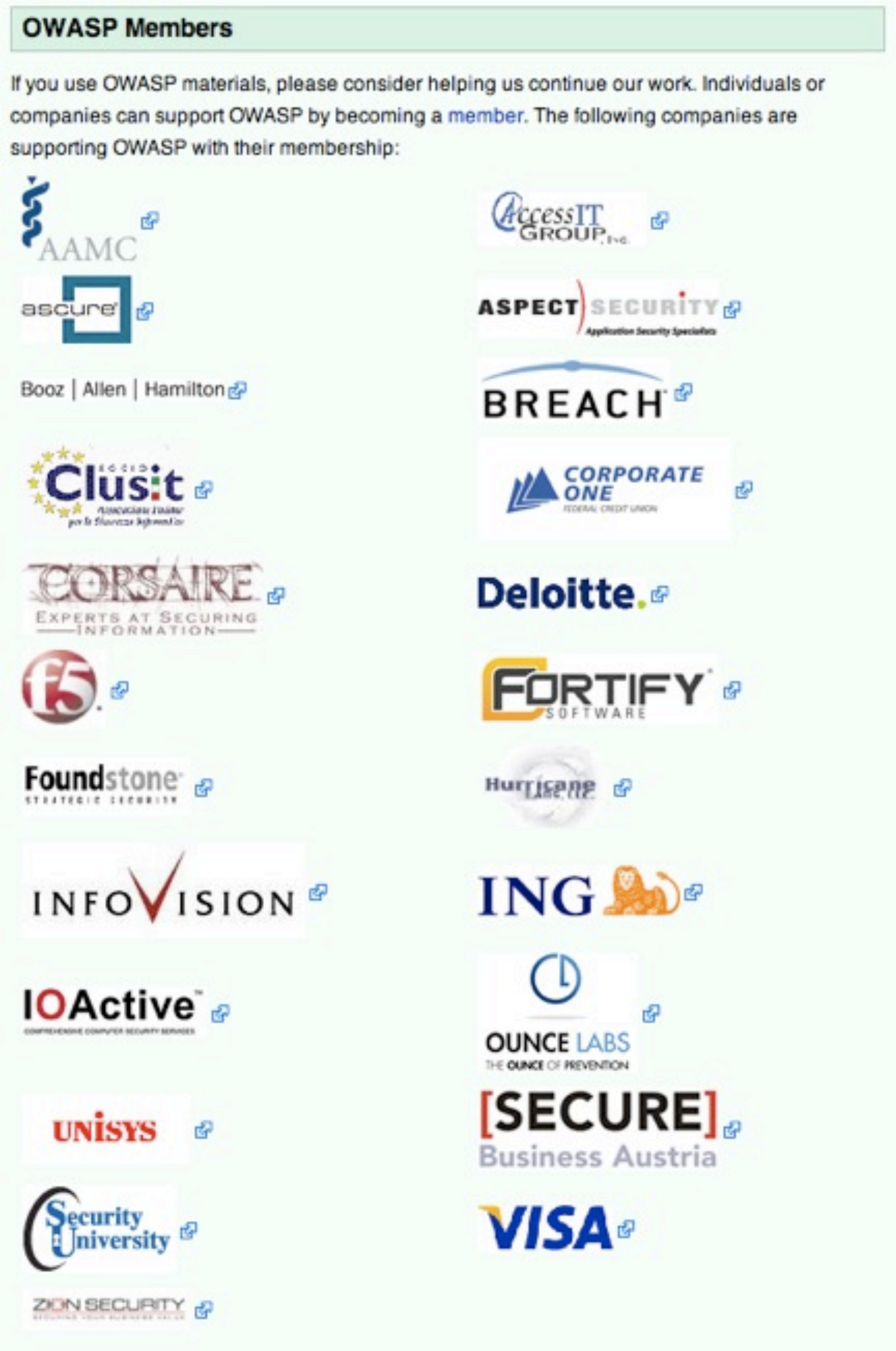
Web Application Security

OWASP

- Open Web Application Security Project
- http://www.owasp.org
- “...develop, purchase and maintain secure applications”



