CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

AUTENTICAZIONE, RESTRIZIONE DEGLI ACCESSI E CENNI DI SICUREZZA

Docente: prof. Romano Simone

a.a. 2024-2025

Introduzione

I due aspetti di sicurezza principali nelle applicazioni web sono:

- **Impedire agli utenti non autorizzati di accedere a dati sensibili**
  Questo processo prevede la **restrizione degli accesso** (identificando quali risorse necessitano di protezione e chi dovrebbe accedervi) e l'**autenticazione** (identificando gli utenti per determinare se sono uno di quelli autorizzati)
- **Impedire agli attaccanti di rubare dati di rete mentre questi sono in transito**
  Questo processo prevede l'uso di TLS (Transport Layer Security) o SSL (Secure Sockets Layer) per crittografare il traffico tra il browser e il server
  - HTTPS: HTTP su una connessione crittografata SSL/TLS
  - SSL è uno standard superato da TLS

Restrizione degli accessi

Gli approcci per le restrizione degli accessi sono gli stessi indipendentemente dal fatto che si utilizzi o meno TSL/SSL

Essi sono:

- **Restrizione degli accessi in maniera dichiarativa**
- **Restrizione degli accessi in maniera programmatica**

Restrizione degli accessi in maniera dichiarativa

- Con la restrizione degli accessi in maniera dichiarativa nessuna delle singole Servlet o pagine JSP necessita di alcun codice sensibile alla sicurezza
- La restrizione degli accessi è gestita dal Servlet Container (occorre dichiarare dei parametri di configurazione in **web.xml**)

Restrizione degli accessi in maniera programmatica

- Con la sicurezza programmatica, le Servlet e le pagine JSP garantiscono la restrizione degli accessi:
  - Per impedire l'accesso non autorizzato, ogni Servlet o pagina JSP protetta deve verificare che l'utente sia autenticato e che abbia il permesso per accedere alla Servlet o pagina JSP richiesta

Restrizione degli accessi: maniera dichiarativa vs. maniera programmatica

- Maniera dichiarativa
  - Vantaggio: le Servlet o pagine JSP protette non necessitano di alcun codice sensibile alla sicurezza
  - Svantaggio: dipendente dal Servlet container, ne consegue che le applicazioni Web non sono portabili
- Maniera programmatica
  - Vantaggio: la applicazioni Web sono portabili, l'approccio è flessibile
  - Svantaggio: il programmatore deve scrivere del codice per proteggere le Servlet o le pagine JSP che necessitano di protezione

Restrizione degli accessi: linee guida

1. Identificare quali Servlet e pagine JSP necessitano di protezione
2. Identificare chi può accedere ad un gruppo di Servlet e pagine JSP
   - Generalmente vi sono Servlet e pagine JSP riservate al solo amministratore, poi vi sono Servlet e pagine JSP alle quali vi può accedere l'utente ma anche l'amministratore (generalmente l'amministratore a accesso alle stesse funzionalità dell'utente, più altre a lui dedicate)
3. Le pagine JSP riservate all'amministratore vanno nella cartella "admin", mentre per le Servlet, l'URL pattern avrà la seguente forma "/admin/myServlet"
4. Le pagine JSP a cui può accedere sia l'utente che l'amministratore vanno nella cartella "common", mentre per le Servlet, l'URL pattern avrà la seguente forma "/common/myServlet"

Restrizione degli accessi: linee guida

- Fare attenzione agli URL relativi
  - Se la mia JSP si trova nella cartella “admin”, gli URL relativi nella pagina fanno riferimento a questa cartella
  - Discorso simile per le Servlet
- Esempio:
  - Se in “admin/protected.jsp” ho la chiamata **response.sendRedirect("login.jsp"),** l’URL risolto è “myWebApp/admin/login.jsp” e non “myWebApp/login.jsp”
  - Posso risolvere nel seguente modo: **response.sendRedirect(request.getContextPath() + "/login.jsp");**
  - Oppure nel seguente modo: **response.sendRedirect("../login.jsp");**

Implementazione delle restrizione degli accessi in maniera programmatica

- **Idea: token nella sessione**
  - I nomi utente e le password sono conservati in una tabella "utenti" della base di dati
  - L'applicazione Web presenta una pagina di login attraverso cui l'utente fornisce il proprio nome utente e password
  - Se il nome utente e la password fornita trovano riscontro nella base di dati, allora salvo nella sessione il "token"
  - In ogni Servlet o pagina JSP protetta, devo assicurarmi che l'utente sia autenticato (ovvero il "token" è nella sessione) e che abbia il diritto di accedete a quella Servlet o pagina JSP
    - Se non ne ha il diritto, lo rimandiamo alla pagina di login (o ad una pagina di errore)
  - Quando eseguo il logout invalido la session con **session.invalidate()**

