Tecnologie di Sviluppo per il Web

Applicazioni Web J2EE: Tecniche di Programmazione

versione 3.0

G. Mecca – mecca@unibas.it – Università della Basilicata

Sommario

- Autenticazione e Autorizzazione
  - ⇨SQL Injection
- Convalida dei Dati
- Riscrittura degli URI
- Campi Nascosti

Autorizzazione e Autenticazione

- Problema ricorrente
  - ⇨accesso ad una (porzione di) applicazione che richiede autenticazione attraverso nome utente e password
- Varie possibili soluzioni
  - ⇨gestione “dichiarativa” utilizzando le funzionalità offerte dal server e HTTP
  - ⇨gestione “procedurale” utilizzando una base di dati di nomi utente e password

Autorizzazione e Autenticazione

- Vantaggi della I soluzione
  - ⇨il meccanismo è supportato dai server e non richiede programmazione
- Svantaggi della I soluzione
  - ⇨richiede che l’amministratore configuri gli utenti registrandoli sul server
  - ⇨non consente la “registrazione” interattiva degli utenti

Autorizzazione e Autenticazione

- Di conseguenza
  - ⇨utilizzeremo la seconda soluzione che è più flessibile
- L'approccio
  - ⇨è esattamente lo stesso già visto per le applicazioni desktop
  - ⇨basato sulla base di dati di utenti e password

Autorizzazione e Autenticazione

- Base di dati di utenti e password
  - ⇨i nomi utente e le password (crittografate) sono conservati in una tabella utenti della base di dati
  - ⇨l'applicazione presenta una pagina iniziale per il login attraverso cui l'utente fornisce il nome utente e la password
  - ⇨il nome utente fornito viene cercato per l'autenticazione attraverso un metodo di ricerca nella base di dati (metodo di un DAO)

Autorizzazione e Autenticazione

- Base di dati di utenti e password (cont.)
  - ⇨il metodo di ricerca restituisce il riferimento ad un bean utente che contiene le informazioni relative all'utente
  - ⇨il bean è capace di autenticare l'utente confrontando la password fornita al login con quella nella base di dati
  - ⇨se l'utente non è autenticato, viene visualizzato un messaggio di errore e si torna al login

Autorizzazione e Autenticazione

- Base di dati di utenti e password (cont.)
  - ⇨se l'utente è autenticato, il riferimento al bean viene salvato nella sessione
  - ⇨le pagine successive che richiedono autorizzazioni verificano la presenza del bean nella sessione
  - ⇨se non trovano il bean rimandano alla pagina del login

aciM1 >>

Autorizzazione e Autenticazione

- Una prima annotazione
  - ⇨attenzione alla pagina login.jsp di aciM1
  - ⇨non genera nessuno schermo, ma si limita a re-indirizzare la richiesta ad altri schermi a seconda della natura della richiesta
- Un altro esempio
  - ⇨schermoEsitoRicercaProprietarioPerCF.jsp

Autorizzazione e Autenticazione

- Si tratta di componenti di “puro controllo”
  - ⇨componenti dell’applicazione che non generano schermi
  - ⇨avrebbero potuto essere sviluppati attraverso servlet, per rendere più evidente la loro funzione
  - ⇨ma averlo scritto sotto forma di pagina .jsp semplifica il ciclo di sviluppo, compilazione e caricamento dell’applicazione

Autorizzazione e Autenticazione

- Una seconda annotazione
  - ⇨la soluzione presenta problemi legati alla sicurezza
- Infatti
  - ⇨la pagina di autenticazione è tipicamente una pagina pubblica
  - ⇨accessibile a qualsiasi utente, anche a potenziali attaccanti maleintenzionati

Autorizzazione e Autenticazione

- Principali problemi di sicurezza
  - ⇨problema n.1: i nomi utente e le password viaggiano in chiaro; sarebbe necessario utilizzare SSL e HTTPS
  - ⇨problema n.2: è necessario proteggersi – molto più che nelle applicazioni desktop – da eventuali problemi di “SQL Injection”

SQL Injection

- Che cos'è la SQL Injection ?
  - ⇨una tecnica per eseguire codice SQL non autorizzato sulla base di dati di un'applicazione client server
- In sintesi
  - ⇨un attaccante può alterare l'esecuzione delle query fornendo valori artefatti attraverso le form che vengono usati come input per costruire query SQL sulla base di dati

