Hack-proof Your Drupal App

Key Habits of Secure Drupal Coding

DrupalCamp CT 2010

Introductions

Erich Beyrent

http://twitter.com/ebeyrent

http://drupal.org/user/23897

My Modules

- Permissions API
- Crowd SSO
- LDAP Extended Groups
- Search Lucene Biblio
- Search Lucene Attachments
- Search Lucene OG
- Visual Search API

Agenda

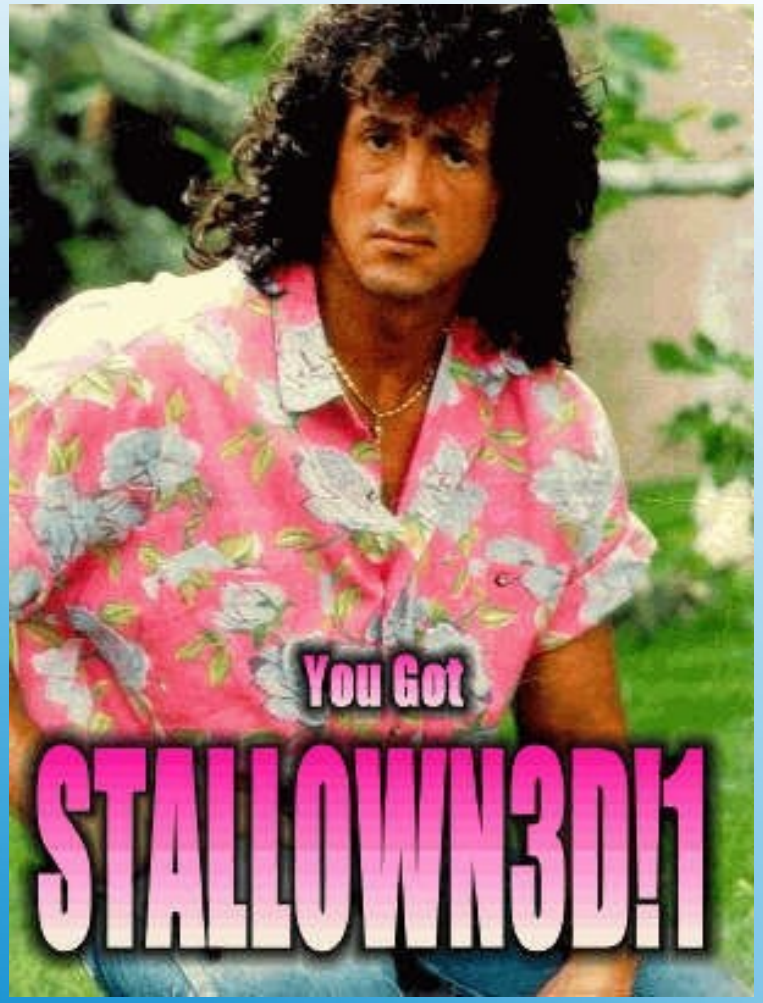
- Secrets to Securing a Social Network
- Key Habits of Secure Drupal Coding
- Vulnerability Detection to Remediation
- Security Resources for Drupal Applications
- See For Yourself - demonstrations of application attacks
- Discussions

Have you ever...


STORY TIME!

- In 2008, I started work on Greenopolis.com
- It was my first big Drupal project
- Lots of custom modules
- Custom theme
- Prior to launch, a security scan was performed

HILARITY DID NOT ENSUE



The Results

- 120 vulnerabilities were discovered
  - XSS
  - CSRF
  - SQL Injection
  - Insufficient Authorization

