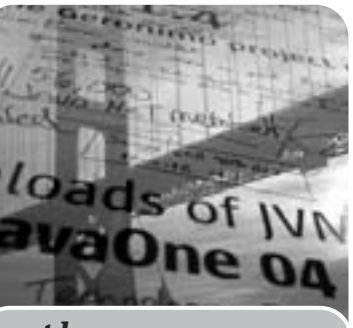


thema

Het internet: de wereld is je marktplaats en je consument. Naast de ongekende mogelijkheden om producten en diensten aan te bieden, open je ook je bedrijf voor de wereld aan boosdoeners. Iedereen kan van je internetapplicatie gebruikmaken, en er dus ook misbruik van maken. Dit betekent dat internetapplicaties meer dan ooit tevoren bestand moeten zijn tegen vijandigheden, boosdoeners, boefjes en wat voor gespuis nog meer.

Het web robuust maken

SQL Injection voor internet-applicaties

Ook vanwege de browsertechniek moeten er aan internetapplicaties andere eisen worden gesteld dan aan een specifieke (fat of thin) client in een meer conventioneel client/server systeem. Dat geldt nog sterker ten opzichte van een desktop-applicatie voor een enkele gebruiker. De aanleiding voor dit artikel ligt in een techniek die ik onder ogen kreeg: SQL Injection. SQL Injection is de mogelijkheid om SQL query's of SQL statements in een webpagina in te vullen en deze pagina vervolgens op te sturen naar de server. Daar worden deze query's uitgevoerd door de achterliggende database. Dit gebeurt op een zodanige manier dat de SQL query's als afzonderlijke query's uitgevoerd worden. SQL Injection kan ingezet worden om de volgende problemen op te lossen:

- Foutmeldingen vanuit de database worden ongewijzigd gemeld aan de gebruiker
- Ingevoerde gegevens worden ongewijzigd overgenomen in SQL statements of query's

OPLOSSING Het eerste probleem betreft de database foutmeldingen. Voor ontwikkelaars is het handig om - tijdens het testen - de opgetreden fout in je browser te tonen. Deze meldingen worden echter nooit meer uit het systeem verwijderd. In de gebruikte architectuur moet dus al rekening gehouden worden met het afhandelen van fouten. Het logging mechanisme en de monitoring van deze log files moeten al aanwezig zijn. In een browser mogen nooit specifieke foutmeldingen komen. In de design-fase van de applicatie moet dus ook al aandacht besteed worden aan: de afhandeling van fouten, en de manier waarop fouten worden gemeld aan de gebruiker.

Het tweede probleem betreft ingevoerde gegevens. Het is onmogelijk om volledig sluitend door de browser te laten bepalen wat er naar de server gestuurd wordt. Gebruikers kunnen eventueel met behulp van javascript geholpen worden bij het invullen van juiste waarden. Juist de boefjes zul je hiermee niet tegen houden. Door de pagina gewoon als tekstbestand op te slaan en deze te wijzigen kan andere informatie naar de server gestuurd worden. Als uitgangspunt moet dan ook worden gekozen voor het uiteindelijk laten valideren van gegevens door de server. Willen we onze applicatie zo robuust mogelijk maken dan moet deze serverside validatie dan ook zo uitgebreid en compleet mogelijk gemaakt worden.

Het is mogelijk om alles te valideren op mogelijke invoerkarakters. Reguliere expressies zijn hier uitermate geschikt voor, met name voor tekstinvoervelden. Om andere manieren van invoer ook op deze manier te checken kan wel eens een beetje overdadig zijn. Verschillende soorten invoerelementen kunnen we met verschillende technieken valideren. Niet alles wat een gebruiker invult is in free-format. Een gebruiker heeft bijvoorbeeld ook de mogelijkheid om keuzes te kunnen maken.

- Userinterface elementen waaraan we kunnen denken zijn:
- Tekstvelden (inclusief password, multi-line en hidden velden)
- Radio buttons
- Check boxen
- Listboxen

TEKSTVELDEN Tekstvelden zijn in grofweg twee verschillende categorieën te verdelen. De eerste categorie

zijn tekstvelden die gebruikt worden om een vast formaat aan invoer te verwerken. Denk hierbij aan bijvoorbeeld een wachtwoord. Voor een wachtwoord kunnen we vaststellen dat het minimaal acht karakters lang moet zijn, minimaal een cijfer moet bevatten, en voor het overige alleen 'printable' karakters. Deze stelling is in een reguliere expressie te vangen, waarmee de invoer gecontroleerd kan worden. Nu is Java een objectgeoriënteerde (OO) taal en biedt het OO model hier weer een mooie oplossing voor: value objects. Fowler definieert value objects als volgt: "*A small simple object, like money or date range, whose equality isn't based on identity.*" Maak voor iedere invoer een aparte class. Dit betekent dat we - in het voorbeeld van een wachtwoord - een Wachtwoord class introduceren. Deze zal dan alle kennis bevatten om te kunnen controleren of een string voldoet aan de eisen van het object. Vervolgens worden alleen objecten van dit type doorgegeven of gerepresenteerd binnen de applicatie. Indien we gebruik maken van een traditionele multi-tier architectuur zullen deze objecten in de interface van iedere laag worden doorgegeven.

