can manipulate or subvert the password change or reset process to change or reset the password of another user or administrator.



Testing for Weak Password Change or Reset Functionalities: How to Test:

- Test Password Reset:
 - What information is required to reset the password?
 - How are reset passwords communicated to the user?
 - Are reset passwords generated randomly?
 - Is the reset password functionality requesting confirmation before changing the password?
- Test Password Change:
 - Is the old password requested to complete the change??