A01:2021 – BROKEN ACCESS CONTROL

Descrizione breve

Una stessa applicazione può essere utilizzata in modo diverso da utenti aventi ruoli differenti.

Il **controllo degli accessi** permette di verificare se un utente è autorizzato ad usufruire di una certa funzionalità dell'applicazione.

Rischi associati all'assenza di controlli degli accessi possono essere la divulgazione di informazioni non autorizzate, la modifica o la distruzione di tutti i dati.

Codice vulnerabile

Il codice seguente utilizza dati non controllati in una query SQL per recuperare dal database le informazioni su un utente:

```
pstmt.setString(1,
request.getParameter("acct"));

ResultSet results =
pstmt.executeQuery( );
```

Scenario d'attacco

Supponendo che la query restituisca le informazioni associate al conto corrente avente come numero quello indicato dal parametro "acct", un malintenzionato potrebbe richiedere la visualizzazione di un conto corrente non proprio se non venisse verificato correttamente che il conto corrente a cui si sta tentando di accedere è il proprio:

```
https://example.com/app/accoun
tInfo?acct=notmyacct
```

A02:2021 – CRYPTOGRAPHIC FAILURES

Descrizione breve

La **crittografia** può essere utilizzata per proteggere i dati manipolati dall'applicazione.

Data in input una stringa (es. password dell'utente), gli algoritmi di crittografia permettono di **codificarla** in un'altra stringa usando una **chiave crittografica**. La stringa ottenuta può essere memorizzata, così da essere successivamente recuperabile (es. login dell'utente) e **decodificabile** nella sua forma originaria.

Codice vulnerabile

Il codice seguente legge una password da un file e la utilizza per connettersi a un database:

```
Properties prop = new
Properties();

prop.load(new
FileInputStream("config.proper
ties"));

String password =
Base64.decode(prop.getProperty
("password"));

DriverManager.getConnection(ur
l, usr, password);
```

Scenario d'attacco

Un dipendente malintenzionato potrebbe scoprire:

- L'algoritmo utilizzato per crittografare la password per accedere al database, cioè **Base64**;
- La stringa codificata dall'algoritmo di crittografia.

Considerando che Base64 è un **algoritmo insicuro** per crittografare le password (la codifica e decodifica non usano chiavi crittografiche, quindi sono automatiche una volta conosciuto l'algoritmo), il malintenzionato potrebbe risalire alla password del database e compromettere il sistema.

A03:2021 – INJECTION

Descrizione breve

È possibile **iniettare codice sorgente eseguibile** in un'applicazione sfruttandone le falle di sicurezza, come l'assenza di controlli sui dati forniti in input dall'utente.

Codice vulnerabile

Il codice seguente definisce una query attraverso la concatenazione di più stringhe, tra cui una fornita come parametro della richiesta HTTP:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario d'attacco

Immaginiamo che un malintenzionato fornisca come parametro in input la stringa:

```
mario-rossi'); DROP TABLE accounts;--
```

L'applicazione eseguirebbe prima una query al database per recuperare le informazioni dell'account con custID='mario-rossi', successivamente sarebbe costretta ad eseguire una query che cancellerebbe la tabella accounts.

Questa tipologia di injection prende il nome di **SQL injection.**

A04:2021 – INSECURE DESIGN

Descrizione breve

Progettare un'applicazione trascurandone la sicurezza implica l'implementazione di una soluzione con numerose vulnerabilità, poiché, già dal principio, non sono stati **concepiti** e **previsti** le misure necessarie per difendersi da malintenzionati.

Codice vulnerabile

Il codice seguente stampa le informazioni della traccia di esecuzione di un'eccezione in System.Err:

```
try {
  /* ... */
} catch(Exception e) {
  e.printStackTrace();
}
```

Scenario d'attacco

Un attaccante potrebbe fornire un input che genera una certa eccezione (ad esempio relativa all'interrogazione del database).

Se l'eccezione non viene propriamente gestita nel blocco catch, l'output che l'attaccante visualizzerebbe potrebbe contenere informazioni sensibili (come la struttura della query SQL, quindi del database, o informazioni private).

A05:2021 – SECURITY MISCONFIGURATION

Descrizione breve

La sicurezza di un'applicazione dipende dall'**interazione** di varie componenti software: una loro **configurazione** scorretta potrebbe compromettere la sicurezza dell'intero sistema.

Ad esempio, l'applicazione potrebbe essere rilasciata con abilitate funzionalità di **debugging** e un attaccante potrebbe sfruttarle per ottenere informazioni sul sistema.

Codice vulnerabile

Il codice seguente stampa le informazioni della traccia di esecuzione di un'eccezione in System.Err:

```
try {
  /* ... */
} catch(Exception e) {
  e.printStackTrace();
}
```

Scenario d'attacco

Un attaccante potrebbe fornire un input che genera una certa eccezione (ad esempio relativa all'interrogazione del database).