De tweede categorie - de vrije invoer (free-format) - is lastiger. Hierbij staat het de gebruiker vrij om willekeurig wat voor een tekst in te voeren. Denk hierbij bijvoorbeeld aan de invoer van een bericht bij het maken van een e-mail. Hier zal een andere aanpak gekozen moeten worden. Ook hier kan onze kwaadwillende gebruiker bijvoorbeeld SQL Injection toepassen. We maken het hem zelfs gemakkelijker door een invoerveld te maken, dat een aantal kolommen breed is en een aantal regels hoog. Ook hier zal dus de invoer moeten worden gecontroleerd. Hiervoor zullen we moeten zoeken naar voorkomens van letterreeksen die we niet willen toestaan.

Ook het zogenaamd *escapen* van letters maakt het mogelijk om kwaadwillende invoer tegen te houden. Escapen betekent dat we sommige karakters vervangen door andere, of laten voorgaan door een andere. Een van de boosdoeners in SQL Injection is het ' (quote) karakter, omdat deze meestal wordt gebruikt in query's om een String aan te duiden.

Quotes

De sql query luidt:

```
Select user_id from users where username =
'john' and passwd = 'doe';
```

of

```
Insert into berichten values('Dit is het
onderwerp',
'En dit is de multi line text');
```

Indien deze query's nu dynamisch opgebouwd is met door gebruikers ingevoerde gegevens zal een bericht met de tekst *'); select * from berichten* de volgende query opleveren.

```
  Insert into berichten values('Dit is het
onderwerp',
       ''); select * from berichten');
```

Het aantal quotes klopt niet, maar het levert wel een bijna correcte query op. Een slimme hacker zal hier een aantal pogingen aan wagen en uiteindelijk alle berichten kunnen lezen. Escapen we nu de quotes dan krijgen we het volgende resultaat:

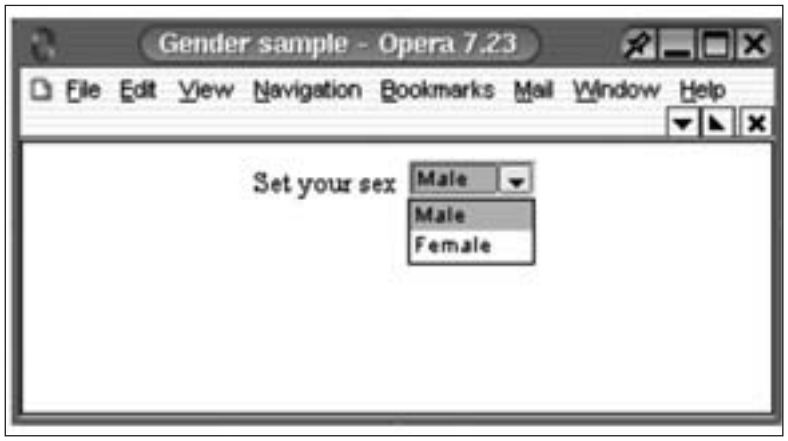
```
  Insert into berichten values('Dit is het
onderwerp',
       ''''); select * from berichten');
```

Behalve quotes zijn er ook andere karakters waar we rekening mee moeten houden. Denk daarbij aan de & en | tekens maar ook < > =.