OWASP Members

If you use OWASP materials, please consider helping us continue our work. Individuals or companies can support OWASP by becoming a member. The following companies are supporting OWASP with their membership:

Don’t Try This At Home

Injection Flaws

- SQL Injection
  - input parameter is passed to DB query string
  - “SELECT * FROM user_data WHERE last_name = '” + param + “'”;
- Command Injection
  - input parameter passed to external program, e.g. Sendmail
  - “; rm -r *”

- Never concatenate strings together to form HQL query

```
def search = {
    Book.findAll("from Book as b where b.title='"
       + params.title +"'")
}
```

```
//use query parameters instead:

def search = {
    Book.findAll("from Book as b where b.title=?",
       [params.title])
}
```

- Never execute user input in a Groovy shell

```
def execute = {
    new GroovyShell().evaluate(params.script)
}
```

Cross Site Scripting (XSS)

- Injecting malicious code (JavaScript, VBScript, ActiveX, HTML, or Flash) into a “trusted” website
- Protect by:
  - validating input
  - encoding output

Prevention

- Config.groovy
  - grails.views.default.codec="html"
- GSP
  - <%@ defaultCodec="html" %>
- Individual Case
  - ${book?.title?.encodeAsHTML()}

Broken Authentication and Session Handling

- Password Strength
  - Factors of Authentication (1, 2, and 3)
- Password Use
- Session ID protection
- Forgotten Password

Cross Site Request Forgery

- Submitting a form without a random token
- Allows an attacker to submit a request from Site A to change the state of something in Site B
- Grails Prevention:
  - use <g:form **useToken="true"** ...>

Cross Site Request Forgery

- http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243
- <img src="http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct#“ width="0" height="0" />

Insecure Configuration Management

- Unpatched security flaws in server software
- Default accounts with default passwords
- Unnecessary backup or sample files
- Secure information in log files
- Comments in code

