Hacktempt: Analysis and demonstration of known security vulnerabilities by building an interactive web application

Riccardo DETOMASO
r.detomaso@studenti.poliba.it
matr.589945

Alessandro IANNONE
a.iannone3@studenti.poliba.it
matr.590152

Tangari MAURO
m.tangari3@studenti.poliba.it
matr.589155

Abstract

This article provides an in-depth analysis of common vulnerabilities found in computer systems, specifically Identification and Authentication Failure, Broken Access Control, Cryptographic Failure, SQL Injection, Security Misconfiguration, Restrict URL Access, and Phishing. These vulnerabilities pose significant risks to the confidentiality, integrity, and availability of sensitive information and can lead to unauthorized access, data breaches, and system compromise.

Table of contents

I. Introduction

Web application security has become an essential priority in today's technology landscape. Cyber-attacks targeting application vulnerabilities pose a constant threat to the confidentiality, integrity, and availability of data. Therefore, it is critical that developers, system administrators, and cybersecurity experts thoroughly understand the most common security vulnerabilities in order to identify, mitigate, and prevent potential attacks.

This paper aims to conduct an in-depth analysis and demonstrate some of the most well-known and relevant security vulnerabilities, including Identification and Authentication Failure, Broken Access Control, Cryptographic Failure, SQL Injection, Security Misconfiguration, Restrict URL Access, and Phishing. In order to provide a hands-on and engaging approach, an interactive demonstration website specifically designed to incorporate these vulnerabilities will be developed.

Precisely through the creation of a web application with deliberate vulnerabilities, the consequences and risks associated with these vulnerabilities will be illustrated in a detailed and tangible manner. Specific attack techniques, exploitation methods, and potential compromise scenarios will be presented. In addition, in-depth explanations of attack mechanisms will be provided, along with strategies and techniques to mitigate these vulnerabilities and strengthen web application security.

The demonstration website will provide an opportunity to explore and interact with simulated situations where these vulnerabilities can be exploited. Concrete examples and code illustrations will be provided to facilitate understanding of the vulnerabilities and related countermeasures to be taken.

II. Overview

In recent years, the Open Web Application Security Project (OWASP) has identified and ranked the most critical vulnerabilities plaguing web applications through its OWASP Top 10 list. This list has evolved significantly over time, reflecting the adaptation of attackers to new technologies and emerging challenges. Specifically, to date the most common vulnerabilities and attacks are:

- **Identification and Authentication Failure:** Identification and authentication have always been a crucial component of web application security. Over the years, vulnerabilities in this area have become increasingly sophisticated, prompting developers to implement more robust protection measures, such as two-factor authentication and the use of secure access tokens.
- **Broken Access Control:** Access and privilege management is an area where numerous vulnerabilities have emerged over time. Attackers are always looking for flaws that allow them to bypass access controls and gain inappropriate permissions. The evolution of Web applications has led to the need to implement more refined access control mechanisms, such as role-based access control lists and granular access policies.
- **Cryptographic Failure:** Cryptography is critical to protect the confidentiality and integrity of data in web applications. Over the years, vulnerabilities related to cryptography have mainly involved the misuse of cryptographic algorithms, inadequate key management, and implementation weaknesses. Evolution has led to the adoption of more robust cryptographic algorithms and the implementation of more secure key management procedures.
- **SQL Injection:** SQL Injection is a vulnerability that has been present for quite some time and has had a significant impact on Web applications. Attackers exploit SQL code injection to gain unauthorized access to the underlying database or to perform malicious operations. Over the years, mitigation mechanisms such as the use of prepared parameters and the implementation of data validation and sanitization controls have been developed.
- **Security Misconfiguration:** Misconfigurations or inadequate configurations of web applications are one of the most common vulnerabilities. These vulnerabilities can allow attackers to obtain sensitive information, perform unauthorized actions, or exploit system weaknesses. Over the years, efforts have focused on establishing guidelines for secure configurations, as well as automating security audits of configurations.
- **Restrict URL Access:** URL access vulnerabilities are related to inappropriate or missing restrictions in accessing Web resources. Attackers can exploit these vulnerabilities to gain access to sensitive information or to perform unauthorized operations. Over time, developers have adopted practices such as implementing robust authorization controls and using secure session management mechanisms to mitigate this type of vulnerability.
- **Phishing:** Phishing is a form of attack that aims to obtain sensitive information, such as user login credentials, through deception and manipulation. Over the years, attackers have refined their phishing techniques, using increasingly sophisticated methods to deceive users. Efforts to counter phishing have focused on educating users, implementing more secure authentication mechanisms, and using advanced detection and prevention tools.

III. Working methodology

A model that is very close to the agile approach was adopted for the creation of this web application. A waterfall methodology was not chosen as it was slower and less productive. The development skills possessed by the team allowed for good collaboration, communication, and most importantly, simultaneity of work.

In the realm of project management, two prominent methodologies have emerged: Agile and Waterfall. Agile is an iterative and flexible approach that emphasizes collaboration and adaptability, while Waterfall follows a sequential, linear path with well-defined phases. Understanding the differences between these methodologies, along with their respective pros and cons, is crucial for making informed decisions about project management approaches. Let's delve into a comprehensive comparison.

Agile is an iterative and flexible approach to project management, widely used in software development but applicable to various industries. It focuses on delivering value early and frequently, adapting to change, and fostering collaboration. Here are some key aspects of Agile:

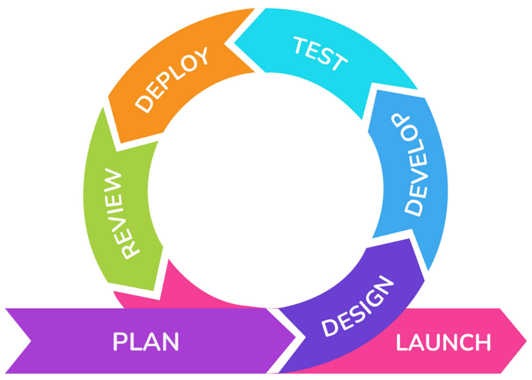
- **Iterative and Incremental Development**: Agile projects are divided into small iterations or sprints, typically lasting a few weeks. Each iteration produces a working increment of the software, allowing for regular feedback and improvement.

- **Flexibility and Adaptability**: Agile embraces change as a natural part of the development process. It recognizes that requirements evolve, and stakeholders' priorities may shift over time. Agile teams are responsive and can quickly adjust their plans to accommodate changes.

- **Customer Collaboration**: Agile methodologies emphasize frequent communication and collaboration with customers or stakeholders. Their input is actively sought and incorporated throughout the development process. This iterative feedback loop ensures that the final product meets the customer's expectations.

- **Self-Organizing Teams**: Agile promotes self-organizing teams that have the autonomy to make decisions and adapt to changing circumstances. This empowers team members to take ownership of their work and fosters innovation and creativity.

- **Continuous Improvement**: Agile encourages continuous learning and improvement. After each iteration, the team reflects on what went well and identifies areas for enhancement. This iterative feedback loop helps the team refine their processes and deliver higher-quality products.



Waterfall is a traditional, sequential project management methodology that follows a linear approach. It is often used for projects with well-defined and stable requirements. Here are some key aspects of Waterfall:

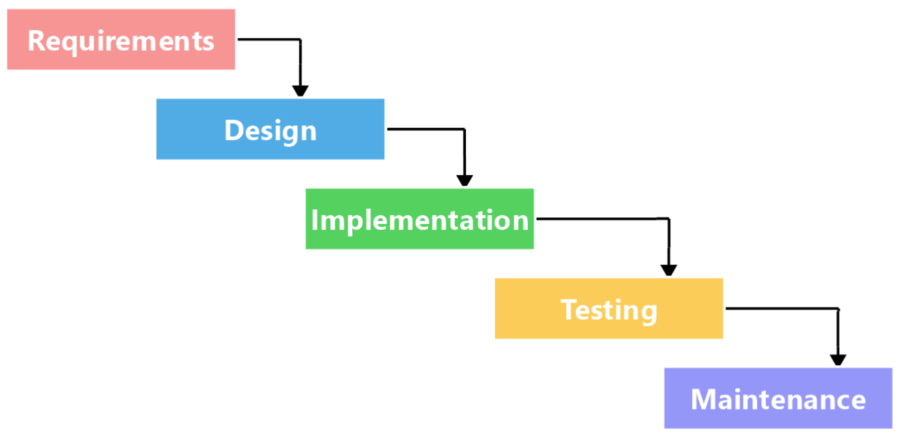
- **Sequential Structure**: Waterfall projects follow a fixed sequence of phases, typically including requirements gathering, design, implementation, testing, and deployment. Each phase has its specific objectives and outputs, and they must be completed in order before progressing to the next phase.