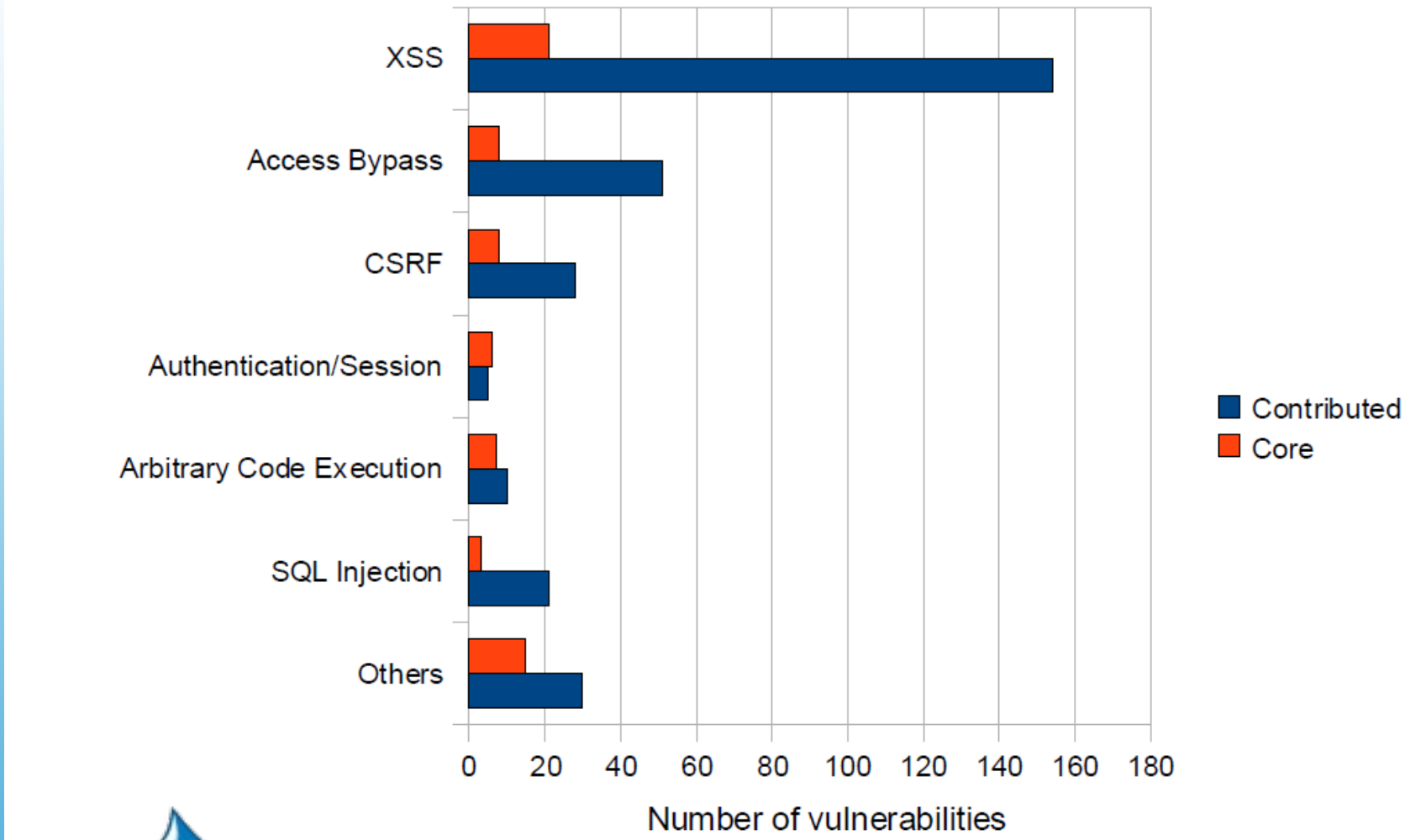
What Was Learned

- 90% of the vulnerabilities existed in the theme
- Untrusted data from the query string was used
- Custom search forms were insecure
- crossdomain.xml caused vulnerabilities

Fixing The Problems

- Complete review of the theme, implementing Drupal output filters
- Code was audited to ensure sanitization of all user data
- Rewrite of the search forms to sanitize user data
- Implemented web services proxy

