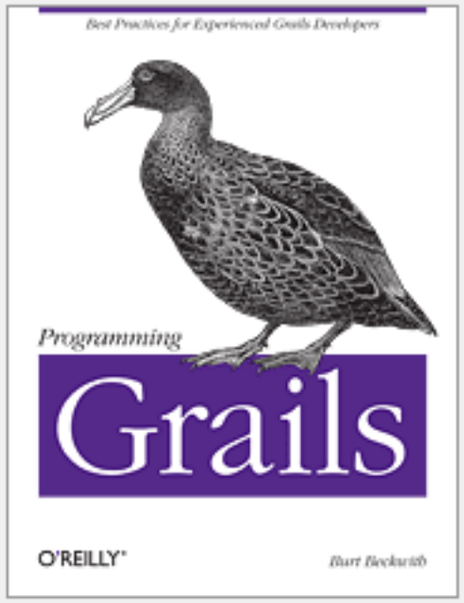
Securing Grails Applications

Burt Beckwith

SpringSource

Who Am I

- Java developer for ~14 years
- Background in Spring, Hibernate, Spring Security
- Grails developer for ~5 years
- SpringSource employee on the Grails team
- 50+ Grails plugins




OWASP
The Open Web Application Security Project

OWASP

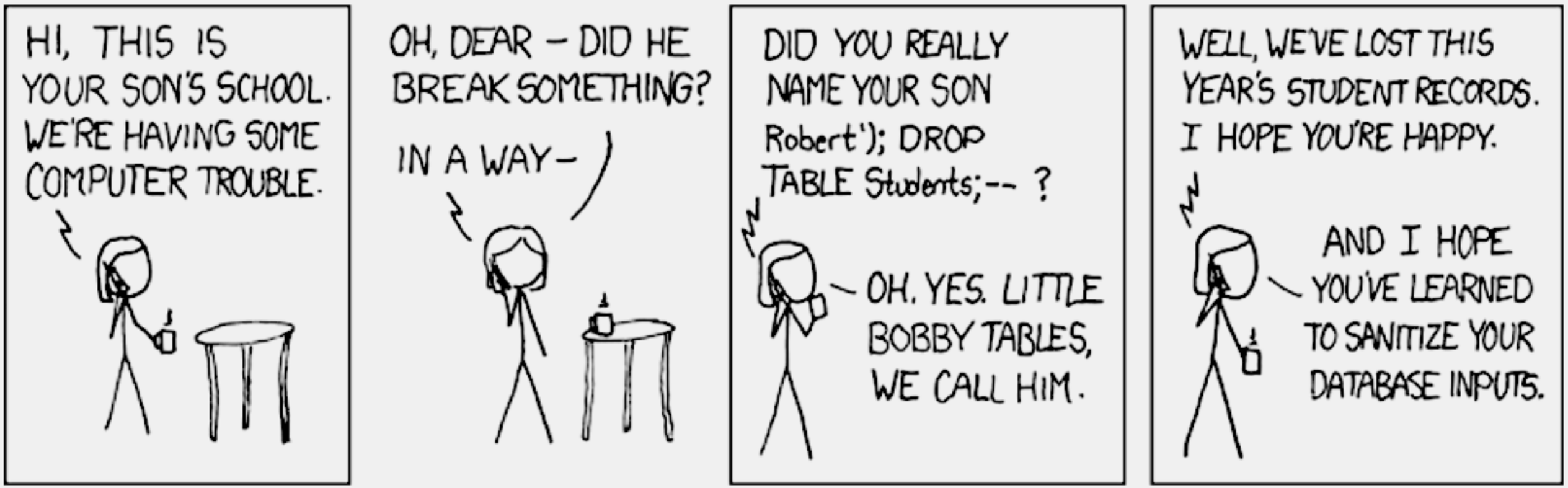
The Open Web Application Security Project

- Publishes top 10 highest-priority security risks
- AntiSamy Project
- ESAPI (The OWASP Enterprise Security API)
- WebScarab and other testing tools
- https://www.owasp.org/index.php/Grails_Secure_Code_Review_Cheat_Sheet

OWASP: A1-Injection

- SQL injection is the most common type
- Grails applications are largely immune
- Dynamically generated SQL/HQL query without properly escaping user input

```
String sql = "select * from person " +
             "where username ='" +
             params.username + "'"
```



https://xkcd.com/327/

OWASP: A1-Injection

- username = "master_of_disaster" is ok
- but username = "' or '1'='1" will generate

```
select * from person where username ='' or '1'='1'
```