- **Comprehensive Planning**: Waterfall requires extensive planning upfront, with a focus on documenting requirements and creating a comprehensive project plan. This initial planning phase aims to identify all project requirements and define the scope of work.

- **Limited Customer Involvement**: In Waterfall, customer involvement is typically limited to the initial phase of gathering requirements. Once the requirements are defined and the project plan is in place, changes requested later in the process can be challenging and costly to implement.

- **Clear Milestones and Formal Sign-offs**: Waterfall emphasizes clear milestones and formal sign-offs at the end of each phase. These milestones provide checkpoints to track progress and ensure that each phase meets predetermined criteria before moving forward.

- **Emphasis on Documentation**: Waterfall places a strong emphasis on documentation throughout the project. Detailed documentation ensures that the project's progress can be tracked, and knowledge can be transferred to future teams.



Pros of Agile:

- Flexibility: Agile embraces change, allowing project teams to respond swiftly to evolving requirements and market dynamics.
- Customer Collaboration: Agile methodologies actively involve customers throughout the project, ensuring alignment with their expectations and increasing customer satisfaction.
- Incremental Value Delivery: Agile focuses on delivering functional increments of the product at regular intervals, enabling customers to realize value early on.
- Enhanced Team Collaboration: Agile encourages self-organizing teams that foster effective communication, shared ownership, and cross-functional collaboration.
- Continuous Improvement: Agile promotes learning and improvement through regular feedback loops, enabling teams to refine their processes and deliver higher-quality products.

Cons of Agile:

- Scope Creep: The flexibility of Agile may lead to uncontrolled changes and scope creep if not managed effectively, potentially impacting project timelines and budgets.
- Customer Availability: Active customer involvement is crucial in Agile, which can be challenging if stakeholders have limited availability or conflicting priorities.
- Uncertainty in Planning: Agile projects may lack upfront predictability, as requirements and priorities evolve, making it challenging to estimate timelines and resource allocation accurately.
- Skill Requirements: Agile methodologies require skilled team members who can adapt to change, collaborate effectively, and make autonomous decisions.
- Documentation Challenges: Agile places less emphasis on extensive documentation, making it difficult to maintain a comprehensive record of the project's progress and decisions.

Pros of Waterfall:

- Clear Planning and Phased Structure: Waterfall provides a well-defined roadmap with distinct phases, making it easier to plan and manage projects.
- Controlled Scope: Waterfall's emphasis on upfront planning helps define project scope and reduces the risk of scope creep during implementation.
- Documentation and Traceability: Waterfall requires comprehensive documentation, ensuring a clear record of requirements, design decisions, and project progress.
- Defined Milestones and Deliverables: Waterfall establishes clear milestones and formal sign-offs, enabling better tracking of progress and managing stakeholder expectations.
- Ease of Project Management: Waterfall's linear nature simplifies project management, as each phase has specific objectives, outputs, and dependencies.

Cons of Waterfall:

- Limited Flexibility for Change: Waterfall's sequential approach makes it challenging to accommodate changes once a phase is completed, potentially leading to delays and cost overruns.
- Limited Customer Involvement: Waterfall primarily involves customers in the initial requirements gathering phase, which may result in a final product that does not fully meet their evolving needs.
- Late Value Delivery: Waterfall's linear approach often delays value delivery until the end of the project when the final product is deployed, potentially misaligning with customer expectations.
- Adaptability to Uncertainty: Waterfall struggles to handle uncertainties and evolving requirements as it relies heavily on upfront planning and may require significant rework to accommodate changes.
- Limited Team Collaboration: Waterfall's hierarchical structure may limit collaboration and creativity, as decision-making is often centralized, and team members have less autonomy.

Agile and Waterfall methodologies offer distinct approaches to project management, each with its own set of advantages and disadvantages. Agile's flexibility, customer collaboration, and continuous improvement make it suitable for dynamic projects, while Waterfall's clear planning, controlled scope, and ease of management align well with stable and well-defined projects. Choosing the right methodology depends on factors such as project characteristics, customer requirements, team dynamics, and tolerance for change. By understanding the pros and cons of Agile and Waterfall, project managers can make informed decisions to maximize project success.

IV. Development process

The development process of the web app went through the following steps, similar to the process of devops:

- **Planning**: define project goals, identify user needs and plan activities.
- **Prototype development**: already have a starting idea of what is the goal to be achieved and meet and solve the first operational problems
- **Code development and versioning**: writing code and implementing software functionality. Using source code control tools, such as Git, the code is versioned and managed collaboratively. During this process, work is done iteratively, frequently delivering portions of working code.
- **Integration**: codes developed in parallel and concurrent manners are continuously integrated
- **Monitoring**: once the web application was created, all different scenarios were tested to cause unexpected behavior.
- **Improvement**: bugs are fixed and negative aspects that emerged in the previous phase are improved

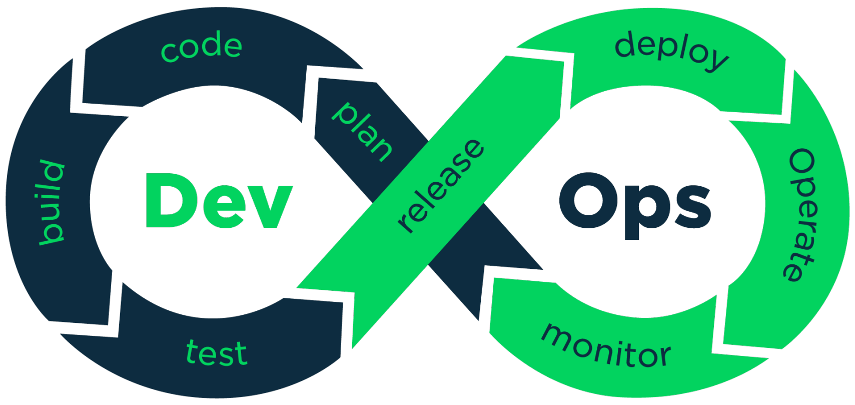
V. What is DevOps?

DevOps is a software development philosophy that promotes close collaboration between development and operations teams, aiming to improve the efficiency, quality and speed of software delivery. Although there are no rigid phases in DevOps, it is possible to identify some key concepts and common practices that characterize the approach. Here is an interpretation of the stages of DevOps:

- **Planning and Analysis**: In this phase, the development and operations teams work together to define project goals, identify user needs, and plan activities. Business objectives are established and requirements and priorities for software delivery are defined.
- **Development and Versioning**: In this phase, development teams write code and implement software functionality. Using source code control tools, such as Git, the code is versioned and managed collaboratively. During this process, teams work iteratively, frequently delivering portions of working code.
- **Continuous Integration**: In this phase, code written by different team members is integrated continuously and automatically. Continuous integration tools allow automatic

testing of newly integrated code, identifying any problems or compatibility conflicts.

- **Continuous Delivery**: In this phase, software is made ready for delivery to the customer or for release to production. Using automation approaches and infrastructure as code, the software distribution and release process is automated to reduce errors and speed up the delivery time.
- **Monitoring and feedback**: Once the software is in production, operations teams monitor the performance and behavior of the application. Monitoring tools are used to collect data on application performance, errors, and usage. This feedback is used to continuously improve the software and plan any updates or fixes.
- **Gathering and continuous improvement**: DevOps promotes a continuous learning and improvement approach. Development and operations teams meet regularly to analyze results, discuss problems encountered, and identify ways to further optimize the software development and delivery process. The goal is to create a virtuous cycle of continuous improvement based on feedback and collaboration.