Esempio: Token nella Session (login.jsp in Protection.zip)

```
...
<body>
<%
List<String> errors = (List<String>) request.getAttribute("errors");
if (errors != null){
    for (String error: errors){ %>
        <%=error %> <br>
    <%
    }
}
%>
<form action="Login" method="post">
<fieldset>
     <legend>Login Custom</legend>
     <label for="username">Username</label>
     <input id="username" type="text" name="username" placeholder="enter username">
     <br>
     <label for="password">Password</label>
     <input id="password" type="password" name="password" placeholder="enter password">
     <br>
     <input type="submit" value="Login"/>
     <input type="reset" value="Reset"/>
</fieldset>
</form>
</body>
</html>
```

Esempio: Token nella Session (Login.java)

```java
@WebServlet("/Login")
public class Login extends HttpServlet {

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

            String username = request.getParameter("username");
            String password = request.getParameter("password");
            List<String> errors = new ArrayList<>();
            RequestDispatcher dispatcherToLoginPage = request.getRequestDispatcher("login.jsp");

            if(username == null || username.trim().isEmpty()) {
                errors.add("Il campo username non può essere vuoto!");
            }
            if(password == null || password.trim().isEmpty()) {
                errors.add("Il campo password non può essere vuoto!");
            }
            if (!errors.isEmpty()) {
                request.setAttribute("errors", errors);
                dispatcherToLoginPage.forward(request, response);
                return; // note the return statement here!!!
            }

            username = username.trim();
            password = password.trim();

            if(username.equals("admin") && password.equals("mypass")){ //admin
                request.getSession().setAttribute("isAdmin", Boolean.TRUE); //inserisco il token nella sessione
                response.sendRedirect("admin/protected.jsp");
            } else if (username.equals("user") && password.equals("mypass")){ //user
                request.getSession().setAttribute("isAdmin", Boolean.FALSE); //inserisco il token nella sessione
                response.sendRedirect("common/protected.jsp");
            } else {
                errors.add("Username o password non validi!");
                request.setAttribute("errors", errors);
                dispatcherToLoginPage.forward(request, response);
            }
    }
```

Qui in realtà si dovrebbero caricare le credenziali dal database

Esempio: Token nella Session (admin/protected.jsp)

```
...
</head>
<%
// Check user credentials
Boolean isAdmin = (Boolean) session.getAttribute("isAdmin");
if ((isAdmin == null) || (!isAdmin)){
    response.sendRedirect(request.getContextPath() + "/login.jsp");
    return;
}
%>
<body>
<h1>Benvenuto nella Pagina Riservata all'Amministratore</h1>

<p>
Congratulazioni! <br> Questa pagina è accessibile all'amministratore soltanto (non all'utente).
</p>

<p>
<a href="<%=request.getContextPath()%>/common/Logout">Logout</a>
</p>

</body>
</html>
```

Esempio: Token nella Session (common/protected.jsp)

```
...
</head>
<%
// Check user credentials
Boolean isAdmin = (Boolean) session.getAttribute("isAdmin");
if (isAdmin == null){
    response.sendRedirect(request.getContextPath() + "/login.jsp");
    return;
}
%>
<body>
<h1>Benvenuto nella Pagina Riservata</h1>

<p>
Congratulazioni! <br> Questa pagina è accesibile sia all'utente che all'amministratore.
</p>

<p>
<a href="Logout">Logout</a>
</p>

</body>
</html>
```

Esempio: Token nella Session (Logout.java)

```java
@WebServlet("/common/Logout")
public class Logout extends HttpServlet {

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

        Boolean isAdmin = (Boolean) request.getSession().getAttribute("isAdmin");
        if (isAdmin == null){
            response.sendRedirect(request.getContextPath() + "/login.jsp");
            return;
        }

        request.getSession().invalidate();
        response.sendRedirect(request.getContextPath() + "/login.jsp");
    }
```