OWASP: A1-Injection

- '; drop table users; --

  or

- '; truncate table users; --
- You'll be scrambling to restore the database from the most recent backup

OWASP: A1-Injection

- Use PreparedStatement with placeholders

```
String sql = "select * from person " +
             "where username = ?"
PreparedStatement ps = conn.prepareStatement(sql)
ps.setString(1, params.username)
```

OWASP: A1-Injection

- Likewise for HQL

```
Person.executeQuery(
    'from Person where username=?',
    [params.username])
```

or

```
Person.executeQuery(
    'from Person where username=:username',
    [username: params.username])
```


Don't trust the users!
http://cdn.morguefile.com/imageData/public/files/r/ronnieb/preview/fldr_2005_05_26/file000331196237.jpg

A2: Cross-Site Scripting (XSS)

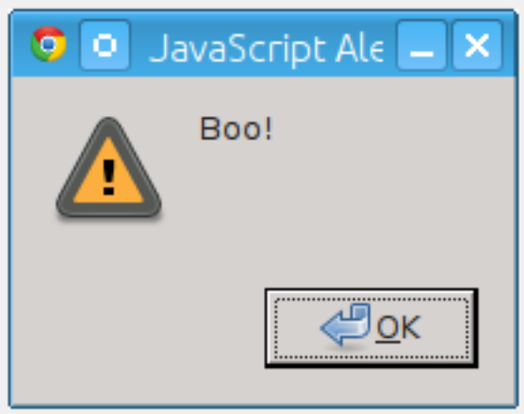
- Takes advantage of unescaped user-supplied input that you display
- e.g. search form, comments, wikis
- "Script injection"


https://secure.flickr.com/photos/conchur/1573136674

A2: Cross-Site Scripting (XSS)

"search" for "<script>alert('Boo!')</script>"

```
<div id='query'>
Your search: "<script>alert('Boo!')</script>"
</div>
```



A2: Cross-Site Scripting (XSS)

Risks

- Bypass security checks
- Access authentication cookies
- Executing a client-side GET or POST

A2: Cross-Site Scripting (XSS)

Remedies

```
grails.views.default.codec = "html"
```

```
<div id='query'>
Your search: "&lt;script&gt;alert('Boo!')&lt;/script&
</div>
```

A2: Cross-Site Scripting (XSS)

Remedies

- Don't use <% ... %> when untrusted

```
<%=person.name%>
```

A2: Cross-Site Scripting (XSS)

Remedies

- Servlet 3.0 - use 'HttpOnly' flag
- Use SSL
- AntiSamy, ESAPI, and HDIV

A3: Broken Authentication and Session Management

A3: Broken Authentication and Session Management

Remedies

- Use Spring Security/Shiro/Commercial option
- Use Bcrypt or salted hash
- Disable URL rewriting
- Use SSL

A3: Broken Authentication and Session Management

Remedies

- Use session fixation protection

```
grails.plugins.springsecurity.
useSessionFixationPrevention = true
```

A3: Broken Authentication and Session Management

- Don't "roll your own"

A4: Insecure Direct Object References

```
<li><a href='/card/view/42'>
Purchases for card xxxx-xxxx-xxxx-1234
</a></li>

<li><a href='/card/view/54312'>
Purchases for card xxxx-xxxx-xxxx-1337
</a></li>
```

A4: Insecure Direct Object References

- Don't use

```
def card = CreditCard.get(params.id)
```

A4: Insecure Direct Object References

- Do this instead

```
def userId = ... // retrieve from auth
def user = User.load(userId)
def card = CreditCard.findByIdAndOwner(
      params.id as Long, user)
```

A4: Insecure Direct Object References

- Or use ACLs

```
@PreAuthorize("hasPermission(
    #id, 'com.yourapp.CreditCard', read)")
CreditCard getCard(long id) {'
    CreditCard.get(id)'
}
```

A4: Insecure Direct Object References

- HDIV library will replace real identifiers with placeholder values

```
<li><a href='/card/view/0'>
Purchases for card xxxx-xxxx-xxxx-1234
</a></li>

<li><a href='/card/view/1'>
Purchases for card xxxx-xxxx-xxxx-1337
</a></li>
```

A5: Cross-Site Request Forgery (CSRF)

- Similar to an XSS attack
- Make requests on other users' behalf and send personal information/cookies/session ids

```
<img src='http://hacker.site.com?foo=bar'/>

<img src='/account/transfer?
     destination=123&amount=1000'
```

A5: Cross-Site Request Forgery (CSRF)

Remedies

- Fix XSS
- Use POST for updates
- Generate a token for each link & form
- http://www.hdiv.org/
- http://grails.org/plugin/hdiv