June 2005 – March 2010



Source: Drupal Security Report
http://drupalsecurityreport.org/

Key Habits of Secure Drupal Coding

- Wrap your output

Key Habits of Secure Drupal Coding

- Wrap your output
- Protect your database

Key Habits of Secure Drupal Coding

- Wrap your output
- Protect your database
- Beware user input

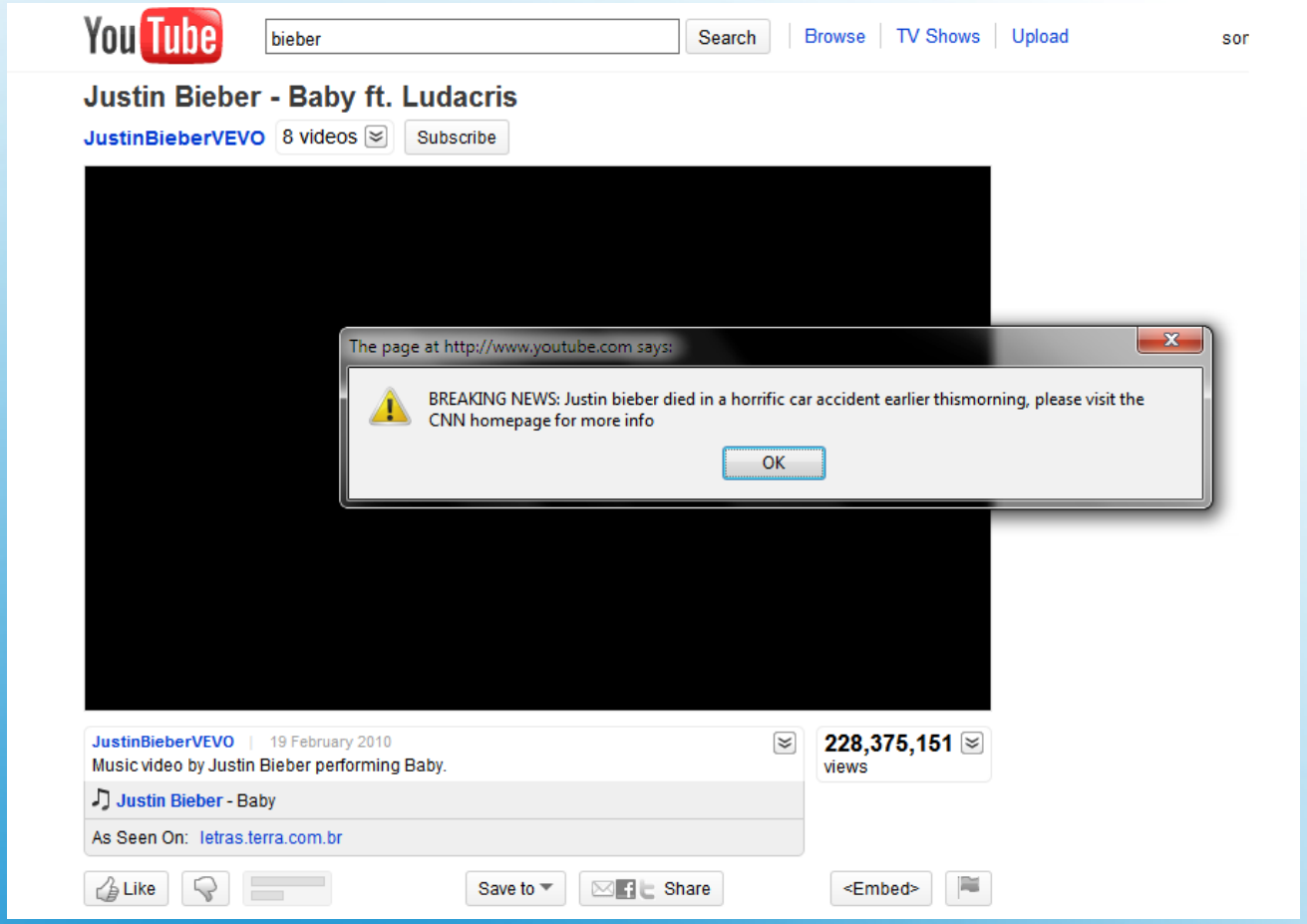
Key Habits of Secure Drupal Coding

- Wrap your output
- Protect your database
- Beware user input
- AJAX risks

Reality

- Security experts estimate that 66% of websites are vulnerable to XSS attacks (Jeremiah Grossman, WhiteHat Security)

Reality



YouTube (July 2010)

Wrap Your Output

`check_plain()`

check_plain()

- This is for simple text without any markup.
- Encodes special characters in a plain-text string for display as HTML.
- Uses drupal_validate_utf8 to prevent cross site scripting attacks on Internet Explorer 6.
- Don't use this when using the t(), l(), theme('placeholder')

Wrap Your Output

`check_plain()`

`check_markup()`

check_markup()

- This is for text which contains markup in some language
- Runs all the enabled filters on a piece of text.

