

## 3.1

### Design Flaws in Authentication

Authentication functionality is subject to more design weaknesses than any

other security mechanism commonly employed in web applications.

#### Bad Passwords

Many web applications employ no or minimal controls over the quality of users’

passwords. It is common to encounter applications that allow passwords that are:

* Very short or blank
* Common dictionary words or names
* The same as the username
* Still set to a default value

#### Brute-Forcible Login

Login functionality presents an open invitation for an attacker to try to guess

usernames and passwords and therefore gain unauthorized access to the application. If the application allows an attacker to make repeated login attempts with different passwords until he guesses the correct one, it is highly vulnerable

even to an amateur attacker who manually enters some common usernames

and passwords into his browser.

Recent compromises of high-profile sites have provided access to hundreds

of thousands of real-world passwords that were stored either in cleartext or

using brute-forcible hashes. Here are the most popular real-world passwords:

* password
* website name
* 12345678
* qwerty
* abc123
* 111111
* monkey
* 12345
* letmein

#### Verbose Failure Messages

A typical login form requires the user to enter two pieces of information — a

username and password. Some applications require several more, such as date

of birth, a memorable place, or a PIN.

When a login attempt fails, you can of course infer that at least one piece of

information was incorrect. However, if the application tells you which piece of

information was invalid, you can exploit this behavior to considerably diminish

the effectiveness of the login mechanism.

#### Vulnerable Transmission of Credentials

If an application uses an unencrypted HTTP connection to transmit login credentials, an eavesdropper who is suitably positioned on the network can, of

course, intercept them. Depending on the user’s location, potential eavesdroppers may reside:

* On the user’s local network
* Within the user’s IT department
* Within the user’s ISP
* On the Internet backbone
* Within the ISP hosting the application
* Within the IT department managing the application

Even if login occurs over HTTPS, credentials may still be disclosed to unauthorized parties if the application handles them in an unsafe manner:

* If credentials are transmitted as query string parameters, as opposed

to in the body of a POST request, these are liable to be logged in various

places, such as within the user’s browser history, within the web server

logs, and within the logs of any reverse proxies employed within the

hosting infrastructure. If an attacker succeeds in compromising any of

these resources, he may be able to escalate privileges by capturing the

user credentials stored there.

* Although most web applications do use the body of a POST request to

submit the HTML login form itself, it is surprisingly common to see the

login request being handled via a redirect to a different URL with the same

credentials passed as query string parameters. Why application developers consider it necessary to perform these bounces is unclear, but having

elected to do so, it is easier to implement them as 302 redirects to a URL

than as POST requests using a second HTML form submitted via JavaScript.

* Web applications sometimes store user credentials in cookies, usually

to implement poorly designed mechanisms for login, password change,

“remember me,” and so on. These credentials are vulnerable to capture

via attacks that compromise user cookies and, in the case of persistent

cookies, by anyone who gains access to the client’s local filesystem. Even if

the credentials are encrypted, an attacker still can simply replay the cookie

and therefore log in as a user without actually knowing her credentials.

#### Password Change Functionality

Surprisingly, many web applications do not provide any way for users to change

their password. However, this functionality is necessary for a well-designed

authentication mechanism for two reasons:

* Periodic enforced password change mitigates the threat of password compromise. It reduces the window in which a given password can be targeted in a guessing attack. It also reduces the window in which a compromised password can be used without detection by the attacker.
* Users who suspect that their passwords may have been compromised need to be able to quickly change their passwords to reduce the threat of unauthorized use.

#### Forgotten Password Functionality

Like password change functionality, mechanisms for recovering from a forgotten password situation often introduce problems that may have been avoided

in the main login function, such as username enumeration.

In addition to this range of defects, design weaknesses in forgotten password functions frequently make this the weakest link at which to attack the

application’s overall authentication logic.

#### “Remember Me” Functionality

#### User Impersonation Functionality

#### Incomplete Validation of Credentials

#### Nonunique Usernames

#### Predictable Usernames

#### Predictable Initial Passwords

#### Insecure Distribution of Credentials

### Implementation Flaws in Authentication

1. Fail-Open Login Mechanisms
2. Defects in Multistage Login Mechanisms
3. Insecure Storage of Credentials

## 3.2

### Securing Authentication

1. Use Strong Credentials
2. Handle Credentials Secretively
3. Validate Credentials Properly
4. Prevent Information Leakage
5. Prevent Brute-Force Attacks
6. Prevent Misuse of the Password Change Function
7. Prevent Misuse of the Account Recovery Function
8. Log, Monitor, and Notify

## 3.3

### Hydra →online tool

hydra -l ignite -P pass.txt 192.168.1.141 ftp

-P = password list

-l = for username

### John the Ripper: used for password cracking (dictionary attack, brute force) →offline tool

Commands:

john –worldlist=/path/to/file –format=<raw-md5/sha1> <shadowed file>

(possible for single user as well as multiple users

John –show passfile