A6: Security Misconfiguration

- Issues with application, server,

  or operating system code

A6: Security Misconfiguration

Remedies

- Uninstall unneeded OS apps & features
- Remove default/automatically created accounts
- Disable unnecessary features
- Never use a default password
- Routinely update software & OS
- Reduce the "attack surface"

A7: Insecure Cryptographic Storage

- Two general approaches: hashing & encrypting
- Hash generates a fixed-length output;

  one-way conversions
- Encryption functions are designed to be decrypted

A7: Insecure Cryptographic Storage

- Common use of hash is password storage
- Do not use MD5 or SHA-1
- Use Bcrypt & large number of iterations

  or SHA-256/SHA-512 + salt

A7: Insecure Cryptographic Storage

- Encrypt "sensitive" information
- http://www.jasypt.org/
- http://grails.org/plugin/jasypt-encryption
- http://www.bouncycastle.org/

A7: Insecure Cryptographic Storage

Remedies

- Don't store passwords on the filesystem, in code, or in source control
- Rotate keys and change passwords frequently
- Use crypto keys that are at least 128 bits

A7: Insecure Cryptographic Storage

Remedies

- Don't write sensitive information to log files
- Avoid native database encryption
- Never implement your own hash or crypto algorithm!

A8: Failure to Restrict URL Access

A8: Failure to Restrict URL Access

Remedies

- Spring Security or Shiro plugin

  or commercial option, or app server

```
grails.plugins.springsecurity.
rejectIfNoRule = true
```

A8: Failure to Restrict URL Access

- Relatively easy to test
- Need a good sitemap
- Mechanical process to functional test URLs

A9: Insufficient Transport Layer Protection

- Easy to intercept network traffic
- Use SSL
- Session id cookie is tricky

```
grails.plugins.springsecurity.auth.
forceHttps = true
```

A9: Insufficient Transport Layer Protection

- Use "channel security" configuration
- Need proper SSL configuration
- Renew certificates as needed
- Purchase from reliable vendor
- Use a key size of at least 128 bits

A9: Insufficient Transport Layer Protection

- Can encrypt traffic between servers
- e.g. MySQL allows SSL

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON database_name.* TO username@servername
IDENTIFIED BY 'the password' REQUIRE SSL;
```

A10: Unvalidated Redirects and Forwards

- Could use an XSS vulnerability to redirect/forward to another site
- Could possibly install malware or trick them into revealing information

A10: Unvalidated Redirects and Forwards

- Don't do this:

  /some/url?nextPage=/user/home

- Easy to misuse:

  /some/url?nextPage=http://othersite.com/...

A10: Unvalidated Redirects and Forwards

Remedies

- Use flash scope to store the next URL
- In email use

/some/url?np=a2

ESAPI

OWASP Enterprise Security API

https://www.owasp.org/index.php/

Category:OWASP_Enterprise_Security_API

ESAPI

- Input validation for text data
- Output encoding for web
- http://grails.org/plugin/xss-sanitizer
- Also in spring-security-core 2.0

ESAPI