SQL Injection

- Per capire meglio il problema
  - ⇨vediamo un esempio tipico
  - ⇨facciamo una escursione nella tecnologia ASP (3.0)
  - ⇨vediamo un frammento di una tipica pagina ASP (3.0) per l'autenticazione degli utenti
  - ⇨si tratta di codice tipico perchè molto utilizzato dagli sviluppatori ASP

```
<%
   dim nomeUtente, password, query, conn, recordSet

   nomeUtente = Request.Form("nomeUtente")
   password = Request.Form("password")

   set conn = server.createObject("ADODB.Connection")
   set recordSet = server.createObject("ADODB.Recordset")
   query = "select count(*) from utenti"
           & "where nomeutente = '" & nomeUtente & "',"
           & " and password = '" & password & "'"

  conn.Open "Provider=SQLOLEDB; DataSource=(local); User Id=sa; Password="
  recordSet.activeConnection = conn
  recordSet.open query

  if not recordSet.eof ther
     Response.write "Utente autenticato"
  else
     Response.write "Errore nel nome utente o nella password"
  end if
%>
```


SQL Injection

- In sintesi
  - ⇨lo script VBScript preleva nome utente e password dalla query string
  - ⇨effettua una selezione sulla tabella utente con i valori di nome utente e password
  - ⇨considera autenticato l'utente se il record set risultante non è vuoto (not recordSet.eof), ovvero se contiene l'ennupla corrispondente all'utente in questione

SQL Injection

- Ma...
  - ⇨supponiamo che un attaccante fornisca i seguenti dati attraverso la form
  - ⇨nomeUtente: xyz, password: ‘ or 1=1 --
- In questo caso la query SQL diventa

```
select count(*) from utenti
where nomeutente = ‘xyz’ and
password = ‘ ’ or 1=1 -- ’
```

NOTA: -- rappresenta un commento per SQLServer

SQL Injection

- Di conseguenza
  - ⇨la condizione è sempre verificata
  - ⇨count(*) restituisce il numero di ennuple contenute nella tabella utenti
  - ⇨l’utente viene sempre considerato autenticato
  - ⇨il meccanismo di autenticazione viene aggirato facilmente

SQL Injection

- Ritorniamo alle nostre applicazioni
  - ⇨in effetti, nell'architettura utilizzata nei nostri esempi questo trucco non funzionerebbe
  - ⇨il DAO tipicamente effettua la selezione solo sulla base del nomeutente
  - ⇨String query = "select * from utenti where nomeutente = '" + nomeUtente + "'";
  - ⇨successivamente il bean confronta le due password

SQL Injection

- Di conseguenza
  - ⇨è estremamente difficile aggirare il meccanismo di autorizzazione
- Ma supponiamo che l'attaccante fornisca
  - ⇨nomeUtente: '; delete * from utenti; '
  - ⇨la query diventa:
  - ⇨select * from utenti where nomeutente = ''; delete * from utenti; ''

SQL Injection

- In questo caso
  - ⇨il comportamento dell'applicazione dipende dal driver e dal DBMS
  - ⇨in molti casi viene semplicemente sollevata una SQLException
  - ⇨in altri casi, però, le due istruzioni SQL vengono eseguite l'una dopo l'altra, e questo porta alla perdita totale dei dati della tabella utenti

SQL Injection

- In sintesi
  - ⇨la logica di questi attacchi è introdurre caratteri particolari (come apici e punto e virgola) per alterare la sintassi della query eseguita
- Come difendersi da questi attacchi ?
  - ⇨in linea di principio una strategia di difesa potrebbe essere quella di analizzare i dati forniti dall'utente per eliminare tutti i caratteri potenzialmente pericolosi

SQL Injection

- Ma questo processo è molto intricato
  - ⇨non è detto che eliminando i caratteri tipici non vengano trovati altri caratteri pericolosi
  - ⇨la filosofia corretta sarebbe piuttosto quella di consentire solo le cose sicure, piuttosto che quella di eliminare quelle insicure
  - ⇨inoltre bisogna tenere in considerazione tutte le possibili codifiche (es: \', URI encoding ecc.)