Token nella session: verso una soluzione migliore

- Nell soluzione vista sinora, in ogni Servlet o pagina JSP protetta, devo assicurarmi che l'utente sia autenticato (ovvero il "token" è nella sessione) e che abbia il diritto di accedete a quella Servlet o pagina JSP
  - Svantaggio: stessa porzione di codice da ripetere in più Servlet o pagine JSP
  - Soluzione: uso di un filtro

Esempio (AccessControlFilter.java in ProtectionFilters.zip):

```
@WebFilter(filterName = "/AccessControlFilter", urlPatterns = "/*")
public class AccessControlFilter extends HttpFilter implements Filter {

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpServletRequest = (HttpServletRequest) request;
        HttpServletResponse httpServletResponse = (HttpServletResponse) response;

        Boolean isAdmin = (Boolean) httpServletRequest.getSession().getAttribute("isAdmin");
        String path = httpServletRequest.getServletPath();
        System.out.println(path);
        if (path.contains("/common/") && isAdmin==null) {
            httpServletResponse.sendRedirect(httpServletRequest.getContextPath() + "/login.jsp");
            return;
        } else if (path.contains("/admin/") && (isAdmin==null || !isAdmin)) {
            httpServletResponse.sendRedirect(httpServletRequest.getContextPath() + "/login.jsp");
            return;
        }

        chain.doFilter(request, response);
    }
```

Esempio (common/protected.jsp in ProtectionFilters.zip):

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Pagina Riservata</title>
</head>

<body>
<h1>Benvenuto nella Pagina Riservata</h1>

<p>
Congratulazioni! <br> Questa pagina è accesibile sia all'utente che all'amministratore.
</p>

<p>
<a href="Logout">Logout</a>
</p>

</body>
</html>
```

SSL/TLS (HTTPS) in Tomcat

Nota che, nella pratica, il certificato SSL/TLS lo rilascia una Certification Authority

1. Crea il certificate SSL/TLS usando il Keytool di Java
   a. Da prompt dei comandi:

      keytool -genkey -alias tomcat -keypass **myPassword** -keystore keystore.jks -storepass **myPassword** -keyalg RSA -validity 360 -keysize 2048

   b. Segui le istruzioni che compaiono nel prompt dei comandi
   c. Al termine verrà creato il file "keystore.jks" nella cartella corrente
2. Sposta il file "keystore.jks":
   - nelle directory principale di Tomcat, in caso di deployment nell'ambiente di produzione,
   - o in <workspace>/.metadata/.plugins/org.eclipse.wst.server.core/tmp0, in caso di deployment nell'ambiente di sviluppo

SSL/TLS (HTTPS) in Tomcat

3. Configura il connettore nel file di configurazione server.xml (clicca sul progetto "Servers" in Eclipse se stai facendo il deployment nell'ambiente di sviluppo, altrimenti vai nella cartella "conf" nella directory principale di Tomcat) come segue:

```
<Connector port="443" protocol="org.apache.coyote.http11.Http11NioProtocol"
maxThreads="150" SSLEnabled="true" maxParameterCount="1000">
    <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
    <SSLHostConfig>
        <Certificate certificateKeystoreFile="keystore.jks"
                     certificateKeystorePassword="myPassword" type="RSA"/>
    </SSLHostConfig>
</Connector>
```

3. Modifica la redirectPort della seguente linea come segue (da 8443 a 443):

```
<Connector connectionTimeout="20000" port="80" protocol="HTTP/1.1"
redirectPort="443"/>
```

4. Avvia Tomcat

5. Nel browser: **https://localhost/yourWebApp/...**

Come impedire agli attaccanti di rubare dati di rete mentre questi sono in transito?

- Maniera programmatica:
  - Per salvaguardare i dati di rete, ogni Servlet o pagina JSP deve verificare il protocollo di rete utilizzato per accedervi (**request.isSecure()**). Se gli utenti tentano di utilizzare una normale connessione HTTP per accedere a uno di questi URL, la Servlet o la pagina JSP deve reindirizzarli manualmente all'equivalente HTTPS (SSL/TLS)
    - Si possono usare i filtri!!!
- Maniera dichiarativa
  - Occorre aggiungere dei parametri di configurazione a web.xml, segue un esempio (nel quale ogni risorsa della mia Web app è accessibile solo tramite HTTPS)

Esempio (Maniera Dichiarativa)

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

Sia nel caso di maniera dichiarativa che programmatica, devi aver configurato Tomcat opportunamente (vedi slide "SSL/TLS (HTTPS) in Tomcat")!

Vedi web.xml in ProtectionFilters.zip

Attacchi Clickjacking

- È una tecnica di attacco informatico che consiste nell'ingannare un utente inducendolo a cliccare su qualcosa di diverso da ciò che percepisce, sfruttando elementi nascosti o trasparenti sovrapposti a contenuti visibili
- Come funziona:
  - Un attaccante crea una pagina web che incorpora (ad esempio tramite un <iframe>) un altro sito legittimo
  - Poi posiziona un elemento invisibile o trasparente sopra il pulsante reale (ad esempio "Mi piace", "Acquista", "Autorizza", ecc.)
  - L'utente crede di cliccare su un normale bottone, ma in realtà sta interagendo con il contenuto nascosto, compiendo azioni a sua insaputa

Difendersi da Clickjacking

Nel file web.xml:

```
<filter>
  <filter-name>httpHeaderSecurity</filter-name>
  <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
   </init-param>
  <init-param><param-name>antiClickJackingEnabled</param-name><param-value>true</param-value>
  </init-param>
   <init-param><param-name>antiClickJackingOption</param-name><param-value>DENY</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>httpHeaderSecurity</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Vedi web.xml in ProtectionFilters.zip

Memorizzazione delle password

- Le password nel database non devono essere memorizzate in chiaro
- Anziché memorizzare la password nel database, va memorizzato l'hash della password
  - Hash: funzione non invertibile che mappa una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita
- Quando si fa il login, occorre calcolare l'hash della password fornita dall'utente e poi confrontarla con l'hash della password memorizzato nel database
- Nota che esistono tecniche più sofisticate per la memorizzazione delle password!!!

Esempio (Login.java in ProtectionPasswordHashing.zip)

```
private String toHash(String password) {
    String hashString = null;
    try {
        java.security.MessageDigest digest = java.security.MessageDigest.getInstance("SHA-512");
        byte[] hash = digest.digest(password.getBytes(StandardCharsets.UTF_8));
        hashString = "";
        for (int i = 0; i < hash.length; i++) {
            hashString += Integer.toHexString(
                                (hash[i] & 0xFF) | 0x100
                            ).substring(1,3);
        }
    } catch (java.security.NoSuchAlgorithmException e) {
        System.out.println(e);
    }
    return hashString;
}
```

L'algoritmo di hashing usato è lo SHA-512, più sicuro di SHA-1 e SHA-256

Esempio: l'hash della password "mypass" è:

"1c573dfeb388b562b55948af954a7b344dde1cc2099e978a992790429e7c01a4205506a93d9aef3bab32d6f06d75b7777a7ad8859e672fedb6a096ae369254d2"

Esempio

```java
@WebServlet("/Login")
public class Login extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        List<String> errors = new ArrayList<>();
        RequestDispatcher dispatcherToLoginPage = request.getRequestDispatcher("login.jsp");
        if (username == null || username.trim().isEmpty()) {
            errors.add("Il campo username non può essere vuoto!");
        }
        if (password == null || password.trim().isEmpty()) {
            errors.add("Il campo password non può essere vuoto!");
        }
        if (!errors.isEmpty()) {
            request.setAttribute("errors", errors);
            dispatcherToLoginPage.forward(request, response);
            return; // note the return statement here!!!
        }
        username = username.trim();
        password = password.trim();
        String hashPassword = toHash(password);
        // Hash of "mypass":
        String hashPasswordToBeMatch =
                "1c573dfeb388b562b55948af954a7b344dde1cc2099e978a992790429e7c01a4205506a93d9aef3bab32d6f06d75b7777a7ad8859e672fedb6a096ae369254d2";
        if (username.equals("admin") && hashPassword.equals(hashPasswordToBeMatch)) { // admin
            request.getSession().setAttribute("isAdmin", Boolean.TRUE); // inserisco il token nella sessione
            response.sendRedirect("admin/protected.jsp");
        } else if (username.equals("user") && hashPassword.equals(hashPasswordToBeMatch)) { // user
            request.getSession().setAttribute("isAdmin", Boolean.FALSE); // inserisco il token nella sessione
            response.sendRedirect("common/protected.jsp");
        } else {
            errors.add("Username o password non validi!");
            request.setAttribute("errors", errors);
            dispatcherToLoginPage.forward(request, response);
        }
    }
```

Qui in realtà si dovrebbero caricare le credenziali dal database (ovvero nome utente e hash della password)