Se l'eccezione non viene propriamente gestita nel blocco catch, l'output che l'attaccante visualizzerebbe potrebbe contenere informazioni sensibili (come la struttura della query SQL, quindi del database, o informazioni private).

A06:2021 – VULNERABLE AND OUTDATED COMPONENTS

Descrizione breve

Un'applicazione è costituita da molteplici componenti software e la sicurezza dei singoli determina la sicurezza dell'intero sistema.

Per questo motivo, è importante avere consapevolezza delle **versioni** che ne si utilizzano, sapere se presentano vulnerabilità, se sono ancora **supportate** o se sono disponibili **aggiornamenti**.

Codice vulnerabile

Un server importa una versione delle librerie di Log4j in cui è stata trovata la vulnerabilità Log4Shell:

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.io.*;
import java.sql.SQLException;
import java.util.*;
```

Scenario d'attacco

Un attaccante invia una richiesta al server contenente dati non attendibili che vengono poi processati da Log4j.

La versione utilizzata di Log4j non effettua controlli sulla bontà dei dati ricevuti ed innesca un meccanismo per l'iniezione di codice remoto.

A07:2021 – IDENTIFICATION AND AUTHENTICATION FAILURES

Descrizione breve

Rischi legati all'**autenticazione** dell'utente e alla gestione delle **sessioni** possono essere causati dall'utilizzo di password deboli, processi insicuri per il recupero delle credenziali e l'invalidazione scorretta delle sessioni.

Codice vulnerabile

Il codice seguente utilizza una password codificata per connettersi a un database:

```
DriverManager.getConnection(url,
"scott", "tiger");
```

Scenario d'attacco

Un dipendente che ha accesso al codice sorgente può scoprire le credenziali per accedere in modo illecito al database. Inoltre, le stesse credenziali possono essere estratte dal bytecode dell'applicazione.

A08:2021 – SOFTWARE AND DATA INTEGRITY FAILURES

Descrizione breve

Un'applicazione può fare riferimento direttamente nel codice sorgente a componenti software e a dati provenienti da **fonti esterne**: è necessario verificarne l'integrità per evitare l'impiego di risorse malevole.

Codice vulnerabile

Il codice seguente incorpora un frammento di codice JavaScript da un content delivery network (CDN):

```
<script
src="https://cdnexample.com/script.js">
```

Scenario d'attacco

Un malintenzionato potrebbe corrompere il codice presente in https://cdnexample.com/script.js, cosicché, quando questo viene incorporato lato client, possa essere iniettato codice malevolo nell'applicazione.

A09:2021 – SECURITY LOGGING AND MONITORING FAILURES

Descrizione breve

Il **monitoraggio** delle attività che coinvolgono un'applicazione può essere effettuato attraverso la stampa di **messaggi di log**. Un loro corretto utilizzo può permettere di identificare prontamente errori e vulnerabilità del sistema (es. messaggi di errore, query eseguite al database, continui tentativi di login falliti).

Codice vulnerabile

Il codice seguente riporta un tentativo di autenticazione di un utente:

```
if LoginUser(){
  // Login successful
  RunProgram();
} else {
  // Login unsuccessful
  LoginRetry();
}
```

Scenario d'attacco

Se un tentativo di login fallisce, è possibile effettuare un nuovo tentativo. Un malintenzionato potrebbe provare e riprovare ad accedere al sistema cercando di indovinare le credenziali e, sfortunatamente, i tentativi falliti di login non verrebbero registrati, quindi nessuno si accorgerebbe dell'attacco subito.

A10:2021 – SERVER-SIDE REQUEST FORGERY

Descrizione breve

Il rischio di Server-Side Request Forgery (SSRF) si verifica ogni volta che un'applicazione web recupera una risorsa remota senza validare l'**URL** fornito dall'utente. Questo potrebbe permettere ad un malintenzionato di forzare l'applicazione ad inviare una richiesta malevola ad una **destinazione inattesa,** anche quando quest'ultima è protetta da un firewall o una VPN.

Codice vulnerabile

Il codice seguente riporta un tentativo di accesso ad un sito web:

```
URL url = new
URL(req.getParameter("url"));

HttpURLConnection conn =
(HttpURLConnection)
url.openConnection();
```

Scenario d'attacco

Consideriamo un'applicazione web che interroga un server per recuperare dei dati attraverso delle API. Per accedere a tali API è necessario specificare un URL (parametro "url").

Un malintenzionato potrebbe indicare http://localhost/admin come URL, per provare ad accedere al contenuto della cartella admin presente sullo stesso server dell'applicazione. Dato che la richiesta d'accesso sarebbe effettuata localmente (localhost), le normali misure di sicurezza verrebbero eluse.