Wrap Your Output

```
check_plain()
check_markup()
filter_xss()
```

filter_xss()

- Filters an HTML string to prevent cross-site-scripting (XSS) vulnerabilities.
  - Removes characters and constructs that can trick browsers.
  - Makes sure all HTML entities are well-formed.
  - Makes sure all HTML tags and attributes are well-formed.
  - Makes sure no HTML tags contain URLs with a disallowed protocol (e.g. javascript:).

Source: http://api.drupal.org/api/function/filter_xss

Wrap Your Output

```
check_plain()
check_markup()
filter_xss()
filter_xss_admin()
```

filter_xss_admin()

- Filters an HTML string to prevent cross-site-scripting (XSS) vulnerabilities.
  - Removes characters and constructs that can trick browsers.
  - Makes sure all HTML entities are well-formed.
  - Makes sure all HTML tags and attributes are well-formed.
  - Makes sure no HTML tags contain URLs with a disallowed protocol (e.g. javascript:).

Source: http://api.drupal.org/api/function/filter_xss

Protect Your Database

```
db_query()
```

db_query()

- Runs a query in the database with arguments to the query, passed in as separate parameters, which are escaped to prevent SQL injection attacks.

db_query()

- CORRECT:
  - db_query("INSERT INTO {table} VALUES (%d, '%s')", $node->profile_age, $node->profile_firstname);
- WRONG:
  - db_query("SELECT * FROM table WHERE field = $node->profile_age");

Protect Your Database

`db_query()`

`db_rewrite_sql()`

db_rewrite_sql()

- Rewrites node, taxonomy and comment queries to respect Drupal's node access mechanism.
- Protects against unauthorized access to content.

db_rewrite_sql()

- CORRECT:
  - db_query(db_rewrite_sql("SELECT * FROM {node} WHERE uid = %d", $uid));
- INCORRECT:
  - db_query("SELECT * FROM {node} WHERE uid = %d", $uid);

Beware User Input

- Sources of user input:
  - Form fields
  - Uploaded files
  - Query string
  - Other sites

AJAX Risks

- AJAX transactions are not private
- Eval() is not 100% safe; use JSONP

Things Good Drupalers Do

- Sanitize output
- Use the Form API
- Use parameterized queries
- Leave core intact

Things That Will Bite You

- Printing raw values
- Modifying data with $_GET
- Parameterized queries? WTF?
- Hacking core and killing kittens

Other Common Mistakes

- ```
  <?php
  global $user;
  // Bad - this will escalate the privileges
  $user = user_load(array('uid' => $uid));
  ?>
  ```
- ```
  <?php
  global $user;
  // SAFE - do this instead
  $account = user_load(array('uid' => $uid));
  ?>
  ```

Other Common Mistakes

- Improper URL access
  - Incorrect usage of 'access callback' in hook_menu()
  - Lack of security settings on views
- Writing forms in HTML
  - Use the Form API to provide automatic CSRF protection

Other Common Mistakes

- Unvalidated and open redirects
  - Iframes, drupal_goto, location.href
- Promiscuous crossdomain.xml files

Don't Trust User Input!

Drupal Security Resources

- http://drupal.org
- Writing Secure Code (http://drupal.org/writing-secure-code)
- Handle Text in a Secure Fashion (http://drupal.org/node/28984)
- Drupal Security Team

Modules

- Coder (http://drupal.org/project/coder)
- Security Review (http://drupal.org/project/security_review)
- Secure Code Review (http://drupal.org/project/secure_code_review)
- Secure Permissions (http://drupal.org/project/secure_permissions)

Books

- Pro Drupal Development book (VanDyk)
- Cracking Drupal: A Drop in the Bucket (Knaddison)
- XSS Scripting Attacks (Grossman)

Questions?