VI. Hacktempt's Architecture

The web-application was created by exploiting the combination of different programming languages, each aimed at certain purposes, such as php, javascript and html. It is connected to an external database through which the simulation of the "sql injection" vulnerability takes place.

As for the architecture of the web app, it can mostly be defined as clientside. In that a large amount of the data processing and user interface work takes place directly on the client device. Some key points of the client-side architecture include:

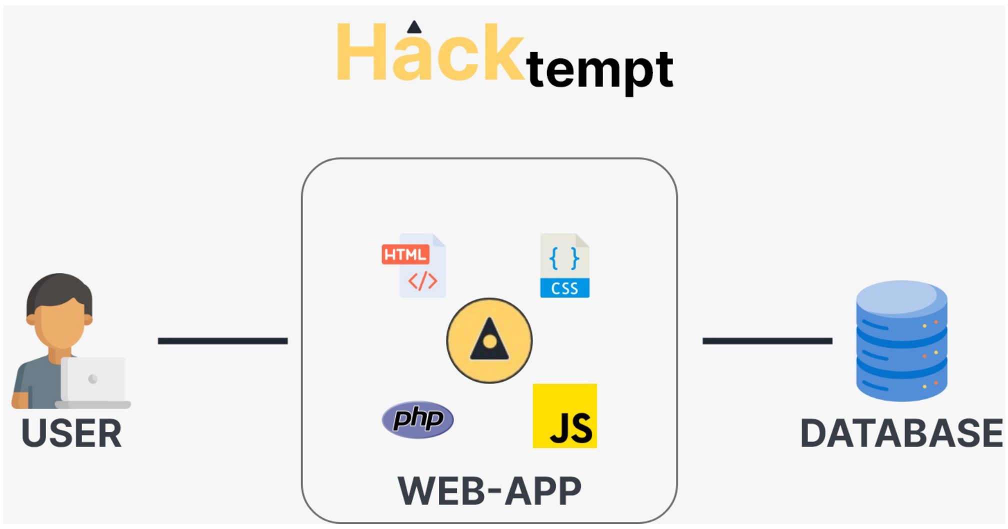
- **Local processing**: Data processing and UI management take place on the client device. The application runs in the browser or in a mobile application on the user's device.
- **Interactive user experience**: Because processing takes place on the client device, the user interface can be very responsive and interactive, providing a smooth user experience.

In these cases, HTML, CSS and javascript technologies are referred to for web development.

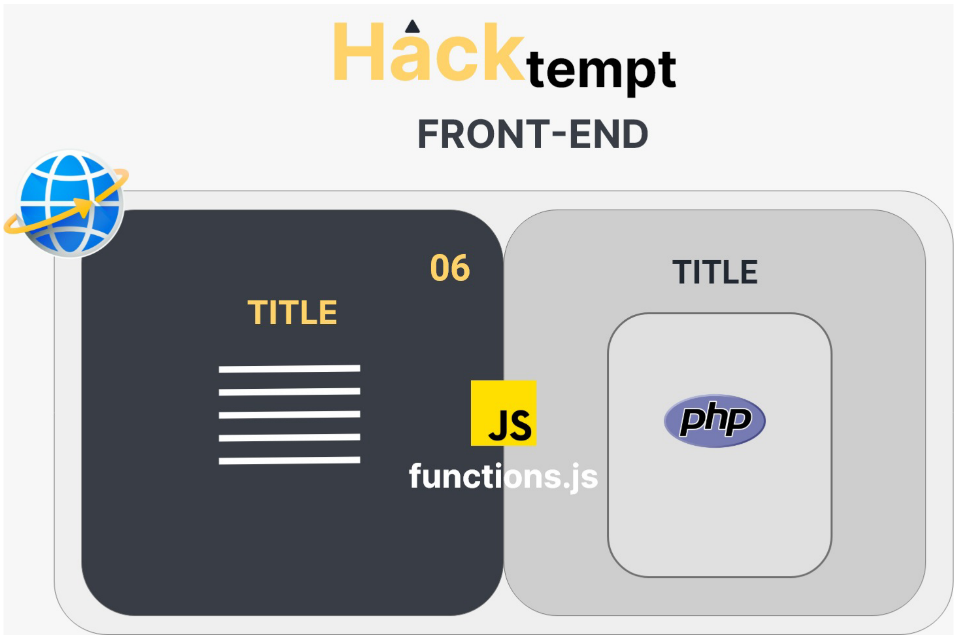
To a small extent it is also server-side, that is, when the server does the data processing. In fact, some key points of the server-side architecture include:

- **Data processing**: The server does the data processing, performing business logic and database access. The client sends requests to the server and receives responses containing the processed data.
- **Centralized scalability and security**: Since the server is the central point of data management, security, authentication and access control policies can be applied centrally. In addition, the server-side architecture allows the system to scale by adding resources to the server to handle more clients.

This is referred to when the PHP language and mySQL for the database come into play.



The structure of the front-end has been set up to be as dynamic as possible. In fact, it is divided into two parts: the left part with the list of all the steps that the user has completed, and the right part containing the actual part concerning the vulnerability to be analyzed. The function.js file is the "conductor" of how the front-end logic works, as it is responsible for displaying the content. It is based on the number of the step the user is in and, consequently, identifies the code that will be translated into front-end content to be shown.



The special feature is that the content of the page changes dynamically, changing only what is necessary according to the number of the step you are in. In fact, the latter is precisely the key element used by the increment() function, which, through the use of loadHtml() and loadFirst(), will govern the behavior of the app.
This type of design greatly facilitates future interventions or any additions to the code, given the configurability created.

Below is the code portion of the above functions:

- Increment function:

```
function increment() {
  var step = Number(document.getElementById("step").innerHTML);
  if (step<7) {
    step = step + 1;
  }
  if (step==7) {
    step=1;
    window.location.href = 'http://localhost/hacktempt/index.php';
  }
  document.getElementById("step").innerHTML = "0" + step;
  if (step > 1) {
    document.getElementById("nice").innerHTML = "";
    document.getElementById("title").innerHTML = "PASSED VULNERABILITIES";
  }
  switch (step) {
    case 2:
    console.log(step);
    document.getElementById("desc-left").innerHTML = "01 Authentication
Failures <br>";
    loadHTML('step02.html');
      break;
    case 3:
    console.log(step);
    document.getElementById("desc-left").innerHTML += "02 Broken Access
Control <br>";
    loadHTML('step03.html');
      break;
    case 6:
    console.log(step);
    document.getElementById("desc-left").innerHTML += "05 Phishing <br>";
    loadHTML('step06.html');
      break;
  }
}
```

- loadHtml function:

```
function loadHTML(filename){
  fetch(filename)
  .then(response=> response.text())
  .then(text=> document.getElementById('rightside').innerHTML = text);
}
```

- loadFirst function

```
function loadFirst(){
  loadHTML('step01.html');
}
```

the loadfirst() function is responsible for loading the first content to be displayed and from which to start the user experience.

VII. Identification and Authentication Failure

- **Description:** This vulnerability occurs when identification and authentication mechanisms fail to grant access only to authorized users. In other words, the application or system fails to properly verify the identity of users, allowing potential attackers to assume false identities or access restricted resources or functionality.

- **Causes:** There can be many causes of this vulnerability. One common cause is the use of weak or easy-to-guess passwords. Others include the lack of implementation of robust authentication mechanisms, such as two-factor authentication (2FA) or multi-factor authentication (MFA), which add an additional layer of security or the lack of implementations such that the available attempts to gain access are limited. In addition, failure to manage user credentials securely, such as storing passwords in plain text or transmitting them unencrypted, can also cause this vulnerability. Finally, lack of proper checks to verify user identity, such as failure to validate the email address or failure to verify authenticity, as well as failure to timeout sessions and invalidate sessions following logout.