SQL Injection

- Una soluzione molto migliore
  - ⇨utilizzare PreparedStatement
- Infatti
  - ⇨il codice SQL di uno statement preparato viene precompilato
  - ⇨non è possibile alterarlo attraverso le costanti fornite dall'utente
  - ⇨in effetti gli statement preparati sono immuni al 99% dei problemi di SQL injection

SQL Injection

- Di conseguenza
  - ⇨nel DAOUtente di aciM1, viene utilizzato uno statement preparato per l'autenticazione
  - ⇨lo statement viene preparato, eseguito e poi chiuso, e quindi evidentemente i benefici non sono legati alla precompilazione
  - ⇨sono esclusivamente dovuti alla gestione dei problemi di SQL injection

aciM1 >>

Convalida dei Dati Forniti dall'Utente

- Problema ricorrente
  - ⇨ogni volta che l'utente fornisce dati attraverso una form, è necessario verificarne la correttezza
- Ma
  - ⇨rispetto al processo tipico di convalida, ci sono delle sottigliezze collegate al fatto che i dati vengono forniti al client e devono poi essere gestiti dalla logica applicativa sul server

Convalida dei Dati Forniti dall'Utente

- Il supporto fornito dal contenitore
  - ⇨limitata: tutti i dati acquisiti dalla query string sono considerati valori di tipo stringa
- Soluzione tipica
  - ⇨convalida sul client utilizzando JavaScript
- Ma
  - ⇨molti browser disabilitano JavaScript
  - ⇨alcuni vincoli (es: password) devono essere necessariamente verificati sul server

Convalida dei Dati Forniti dall'Utente

- Soluzione proposta
  - ⇨convalida sul server e utilizzo di "formBean"
- FormBean
  - ⇨un formbean è un componente in cui vengono raccolti i parametri di una form
  - ⇨uno per ogni form che richiede convalida
  - ⇨una proprietà di tipo stringa per ciascun controllo della form
  - ⇨un metodo convalida() per la verifica dei dati

Convalida dei Dati Forniti dall'Utente

- Attenzione a non fare confusione
  - ⇨è frequente il caso in cui c'è corrispondenza tra controlli di una form (es: form per l'inserimento di un proprietario)
  - ⇨e attributi di un bean (es: bean Proprietario)
  - ⇨questo produce l'effetto per cui nell'applicazione ho due componenti molto simili: bean e formBean

Convalida dei Dati Forniti dall'Utente

- Differenze tra un bean e un formBean
  - ⇨un formBean è un componente del controllo e non del modello
  - ⇨rappresenta non un concetto del modello ma i dati di una query string
  - ⇨tutte le proprietà sono di tipo string
  - ⇨non contiene metodi di logica applicativa, tranne il metodo public List convalida() che restituisce l'eventuale lista di errori

Convalida dei Dati Forniti dall'Utente

- Le due funzioni del formBean
  - ⇨I funzione: serve a verificare i dati forniti dall'utente PRIMA di modificare il bean
  - ⇨II funzione: serve a ricordare i dati forniti dall'utente quanto lo schermo viene rigenerato in caso di errori di convalida (in modo da poter visualizzare all'utente i dati precedentemente forniti e chiedergli di correggerli: importante per l'usabilità)

Convalida dei Dati Forniti dall'Utente

- Un esempio: esempioConvalida
  - ⇨lo schermo per l'inserimento del proprietario
  - ⇨la form contiene i controlli "nome", "codiceFiscale", "cittaDiResidenza", "annoPatente"
  - ⇨il bean relativo è Proprietario.java
  - ⇨il formBean relativo è FormProprietario.java

esempioConvalida >>

Convalida dei Dati Forniti dall'Utente

- Strategia per la convalida
  - ⇨per ogni attività di acquisizione di dati da form che richiede convalida, il programmatore definisce due pagine
  - ⇨una pagina server di acquisizione (genera la form per il client)
  - ⇨una pagina server di conferma dell'acquisizione (verifica i dati e genera un messaggio di conferma)