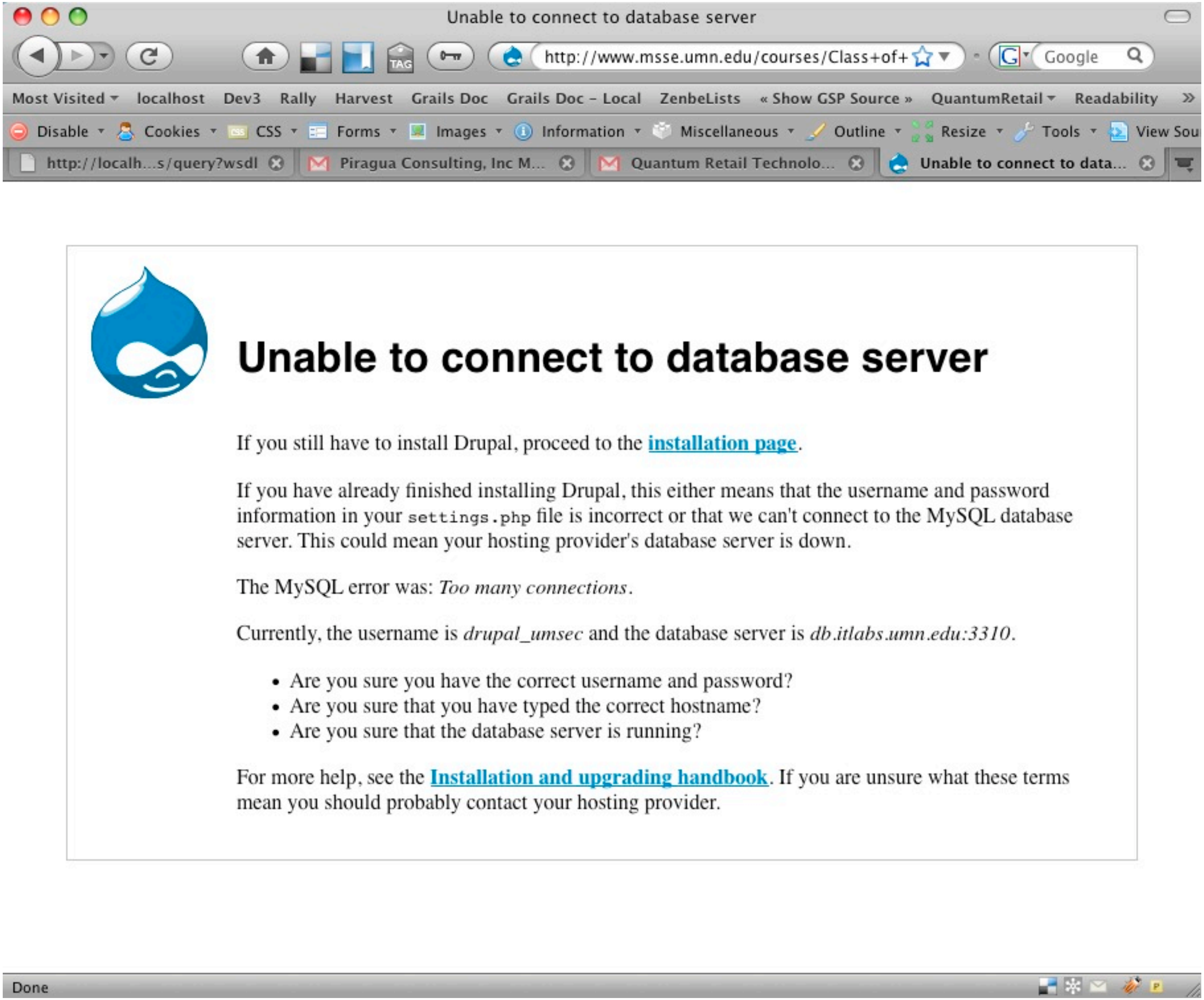
Defaults

- Default management applications (e.g. Tomcat Manager, Linksys Router)
- Default username / passwords

```
<tomcat-users>
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
</tomcat-users>
```

Improper Error Handling

- Errors give clues about your application
- Don’t display stack traces to users
- Log errors; display an error code
- “File not Found” vs. “Access Denied”



Unable to connect to database server

If you still have to install Drupal, proceed to the **installation page**.

If you have already finished installing Drupal, this either means that the username and password information in your `settings.php` file is incorrect or that we can't connect to the MySQL database server. This could mean your hosting provider's database server is down.

The MySQL error was: *Too many connections*.

Currently, the username is *drupal_umsec* and the database server is *db.itlabs.umn.edu:3310*.

- Are you sure you have the correct username and password?
- Are you sure that you have typed the correct hostname?
- Are you sure that the database server is running?

For more help, see the **Installation and upgrading handbook**. If you are unsure what these terms mean you should probably contact your hosting provider.

Insecure Storage

- Encrypt sensitive information
- Use one way hash for passwords, with a salt
- Beware of backups
- Use an encryption algorithm open to public scrutiny
- If you don’t need it, don’t store it

Broken Access Control

- Insecure IDs
- Forced Browsing Past Access Control Checks
- File Permissions
- Browser Cache
- Path Traversal

Secure?

- /person/show/3

Unvalidated Input

- Consider all input to be "tainted"
  - never use an input parameter in code until it has been validated
- Client Side Validation is not enough
- Don't trust hidden fields
- Use positive (not negative) validation

Positive Validation

- Data type (string, integer, real, etc…)
- Allowed character set
- Minimum and maximum length
- Whether null is allowed
- Whether the parameter is required or not
- Whether duplicates are allowed
- Numeric range
- Specific legal values (enumeration)
- Specific patterns (regular expressions)

Data Binding

```
def p = Person.get(1)
p.properties['firstName','lastName'] = params
```

```
def p = new Person()
bindData(p, params, [include:['firstName','lastName]])
```

```
def p = new Person()
bindData(p, params, [exclude:'dateOfBirth'])
```

Denial of Service

- Utilize Load Balancing
- Make it difficult to start a new session
- Perform load testing
- Throttle requests - e.g. one request per user session at a time
- Don’t allow unauthenticated users to perform “expensive” operations

```
def list = {
    params.max = Math.min( params.max?.toInteger() ?: 0, 100)
    [bookList: Book.list(params)]
}
```