- **Types of attacks:** Some attacks related to this vulnerability are Credential Stuffing which exploits a database of stolen passwords to try to gain access to a system or resource, Brute Force Attack in which various username/password combinations are recursively tested, Network Sniffing the use of the user's session ID exposed in the URL, and Session Fixation by which the attacker steals a user's session ID and is able to access the authenticated session by being able to perform actions "on behalf of the unwitting user."

- **Possible solutions:** To mitigate this vulnerability, it is important to implement secure authentication methods. Two-factor authentication (2FA) can provide an additional layer of security by requiring users to provide additional information or complete a second verification to access the account. It is also advisable to use complex and unique passwords for each account, avoiding easy-to-understand words and using a combination of uppercase and lowercase letters, numbers, and special characters. Password management policies can be implemented to require users to use passwords that meet certain complexity criteria and to change them periodically. It is critical to protect user credentials, such as by encrypting passwords during storage and transmission. In addition, it is important to monitor access activity to detect potential suspicious behavior, such as repeated login attempts or access from unusual geographic locations, in order to prevent unauthorized access.

- **Application code:**

```
function script01(){
    var username = document.getElementById("username-input").value;
    var password = document.getElementById("password-input").value;
    if (username == "admin" && password == "admin") {
        document.getElementById("wya").innerHTML = "ADMIN";
    } else document.getElementById("wya").innerHTML = "YOU'RE NOT AN ADMIN";
}
```

In the specific case of our web application, the "Identification and Authentication Failure" vulnerability is related to the absence of controls on the type of user registration credentials. Specifically, no limitations are set regarding minimum length, presence of upper and lower case letters, numbers or special characters.
This flaw therefore allows users to register, and subsequently log in to the platform, using default credentials (such as "admin/admin"), which are insecure and easily retrieved by a malicious user via a Brute Force Attack.
This vulnerability can lead to serious consequences, allowing the attacker to gain unauthorized access to the platform, exposing sensitive user data or allowing them to perform malicious activities.

- **Non-vulnerable code version:**

```
const bcrypt = require('bcryptjs');

// Funzione per validare l'input dell'utente
function validateInput(username, password) {
  if (username.length === 0 || password.length === 0) {
    return false;
  }

  const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$/;

  return passwordRegex.test(password);
}

// Funzione per autenticare l'utente
function authenticateUser(username, password) {
  if (!validateInput(username, password)) {
    document.getElementById("wya").innerHTML = "Invalid input.";
    return;
  }

  // Esempio di confronto sicuro della password memorizzata utilizzando
bcrypt: 'hashedPassword' rappresenta l'hash della password memorizzato nel
tuo database, restituisce true se la password fornita corrisponde all'hash
memorizzato, altrimenti false
  const hashedPassword =
"$2a$10$DYTkH8NhJHT/iVV3kA81eubCXWWZXHqBx/4K33dREjf6I0bZG7bVe"; //

  if (username === "admin" && validatePassword(password, hashedPassword)) {
    document.getElementById("wya").innerHTML = "ADMIN";
  } else {
    document.getElementById("wya").innerHTML = "YOU'RE NOT AN ADMIN";
  }
}

// Funzione per validare la password utilizzando bcrypt
function validatePassword(password, hashedPassword) {
  return bcrypt.compareSync(password, hashedPassword);
}
```

This code is intended to provide a basic implementation for validating user input and authentication using bcrypt:

1. **“function validateInput(username, password) { ... }”:** This function is used to validate user input, specifically checking if the username and password are not blank and checking if the password meets the criteria specified by the regular expression passwordRegex. It returns true if the input is valid, otherwise it returns false.

2. **“function authenticateUser(username, password) { ... }”:** This function handles the user authentication process.
   It checks whether the user's input is valid by calling the function validateInput:
   - If the input is invalid, the content of the HTML element with the id 'wya' is set to 'Invalid input.' and the function is aborted
   - If the input is valid, the username and password combination is compared with an example password stored in the hashedPassword hash. If the combination matches, the content of the HTML element with the id 'wya' is set to 'ADMIN', otherwise it is set to 'YOU'RE NOT AN ADMIN'.

3. **“function validatePassword(password, hashedPassword) { ... }”:** This function uses the bcrypt library to compare the password supplied by the user with the hashed password for that user, which is stored in the database. Before proceeding to the comparison, the 'compareSync' method takes care of calculating the hash corresponding to the password entered from the front-end by the user using the same hashing algorithm.
   It returns true if the supplied password matches the stored hash, otherwise it returns false.

VIII. Broken Access Control

- **Description:** This vulnerability occurs when access controls are not properly implemented or are not adequate to restrict access to resources or functionality to only authorized users. This can allow attackers to gain unauthorized privileges or access sensitive data, compromising the security of the application or system.

- **Causes:** The main causes of this vulnerability are lack of authorization checks, according to which the application does not adequately verify user privileges for access to specific resources, lack of user role control that allows normal users to access functionality reserved for administrators. Then we have the lack of input validation that allows attackers to enter malicious or invalid data by circumventing access controls, the various configuration errors that create flaws in the system, the manipulation of the URL through which the attacker is able to bypass the controls, and the various injection flaws that allow malicious input. Finally, cross-site scripting (XSS) through which malicious scripts aimed at performing malicious actions such as stealing users' personal information can be executed.

- **Types of attacks:** One of the main attacks is the user ID attack, in which an attacker manipulates the ID in the URL or parameters to access other users' data or functionality. Another is privilege modification, specifically the attacker tries to obtain higher privileges that allow him or her to access resources hidden from normal users. The enumeration attack, on the other hand, consists of repeated attempts to guess or infer the URL of protected resources, and finally the session attack involves acquiring or impersonating valid sessions so that the attacker can access protected resources by pretending to be the user.

- **Possible solutions:** To mitigate this vulnerability, it is necessary to implement robust access controls based on principles such as role-based access control (RBAC) or attribute-based access control (ABAC). These approaches allow users' privileges to be effectively assigned and managed based on their roles or attributes. It is important to verify authentication and authorization for each request, ensuring that users are properly authenticated before granting them access. In addition, it is critical to enforce access restrictions at the URL or resource level, ensuring that only authorized users can access certain features or resources, and finally, it is recommended to perform regular security testing to identify potential access flaws and correct them in a timely manner.

- **Code example:**

```
class Order:
    def __init__(self, order_id, user_id):
        self.order_id = order_id
        self.user_id = user_id
        self.is_completed = False


class OrderManagementSystem:
    def __init__(self):
        self.orders = []

    def find_order(self, order_id):
        for order in self.orders:
            if order.order_id == order_id:
                return order
        return None

    def create_order(self, order_id, user_id):
        order = Order(order_id, user_id)
        self.orders.append(order)
        print(f"Ordine {order_id} creato per l'utente {user_id}")

    # VULNERABILE: Completa l'ordine senza verificare l'utente
    def complete_order(self, order_id, user_id):
        order = self.find_order(order_id)
        if order is None:
            print(f"Ordine {order_id} completato")
        else:
            order.is_completed = True
            print(f"Ordine {order_id} completato")
```

The script provided represents a code implementation for the management of a company warehouse.
In particular, it is possible, by means of special functions, to create (save them within a list) or complete orders.
However, the code proves to be vulnerable to Broken Access Control as it does not implement the correct controls for the completion of orders in two different situations:

1. When a user other than the one who created the order (and therefore unauthorised) attempts to complete it: the code does not perform the necessary checks on user roles and authorisation, allowing unauthorised users to complete orders.

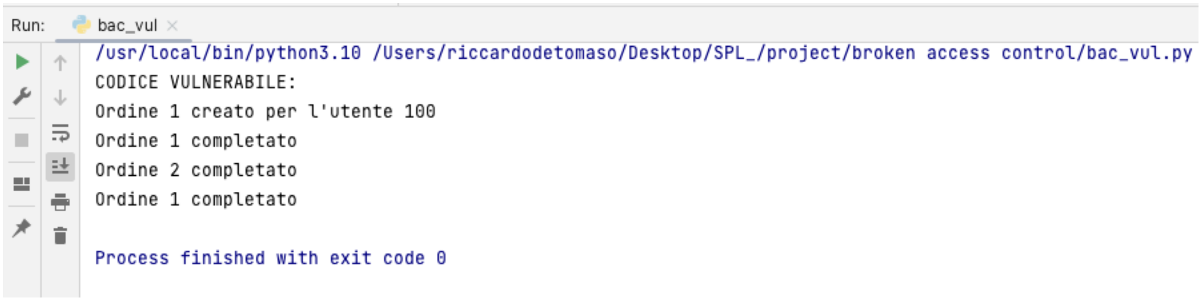
2. When any user attempts to complete an order that was never created: the code does not handle this situation correctly due to configuration errors.