Convalida dei Dati Forniti dall'Utente

- Strategia (cont.)
  - ⇨la form invoca la pagina di conferma
  - ⇨la pagina di conferma istanzia il formBean e chiama il metodo di convalida
  - ⇨se non ci sono errori, produce il messaggio di conferma e procede nelle operazioni
  - ⇨altrimenti, inoltra (restituisce) la richiesta alla pagina di acquisizione, salvando nella richiesta il formBean e gli errori

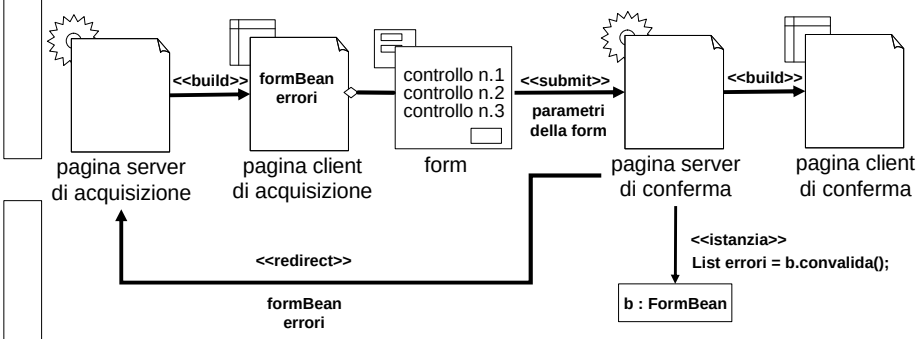
Convalida dei Dati Forniti dall'Utente

- Strategia (cont.)
  - ⇨la pagina di acquizione visualizza gli errori e riproduce la form mostrando all'utente i valori precedentemente inseriti
  - ⇨l'utente corregge i dati e risottomette la form
  - ⇨il procedimento si ripete finché i dati non sono corretti

Convalida dei Dati Forniti dall'Utente

esempioConvalida >> aciM1 >>

- Descrizione della strategia



Convalida dei Dati Forniti dall'Utente

- Riassumendo il processo
  - ⇨viene visualizzata la pagina di immissione; il formBean inizialmente è vuoto e i controlli della form non hanno valori
  - ⇨l'utente sottomette la form; viene generata la richiesta HTTP
  - ⇨l'applicazione avvia il processo di convalida: viene istanziato il formBean e chiamato il metodo di convalida

Convalida dei Dati Forniti dall'Utente

- Riassumendo il processo (cont.)
  - ⇨se ci sono errori, si torna allo schermo di immissione, e il formBean viene utilizzato per ripristinare nella form i valori forniti dall'utente
  - ⇨l'utente corregge i valori e risottomette
  - ⇨se non ci sono ulteriori errori di convalida, vengono eseguite le azioni previste sul modello (es: creazione o aggiornam. bean)
  - ⇨viene visualizzato lo schermo di conferma

Convalida dei Dati Forniti dall'Utente

>> esempioConvalida

- Un altro esempio
  - ⇨una convalida avanzata
  - ⇨mostra come convalidare controlli delle form che non sono di tipo testo
  - ⇨es: radio, checkbox, select

Riscrittura degli URI

- Tecnica base per istaurare una sessione
  - ⇨a ciascuna sessione è associato un id unico
  - ⇨il server invia al browser un cookie con l'id della sessione
  - ⇨il browser lo accetta e successivamente lo restituisce
  - ⇨la gestione dei cookie è del tutto trasparente per l'utente

Riscrittura degli URI

- Non sempre
  - ⇨l'implementazione delle sessioni basata su cookie funziona
- Molti browser rifiutano i cookie
  - ⇨impedendo di ricondurre richieste successive alla stessa sessione
  - ⇨i contenitori J2EE dispongono di un'alternativa: riscrittura degli URI ("URL rewriting")

Riscrittura degli URI

- Riscrittura degli URI
  - ⇨tecnica basata sull'idea di aggiungere l'id della sessione agli URI delle richieste della sessione stessa
  - ⇨in modo che, anche in assenza di cookie, sia possibile ricondurre ciascuna richiesta alla sessione corrispondente

Riscrittura degli URI

- URI riscritti
  - agli URI inseriti nel codice HTML delle risposte inviate al client il contenitore aggiunge l'identificatore della sessione
  - esempio: http://localhost:8080/indovinam1/tentativo.jsp;jsessionid=E4D371A0710
  - pagine che appartengono a sessioni diverse conterranno URI diversi
  - alla richiesta successiva, l'id della sessione verrà estratto dall'URI e ricondotto alla sess.