```
org.owasp.esapi.ESAPI.encoder()
encodeForCSS()
encodeForHTML()
encodeForHTMLEntity()
encodeForJavascript()
```

and more!

AntiSamy

OWASP AntiSamy Project

https://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project

AntiSamy

- HTML input validation & cleaning
- HTML output translation
- Uses policy files

AntiSamy

The Open Web Application Security Project

- http://grails.org/plugin/sanitizer
- Also in spring-security-core 2.0

HDIV

HTTP Data Integrity Validator

- http://www.hdiv.org/
- Maintains APIs through interception
- Data integrity of unmodifiable data

  (links, hidden fields, combo values, etc.)
- CSRF Tokens in forms and links

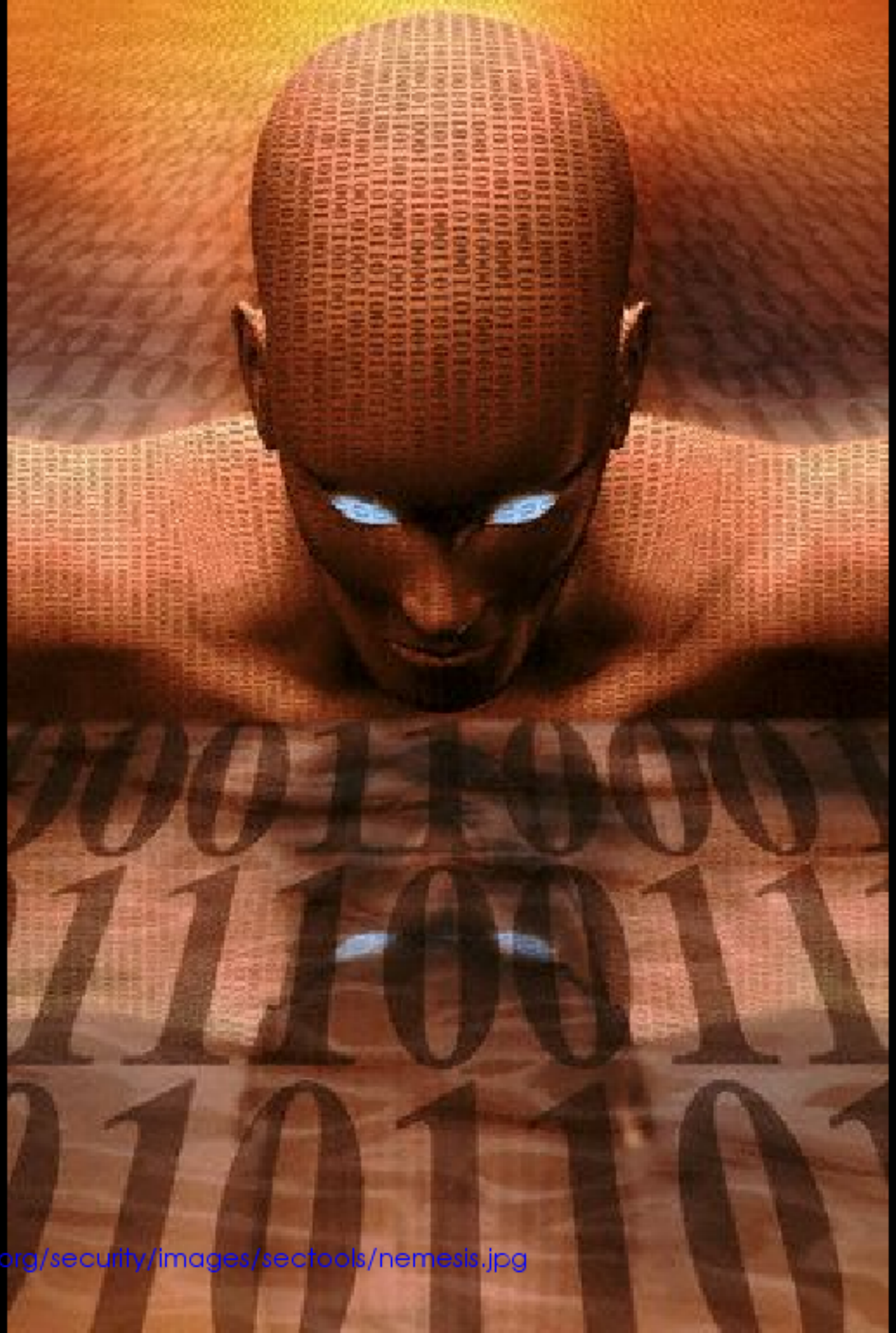
HDIV

HTTP Data Integrity Validator

- Support in Spring MVC as of 3.1

```
org.springframework.web.servlet.
support.RequestDataValueProcessor
```

- Support in Grails as of 2.3
- http://grails.org/plugin/hdiv



http://www.compsec.org/security/images/sectools/nemesis.jpg

This presentation made with

Q & A

Courses

- Course Search
- Degree Course Search
- Tips for Choosing Courses
- January Courses
- Fall Syllabi & Websites

Home » Courses » Computer Science » Web Application Development with Groovy and Grails

CSCI E-56 Web Application Development with Groovy and Grails

This course provides a comprehensive overview of using the Groovy language and the Grails framework to rapidly create real-world web applications. Students learn front-end technologies (MVC, Ajax, REST), persistence with GORM using Hibernate and NoSQL datastores, convention-over-configuration, and application and plugin development best practices. Topics include artifacts, internationalization, building and deploying, testing, integration, security, and performance. Groovy topics including dynamic and static typing, closures, DSLs, and the meta-object protocol.

Prerequisite: experience with Java or another object-oriented programming language and some server-side web development. (4 credits)

Fall term 2013 (14325)

Burt Beckwith, MS, Senior Software Engineer, SpringSource.

Class times: Fridays beginning Sept. 6, 5:30-7:30 pm. Required sections to be arranged.

Course tuition: noncredit $2,050, undergraduate credit $2,050, graduate credit $2,050.

Online option available.

Thank you!