- **Non-vulnerable code:**

```
# CORRETTO: Verifica l'utente prima di completare l'ordine
def complete_order(self, order_id, user_id):
    order = self.find_order(order_id)
    if order is None:
            print(f"Ordine {order_id} non trovato")
        elif order.user_id != user_id:
            print(f"Accesso negato: l'utente {user_id} non può completare
    l'ordine {order_id}")
        else:
            order.is_completed = True
            print(f"Ordine {order_id} completato")
```
Through the following lines of code I am going to print the results obtained from both codes:

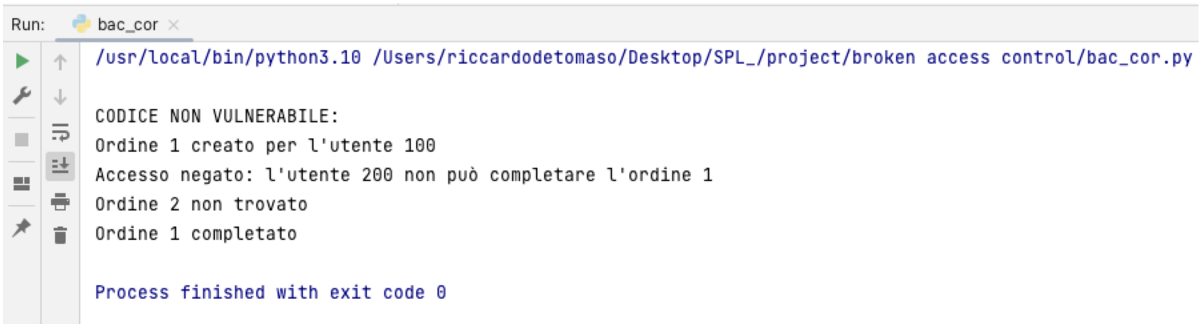
```
    system = OrderManagementSystem()

    system.create_order(1, 100)  # Creazione di un ordine
    system.complete_order(1, 200)  # Tentativo completamento ordine da
                                    utente non autorizzato
    system.complete_order(2, 100)  # Tentativo completamento ordine
                                    inesistente
    system.complete_order(1, 100)  # Completamento ordine da utente
                                    autorizzato
```

VULNERABLE CODE OUTPUT:



```
/usr/local/bin/python3.10 /Users/riccardodetomaso/Desktop/SPL_/project/broken access control/bac_vul.py
CODICE VULNERABILE:
Ordine 1 creato per l'utente 100
Ordine 1 completato
Ordine 2 completato
Ordine 1 completato

Process finished with exit code 0
```

NON-VULNERABLE CODE OUTPUT:



```
/usr/local/bin/python3.10 /Users/riccardodetomaso/Desktop/SPL_/project/broken access control/bac_cor.py

CODICE NON VULNERABILE:
Ordine 1 creato per l'utente 100
Accesso negato: l'utente 200 non può completare l'ordine 1
Ordine 2 non trovato
Ordine 1 completato

Process finished with exit code 0
```

It can be seen that, in the case of the vulnerable code, no check is made on the correspondence between the user who created the order and the user who is attempting to complete it, and the situation in which the order to be completed has never been created (it is not present within the "orders" list) is not properly handled.

IX. Cryptographic Failures

- **Description:** This vulnerability occurs when there are errors or weaknesses in the cryptographic mechanisms used to protect sensitive data, allowing attackers to compromise the confidentiality or integrity of the information. Encryption is a fundamental aspect of computer security and is used to protect sensitive data from unauthorized access or unwanted modification.

- **Causes:** The causes of this vulnerability can stem from several sources. One common cause is the use of weak or outdated cryptographic algorithms such as DES (Data Encryption Standard), rather than algorithms such as AES (Advanced Encryption Standard) or RSA (Rivest-Shamir-Adleman). Another cause can be the incorrect implementation of cryptographic protocols related to the omission of critical steps in the encryption sequence or the use of incorrect parameters that can weaken the effectiveness of the encryption. Another key issue is that of cryptographic keys; poor management or failure to protect them can allow attackers to gain access and decrypt encrypted data. Finally, the use of insecure communication protocols, such as HTTP instead of HTTPS, can expose data to risk of interception or manipulation.

- **Types of attacks:** The main attack is Brute Force , in which attackers try to break a cryptographic system by trying all possible combinations of keys. Others are chosen-plaintext or chosen-ciphertext, in which attackers can choose the plaintext or ciphertext to use during the attack. This allows them to obtain cryptographic key information or perform cryptanalysis attacks to compromise the system. In contrast, the oracle padding attack exploits vulnerabilities in the server's handling of error messages when invalid padding is provided, going on to obtain information about the cryptographic key. Finally, the key management attack targets vulnerabilities present in the management process (generation, storage, or distribution) of cryptographic keys.

- **Possible solutions:** To mitigate this vulnerability, it is important to use strong and reliable cryptographic algorithms that comply with recognized security standards. It is critical to adopt cryptographic algorithms that do not have known weaknesses and that can withstand decryption attacks. In addition, it is crucial to use secure communication protocols such as HTTPS to protect data transmissions over insecure networks by ensuring that data is encrypted during communication. Proper management of cryptographic keys is equally important. This includes proper key protection, use of secure key generation methods, periodic key rotation, and proper key storage and distribution. It is also essential to implement certificate and security update management procedures to ensure that cryptographic certificates are valid and that the algorithms and protocols used are always up to date with the latest security patches.

- **Application Code:**

```
// Esempio di utilizzo
/*let plaintext = "HacktemptSPL";
let shift = 3;

let ciphertext = encrypt(plaintext, shift);
console.log("Testo cifrato:", ciphertext);

let decryptedText = decrypt(ciphertext, shift);
console.log("Testo decifrato:", decryptedText);*/
function script03(){
    var input = document.getElementById('token-input');
    var token = 'KdfnwhpswVSO';
    if (input.value == decrypt(token, 3)) {
        document.getElementById('wya').innerHTML = 'TOKEN DECRYPTED
CORRECTLY';
    } else document.getElementById('wya').innerHTML = 'TOKEN
DECRYPTED NOT CORRECTLY'
}
```

The web application has serious vulnerabilities related to the use of an extremely weak encryption algorithm such as Caesar's Cipher, which is based on a fixed shift of the letters of the alphabet, making it vulnerable to Brute Force attacks and other Cryptanalysis techniques.

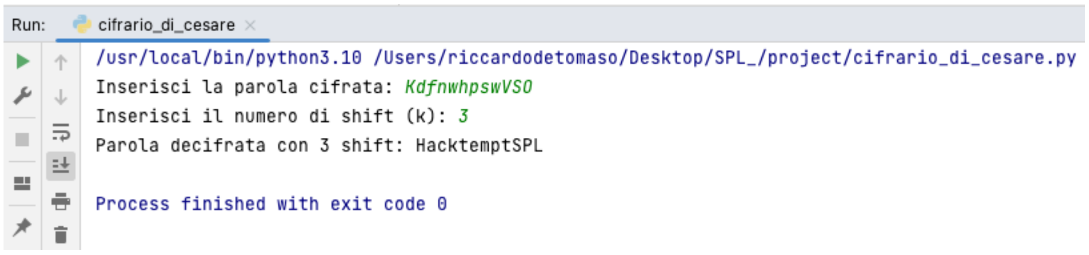
In this specific case, the vulnerability is even greater because the script for implementing Caesar's Cipher, which should be kept secret, is instead stored in plain text in the application's source code. This easily allows the attacker to trace the value of "k," which indicates the number of shifts performed to encrypt the text, and thus obtain the decrypted version of the token.