Checkboxen Radiobuttons, checkboxen en listboxen kunnen teruggebracht worden naar een lijst van vaste waarden waaruit de gebruiker één of meerdere keuzes kan maken, zoals in dit HTML-voorbeeld is te zien:

```
<html>
  <head>
    <title>Gender sample</title>
  </head>
  <body>
    <form action="post" target="/gender">
      <center>Set your sex :
      <select name="gender">
        <option value="m"
selected>Male</option>
        <option value="f">Female</option>
      </select>
      </center>
    </form>
  </body>
</html>
```

In deze pagina moet een keuze voor gender gemaakt worden. Er kan maar uit twee verschillende mogelijkheden gekozen worden. Indien er gebruik gemaakt wordt van Struts kan de bijbehorende Java form class er zo uitzien:



FIGUUR 1. Checkboxen en listboxen kunnen teruggebracht worden naar een lijst van vaste waarden

```
public class GenderForm extends ActionForm {
     String gender = "";

     public String getGender() {
          return gender;
     }
     public void setGender(String gender) {
          this.gender = gender;
     }
}
```

In de bijbehorende Struts action kunnen we de waarde voor *gender* uitlezen en simpel als *String* doorgeven aan een bijbehorende query, met alle gevolgen van dien. Ook hier maken we gebruik van een *Value Object* om ons probleem te kunnen tackelen. Door de opgevoerde waarde voor *gender* niet als *String* door te geven maar als een eigen classe.

```
public class Sex {
  private String code;
  Sex(String code) {
     super();
     this.code = code;
  }

  public String toString() {
     return code;
  }
}
```

Nu hebben we de mogelijkheid om opgevoerde waarde niet meer als *String* door te geven naar de database tier maar als een specifieke classe. Maar dan zijn we er nog niet. Alles wat de gebruiker invoert, wordt nog steeds ongewijzigd doorgegeven. Hiervoor moet de mogelijke waarde die de gebruiker kan invoeren beperkt worden. Om dit te verwezenlijken introduceren we een enumeratie van een mogelijke waarde. En een *valueOf* methode. Deze *valueOf* methode zal dan de waarde die een gebruiker invoert, omzetten naar een corresponderend object.

```
public class Sex {
  private String code;
  Sex(String code) {
     super();
     this.code = code;
  }

  public static final Sex MAN = new Sex("M");
  public static final Sex VROUW = new
  Sex("V");
  public static final Sex UNKNOWN = new
  Sex("U");

  public static final Sex valueOf(String
  code) {
     if (MAN.toString().equals(code)) {
        return MAN;
     }
     if (VROUW.toString().equals(code)) {
        return VROUW;
     }
     return UNKNOWN;
  }

  public String toString() {
     return code;
  }
}
```

Hiermee is het mogelijk om aan de rand van ons systeem,waar we gebruikersinvoer valideren, te controleren of een invoer aan een bepaalde conditie voldoet. De code `Sex sex = Sex.valueOf(getGender());` zal dus te allen tijde een valide object opleveren met een van de drie mogelijke waarden. Hier moeten we dus nog een extra check uitvoeren op de validiteit van het opgeleverde object door een kleine refactoring van de code. Introductie van een *InvalidParameterException* kunnen we de code als volgt aanpassen.

```
public static final Sex valueOf(String code)
throws InvalidParameterException {
  if (MAN.toString().equals(code)) {
     return MAN;
  }
  if (VROUW.toString().equals(code)) {
     return VROUW;
  }
  throw new InvalidParameterException();
}
```

Waarna we met behulp van een *try - catch* block de validatie kunnen uitvoeren.

```
        try {
            Sex sex =
Sex.valueOf(getGender());
                .
  .
  .
  .
        } catch (InvalidParameterException
e) {
             e.printStackTrace();
        }
```

CONCLUSIE Het melden van foutcondities aan gebruikers moet ook tijdens de design fase al besproken worden. In de gebruikte architectuur moet ook plaats zijn voor het melden (loggen) van fouten en voor mogelijkheden om dit te kunnen monitoren. Door een goede sluitende validatie van alle gegevens die onze applicatie ingaan, kunnen we er voor zorgen dat alleen die informatie wordt verwerkt die we willen en daarmee ook kunnen accepteren. In eerste instantie denken we daarbij aan gebruikersinvoer, maar de hier geschetste voorstellen kunnen ook toegepast worden bij het verwerken van gegevens afkomstig uit een ander systeem. Door gebruik te maken van *Value objects*, specifieke objecten voor een bepaald type kunnen we de interne structuren van onze applicatie verstevigen. De functie die een gebruiker moet valideren zal niet meer twee afzonderlijke *Strings* accepteren maar een object *GebruikersNaam* en een object *Wachtwoord*. Hiermee kunnen we dus contracten maken binnen ons systeem. Door gebruik te maken van enumeratie is het mogelijk om gebruik te maken van vaste reeksen. Van de classe Sex kunnen maar twee verschillende objecten bestaan. (MAN en VROUW). Iets anders is niet mogelijk.

Philippe Tjon-A-Hen is ICT Consultant op het terrein van Java en architectuur. Hij is werkzaam bij Ordina (e-mail: philippe.a.hen.tjon@ordina.nl).