Riscrittura degli URI

- Attenzione
  - ancora più dei cookie, la tecnica di URL rewriting si presta a violazioni della privacy
  - infatti gli URI sono visibili (es: nella cronologia, nelle cache o nei bookmark)
  - possono essere riprodotti artificialmente negli URI richiesti per interferire in una sessione

Riscrittura degli URI

- Di conseguenza
  - ⇨il contenitore riscrive gli URI solo se lo sviluppatore lo richiede esplicitamente
  - ⇨eseguendo il metodo String encodeURL(String uri) di HttpServletResponse
  - ⇨il parametro rappresenta l'uri non riscritto
  - ⇨il risultato del metodo è l'uri riscritto dal contenitore

Riscrittura degli URI

>> aciM1

- Inoltre
  - ⇨per evitare problemi di sicurezza, il contenitore tende a preferire i cookie alla riscrittura degli URI
  - ⇨ovvero, riscrive effettivamente gli URI solo nel caso in cui si accorge che il client rifiuta i cookie
  - ⇨se un client accetta i cookie, il metodo encodeURL() lascia l'uri immutato

Riscrittura degli URI

- Suggerimento
  - ⇨utilizzare sempre encodeURL() per rendere più robusta la gestione delle sessioni
  - ⇨eventualmente proteggendo il traffico da intrusioni utilizzando HTTPS (in modo che non sia possibile “sniffare” dall’esterno gli ID delle sessioni intercettando le richieste)

Campi Nascosti

- La riscrittura degli URI
  - ⇨è in effetti un meccanismo per fare in modo che il client invii informazioni aggiuntive al server
  - ⇨in modo che queste informazioni aiutino il server nella gestione delle sessioni di lavoro
- Una tecnica simile
  - ⇨è basata sull’utilizzo di campi nascosti (“hidden”) all’interno delle form HTML

Campi Nascosti

- Campi “hidden”
  - aggiungono alla query string prodotta sottomettendo la form dei valori aggiuntivi rispetto a quelli forniti dall’utente
- Esempio
  - nell’applicazione IndovinaIlNumero, c’è il problema di cominciare una nuova partita dopo avere invalidato la sessione
  - è necessario conservare il nome del giocatore

Campi Nascosti

>> indovinato.jsp

- Soluzione
  - un campo nascosti della form HTML
  - <input type=“hidden” value=“<valore>” />
  - la form che consente di cominciare la nuova partita ha un campo hidden che contiene il nome del giocatore
  - in questo modo la pagina tentativo.jsp riceve una richiesta in cui c’è il parametro nome

Campi Nascosti

- In effetti
  - ⇨l'utilizzo dei campi "hidden" è una tecnica potenzialmente alternativa ai cookie per mantenere lo stato tra richieste diverse
  - ⇨in entrambi i casi vengono inviate al client informazioni sullo stato che gli si chiede di rispedire con le richieste successive
  - ⇨i campi hidden garantiscono meno privatezza dei cookie (il codice HTML è più visibile)

Campi Nascosti

- Una soluzione ancora diversa
  - ⇨utilizzare richieste get
- Negli esempi
  - ⇨la stragrande maggioranza delle richieste sono di tipo post e vengono scatenate da bottoni
  - ⇨ma è possibile anche costruire nel codice JSP richieste di tipo get, specificando manulamente i parametri della query string

Campi Nascosti

- Esempio: indovinato.jsp

```
<a href="tentativo.jsp?nome=<%=partita.getNome()%>">
     Comincia un'altra partita
</a>
```

- In questo modo
  - ⇨il bottone viene sostituito da un link
  - ⇨il link scatena una richiesta di tipo get
  - ⇨la richiesta get ha un parametro nome nella query string contenente il nome del giocatore

Riassumendo

- Autenticazione e Autorizzazione
  - ⇨SQL Injection
- Convalida dei Dati
- Riscrittura degli URI
- Campi Nascosti

Termini della Licenza