To demonstrate this, let us verify how, using a simple script in Python to decrypt Caesar's Cipher, it is possible to simply obtain the word to be decrypted.

```
def decifra_cesare(parola_cifrata):
    parole_decifrate = []

    # Prova tutte le 26 possibili chiavi di cifratura
    for chiave in range(1,26):
        parola_decifrata = ""

        # Decifra la parola con la chiave corrente
        for carattere in parola_cifrata:
            if carattere.isalpha():
                valore_carattere = ord(carattere)
                valore_carattere -= chiave

                if carattere.isupper():
                    if valore_carattere < ord('A'):
                        valore_carattere += 26
```

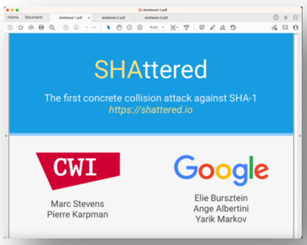
```python
                    elif valore_carattere > ord('Z'):
                        valore_carattere -= 26
                elif carattere.islower():
                    if valore_carattere < ord('a'):
                        valore_carattere += 26
                    elif valore_carattere > ord('z'):
                        valore_carattere -= 26

                parola_decifrata += chr(valore_carattere)
            else:
                parola_decifrata += carattere

        parole_decifrate.append(parola_decifrata)

    return parole_decifrate


parola_cifrata = input("Inserisci la parola cifrata: ")
shift = int(input("Inserisci il numero di shift (k): "))

parola_decifrata = decifra_cesare(parola_cifrata, shift)
print(f"Parola decifrata con {shift} shift: {parola_decifrata}")
```

OUTPUT:

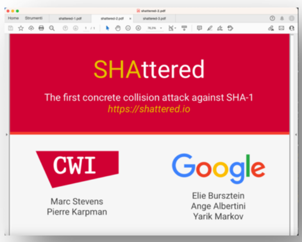


```
/usr/local/bin/python3.10 /Users/riccardodetomaso/Desktop/SPL_/project/cifrario_di_cesare.py
Inserisci la parola cifrata: KdfnwhpswVSO
Inserisci il numero di shift (k): 3
Parola decifrata con 3 shift: HacktemptSPL

Process finished with exit code 0
```

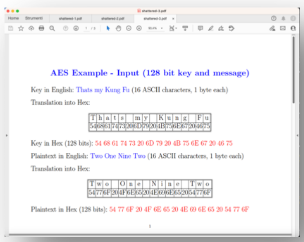
- **Code example:** the following example highlights another cause that generates Cryptographic Failure, namely the use of weak hash algorithms (ex. SHA-1). It will be seen how, by taking three different pdf files and calculating their hashes, collision problems will be generated with SHA-1 while this will not happen with the stronger and more robust variant (SHA-256).



shattered-1.pdf



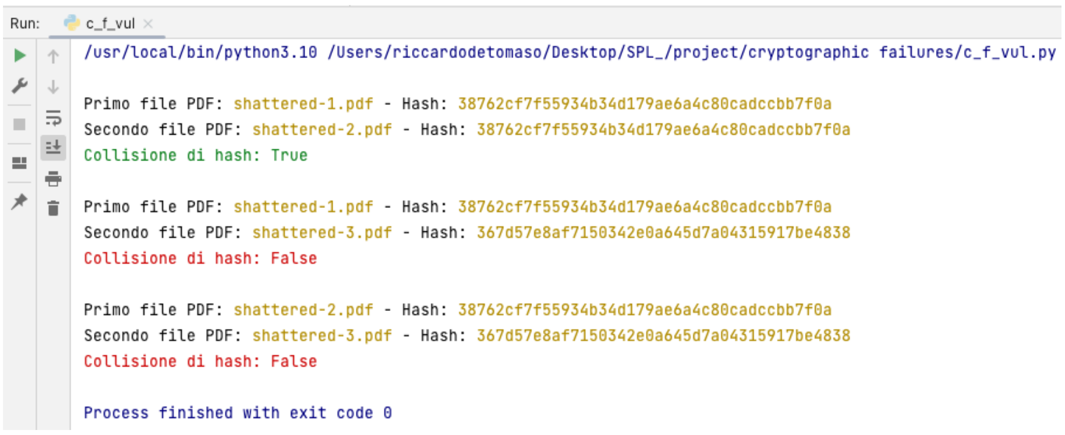
shattered-2.pdf



shattered-3.pdf

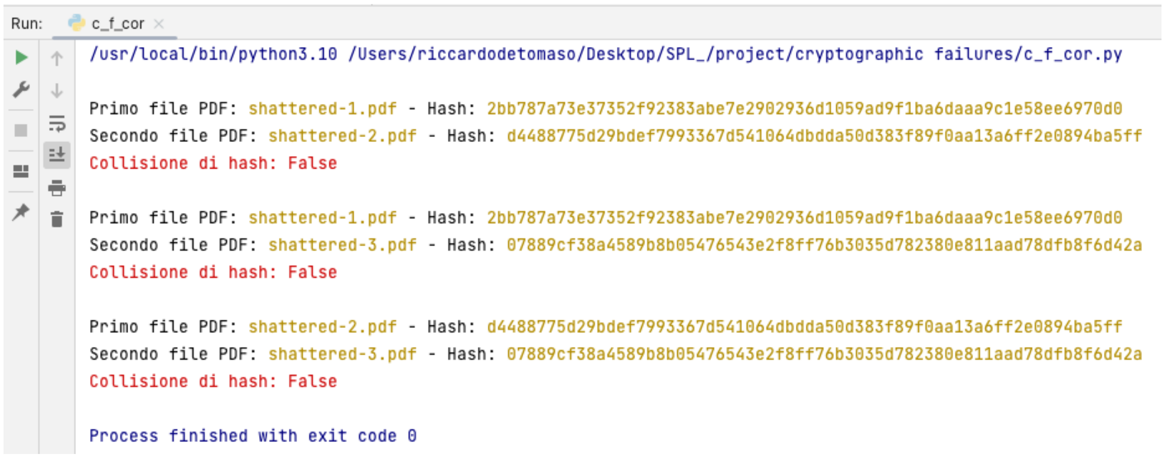
```python
import hashlib

def calculate_file_hash(file_path):
    with open(file_path, "rb") as file:
        data = file.read()
        return hashlib.sha1(data).hexdigest()

pdf_files = ["shattered-1.pdf", "shattered-2.pdf", "shattered-3.pdf"]

hash_1 = calculate_file_hash(pdf_files[0])
hash_2 = calculate_file_hash(pdf_files[1])
hash_3 = calculate_file_hash(pdf_files[2])
print(f"\nPrimo file PDF: {pdf_files[0]} - Hash: {hash_1}")
print(f"Secondo file PDF: {pdf_files[1]} - Hash: {hash_2}")
print(f"Terzo file PDF: {pdf_files[2]} - Hash: {hash_3}")

def print_collision_result(file1, file2, collision):
    print(f"Collisione di hash tra il {file1} e il {file2} file:
{collision}")

print_collision_result("primo", "secondo", hash_1 == hash_2)
print_collision_result("primo", "terzo", hash_1 == hash_3)
print_collision_result("secondo", "terzo", hash_2 == hash_3)
```

VULNERABLE CODE OUTPUT (SHA-1):



```
/usr/local/bin/python3.10 /Users/riccardodetomaso/Desktop/SPL_/project/cryptographic failures/c_f_vul.py

Primo file PDF: shattered-1.pdf - Hash: 38762cf7f55934b34d179ae6a4c80cadccbb7f0a
Secondo file PDF: shattered-2.pdf - Hash: 38762cf7f55934b34d179ae6a4c80cadccbb7f0a
Collisione di hash: True

Primo file PDF: shattered-1.pdf - Hash: 38762cf7f55934b34d179ae6a4c80cadccbb7f0a
Secondo file PDF: shattered-3.pdf - Hash: 367d57e8af7150342e0a645d7a04315917be4838
Collisione di hash: False

Primo file PDF: shattered-2.pdf - Hash: 38762cf7f55934b34d179ae6a4c80cadccbb7f0a
Secondo file PDF: shattered-3.pdf - Hash: 367d57e8af7150342e0a645d7a04315917be4838
Collisione di hash: False

Process finished with exit code 0
```

NON-VULNERABLE CODE OUTPUT (SHA-256):



```
/usr/local/bin/python3.10 /Users/riccardodetomaso/Desktop/SPL_/project/cryptographic failures/c_f_cor.py

Primo file PDF: shattered-1.pdf - Hash: 2bb787a73e37352f92383abe7e2902936d1059ad9f1ba6daaa9c1e58ee6970d0
Secondo file PDF: shattered-2.pdf - Hash: d4488775d29bdef7993367d541064dbdda50d383f89f0aa13a6ff2e0894ba5ff
Collisione di hash: False

Primo file PDF: shattered-1.pdf - Hash: 2bb787a73e37352f92383abe7e2902936d1059ad9f1ba6daaa9c1e58ee6970d0
Secondo file PDF: shattered-3.pdf - Hash: 07889cf38a4589b8b05476543e2f8ff76b3035d782380e811aad78dfb8f6d42a
Collisione di hash: False

Primo file PDF: shattered-2.pdf - Hash: d4488775d29bdef7993367d541064dbdda50d383f89f0aa13a6ff2e0894ba5ff
Secondo file PDF: shattered-3.pdf - Hash: 07889cf38a4589b8b05476543e2f8ff76b3035d782380e811aad78dfb8f6d42a
Collisione di hash: False

Process finished with exit code 0
```

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash algorithm belonging to the SHA-2 family of algorithms. It therefore offers greater resistance to hash collisions than SHA-1 for two reasons:

1. Digest size: SHA-1 produces a digest of 160 bits, while SHA-256 produces a digest (output) of 256 bits. A larger digest size provides a large output space, increasing the probability of avoiding random collisions.

2. Algorithm structure: SHA-1 uses a compression structure based on less secure hash functions than SHA-256.

X. SQL Injection

- **Description:** This vulnerability occurs when a web application does not properly handle input provided by users. SQL is a language used to communicate with databases, and SQL injection exploits gaps in input validation procedures to insert or execute malicious code within queries sent to the database.
  SQL injection can allow attackers to obtain sensitive information from the database, modify or delete data, perform unauthorized operations, or take complete control of the system.

- **Causes:** The causes of this vulnerability can stem from several sources. The use of unparameterized or unprepared SQL queries is a common cause. This occurs when SQL queries are constructed by directly concatenating unfiltered input strings, allowing attackers to insert malicious SQL code within the input. The omission of input validation or unfiltered input within queries allows attackers to inject characters or sequences that can alter the meaning of SQL queries and lead to unwanted database manipulation. Using unsafe dynamic SQL constructs, such as creating SQL queries based on user input without proper precautions, can expose the application to SQL injection risks. Finally, direct and unsupervised access to query strings, without proper sanitization of input data, can allow attackers to manipulate queries and gain unauthorized access to data.

- **Types of attacks:** One of the main attacks is commenting attacks in which the attacker uses SQL comments such as "--" to ignore the rest of the query and insert malicious code without detection. Another type of attack is based on the UNION operator, where attackers try to combine the result of a malicious query with the results of a legitimate query. As for error-based attacks, they exploit error messages generated by SQL queries to obtain information about the database while Boolean-based attacks exploit Boolean statements such as "OR 1=1" in queries to extract information from the database. Finally, bypass attacks aim to bypass security measures implemented to protect against SQL Injection attacks and gain unauthorized access to the database.

- **Possible solutions:** To mitigate this vulnerability, it is critical to use parameterization or prepared statements in SQL queries. These mechanisms allow SQL commands to be separated from input data, preventing the direct inclusion of user input in queries. Also, it is important to avoid building dynamic queries by concatenating unfiltered input strings, as this opens the door to SQL injection. Implementing validation and filtering of input data is another essential step. This can include the use of escaping functions or libraries that remove or neutralize special characters that can alter the meaning of queries. It is advisable to use secure libraries and frameworks that offer SQL injection prevention features, such as the use of default parameterized queries. Also, adopt secure development practices such as separating database privileges, limiting application access to only necessary operations, and conduct regular security testing, such as penetration testing, to identify potential vulnerabilities and correct them in a timely manner.

- **Application code:**

```
if (isset($_POST['email'])) {
    $email = $_POST['email'];
    $user = $conn->query("SELECT * FROM users WHERE email='{$email}'");

    while ($row = $user->fetch_assoc()) {
        die("USERNAME: " .$row["username"] ."<br> <button type='button'
name='button' class='next' onclick='increment()' id='next-
button'>NEXT></button>");
    }
}
```

The web application is vulnerable to SQL Injection, especially in this section the goal is to derive the username related to the entered email.
What happens is that by copying the string `'OR '1'='1` instead of the email, the system still shows the username since that is the starting query:

```
query("SELECT * FROM users WHERE email='{$email}'");
```

so, since condition 1=1 is always true, the result is shown.

XI. Security Misconfiguration

- **Description:** This vulnerability occurs when systems are misconfigured or insecurely configured, potentially opening various security holes. Security misconfiguration can affect various aspects, such as web server configuration, authorization management, security policies, session management, and more.

- **Causes:** One of the main factors is failure to apply security patches or updates, which can leave known vulnerabilities open. The use of unmodified default configurations can be a problem, as these configurations are often publicly known and can be easily exploited by attackers. The omission of proper security controls or permissions can allow attackers to access resources or functionality that should be restricted to authorized users only. Exposure of sensitive information through configuration errors can put data confidentiality at risk. Finally, incorrect implementation of firewalls and security policies can compromise the system's ability to defend itself and make resources vulnerable to attack.

- **Types of attacks:** Major attacks include directory traversal, which involves accessing file system directories that should not be accessible, and unauthorized access to resources, which allows unauthorized users to access resources restricted to certain users or roles. Two other attacks are related to unencrypted exposure of sensitive information such as passwords, encryption keys, or users' personal information, and the use of weak or default configurations (default passwords, weak cryptographic algorithms, or outdated TLS configurations).

- **Possible solutions:** To mitigate this vulnerability, it is critical to follow security instructions when configuring systems. This includes regularly applying security patches and updates to correct known vulnerabilities. It is also important to remove or disable unnecessary default configurations, as they can be exploited by attackers. Proper implementation of security controls such as authentication, authorization, and encryption are essential to protect resources. Firewalls and security rules must be properly configured to filter unwanted traffic and protect resources from unauthorized access attempts. Using configuration scanning tools can help identify potential security vulnerabilities and allow you to make necessary corrections. Finally, it is important to conduct regular security tests and evaluations of configurations to identify potential security holes and ensure that configurations are aligned with security rules.

XII. Phishing

- **Description:** Phishing is a sophisticated social engineering attack technique that aims to exploit users' trust to obtain sensitive information, such as passwords or login details. Attackers pretend to be trusted entities or well-known online services, creating fake websites or emails that look authentic in order to trick people into revealing confidential information.

- **Causes:** There are many causes of this threat, first and foremost the inexperience or lack of awareness of users, who are unable to recognize the signs of a fraudulent email or website and therefore give out their information without suspecting anything. Attackers create websites or emails that closely mimic legitimate entities or services, making them difficult to distinguish from real ones. Other causes include ineffective spam filters that do not properly distinguish legitimate emails from fraudulent ones, or the use of spoofing techniques (a technique of manipulating the source IP address of a network packet so that it appears to come from another source).

- **Types of attacks:** One of the most common phishing attacks is e-mail phishing; attackers send fraudulent e-mails that appear to be from legitimate organizations and convince people to provide their login credentials or other personal information. Another type of attack is smishing, which is done through SMS. Vishing, on the other hand, is a phishing attack that occurs through phone calls. Finally, we have phishing spear in which attackers collect specific information about a person and use it to create personalized e-mails or messages, and pharming in which vulnerabilities in DNS servers are exploited by manipulating DNS records to direct people to counterfeit websites.

- **Possible solutions:** To protect against this vulnerability, it is essential to educate users about identifying phishing e-mails and online scams. In addition, using phishing filters can block most incoming phishing e-mails, providing a basic defense against this type of attack. Keeping software and browsers up to date at all times is another key measure, as security patches often fix vulnerabilities that could be exploited by attackers. Using multi-factor authentication tools adds another layer of protection for sensitive user data by requiring the use of a second form of verification in addition to a password.

- **Application Code:**

```
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Salva le informazioni nel file di testo
    $file = 'dati.txt';
    $data = "Username: $username\nPassword: $password\n\n";

    if (file_put_contents($file, $data, FILE_APPEND | LOCK_EX) !==
false) {
        echo 'Informazioni salvate con successo!';
    } else {
        echo 'Si è verificato un errore durante il salvataggio delle
informazioni.';
    }
}
?>
```

The web application has a phishing vulnerability that allows an unwitting user to enter their login credentials on a malicious site that has a graphical interface identical to Instagram. Through a .php code, username and password information is captured and stored in the 'data.txt' text file, which is accessible to the attacker. Subsequently, this information is used to redirect the user to the official Instagram website, providing them with access to their profile.

XIII. Restrict URL Access

- **Description:** This vulnerability occurs when web applications do not properly implement the necessary controls to restrict access to specific resources or pages. This opens the door for unauthorized users, allowing them to access restricted content or exploit critical functionality that should only be accessible to authorized individuals.

- **Causes:** The causes of this vulnerability can be several. First is the common omission that is the lack of authorization checks at the URL level, i.e., applications do not check whether the user has the appropriate privileges to access a given resource or page. Then we have the incorrect implementation of access controls that can allow unauthorized people to circumvent restrictions. In addition, the absence of access restrictions based on user roles or privileges can create security holes. Finally, errors in the creation of application configuration files that can open gaps that allow unauthorized access.

- **Types of attacks:** Among the main attacks is URL Manipulation in which attackers modify the URL or its parameters to gain access to restricted resources or web pages. URL Enumeration, on the other hand, involves systematically searching for hidden URLs or private resources to discover sensitive information. Finally, the File Inclusion attack in which attackers exploit the dynamic inclusion functionality of external files without proper validation or restriction to gain access to sensitive resources.

- **Possible solutions:** To mitigate this vulnerability, it is critical to implement proper access controls at the URL or resource level. This involves verifying authentication and authorization for each request, ensuring that only users with the correct privileges can access critical content or functionality. In addition, it is important to implement access restrictions based on user roles or privileges, so that only those who have the right to access certain data or functions can do so. The use of secure frameworks or libraries that provide access management capabilities can be of great help in ensuring proper protection. In addition, it is essential to carefully implement application configuration files to set the necessary access restrictions.

- **Application code:** The web application has a vulnerability related to URL access restriction. A malicious user, having access to the source code, is able to make changes to the script and make the "END" button visible on the front-end, which allows the user to return to the home page. This vulnerability is the result of a misconfiguration that allows the visibility flag, initially set to "hidden," to be changed:

```
<button type="button" name="button" class="next" onclick="increment()"
id="next-button"
        style="visibility:hidden;">END
</button>
```

The vulnerability can be exploited by modifying the code as follows:

```
<button type="button" name="button" class="next" onclick="increment()"
id="next-button"
        style="visibility:visible;">END
</button>
```

In this way, the button becomes visible on the screen.

XIV. Conlcusion

In conclusion, this paper examined several security vulnerabilities that can put information systems at risk and compromise the protection of sensitive data. These vulnerabilities include problems with identification and authentication, access control, encryption, SQL injection, security misconfiguration, unauthorized URL access, and phishing.

It is critical to understand and address these vulnerabilities to ensure the security of systems. Implementing appropriate security measures, such as proper identity and authentication management, strict access control, use of robust cryptographic algorithms, secure configuration of systems, prevention of SQL injection, protection of URLs, and user awareness of phishing threats, is essential to mitigate risks and protect sensitive data.

In addition, it is important to adopt a continuous security mindset, implementing risk management processes and constant monitoring of systems to identify and respond to new threats in a timely manner. Collaboration between security experts, developers and end users is crucial to ensure overall system security.

Investing in cybersecurity and adopting recommended best practices are critical steps to protect information systems and preserve user confidence in using digital assets. Only through a focused approach to security and continuous efforts to improve protection measures can the growing complexity of cyber threats be effectively addressed and the associated risks mitigated in a way that ensures the confidentiality of information and preserves the trust of users and stakeholders.

Bibliographical references

«A01 Broken Access Control - OWASP Top 10:2021». https://owasp.org/Top10/A01_2021-Broken_Access_Control/.

«A02 Cryptographic Failures - OWASP Top 10:2021». https://owasp.org/Top10/A02_2021-Cryptographic_Failures/.

«A03 Injection - OWASP Top 10:2021». https://owasp.org/Top10/A03_2021-Injection/.

«A05 Security Misconfiguration - OWASP Top 10:2021». https://owasp.org/Top10/A05_2021-Security_Misconfiguration/.

«A07 Identification and Authentication Failures - OWASP Top 10:2021». https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/.

«Access control vulnerabilities and privilege escalation | Web Security Academy». https://portswigger.net/web-security/access-control.

Baazaoui, Bilel. «Vulnerabilità Cryptographic Failures, che cos'è?» *Onorato Informatica Srl* (blog) https://www.onoratoinformatica.it/vulnerabilita-informatiche/vulnerabilita-cryptographic-failures-di-che-cosa-si-tratta/.

«Vulnerabilità di Identification and Authentication Failures, che cos'è». *Onorato Informatica Srl* (blog). https://www.onoratoinformatica.it/vulnerabilita-informatiche/vulnerabilita-di-identification-and-authentication-failures-che-cose/.

Banu. «What Is Broken Access Control Vulnerability?» *Cybersecurity Exchange* (blog), 12 ottobre 2022. https://www.eccouncil.org/cybersecurity-exchange/web-application-hacking/broken-access-control-vulnerability/.

Cisco. «Cos'è il phishing? Esempi di attacchi di phishing e definizione». https://www.cisco.com/c/it_it/products/security/email-security/what-is-phishing.html.

Cyolo. «Identification And Authentication Failures And How To Prevent Them». https://cyolo.io/blog/identification-and-authentication-failures-and-how-to-prevent-them.

«Failure to Restrict URL Access - Enterprise Security | Montana State University». https://www.montana.edu/uit/security/web/failure-to-restrict-url-access.html.

«How to Prevent Broken Access Control Vulnerability». https://www.eccouncil.org/cybersecurity-exchange/web-application-hacking/broken-access-control-vulnerability/.

iBarry. «Phishing». https://www.ibarry.ch/it/rischi-di-internet/phishing/.

Malwarebytes. «Cos'è il phishing? Tipi di phishing ed esempi».

https://it.malwarebytes.com/phishing/.

RHC, Redazione. «Che cos'è il Broken Access Control? Spiegazione della vulnerabilità più diffusa al mondo». Red Hot Cyber, 15 settembre 2022. https://www.redhotcyber.com/post/che-cose-il-broken-access-control-spiegazione-della-vulnerabilita-piu-diffusa-al-mondo/.

«SQL Injection». https://www.w3schools.com/sql/sql_injection.asp.
Veracode. «Failure to Restrict URL Access». https://www.veracode.com/security/failure-restrict-url-access.

«Cos'è la metodologia agile?». https://www.redhat.com/it/devops/what-is-agile-methodology.

Atlassian. «What Is Agile?» Atlassian. https://www.atlassian.com/agile.

«What Is Agile Methodology in Project Management?» https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/.

Software Quality. «What Is the Waterfall Model? - Definition and Guide». https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model.

«What is Waterfall Methodology? – Forbes Advisor». https://www.forbes.com/advisor/business/what-is-waterfall-methodology/.

Team, Adobe Communications. «Waterfall Methodology: Project Management | Adobe Workfront». https://business.adobe.com/blog/basics/waterfall.