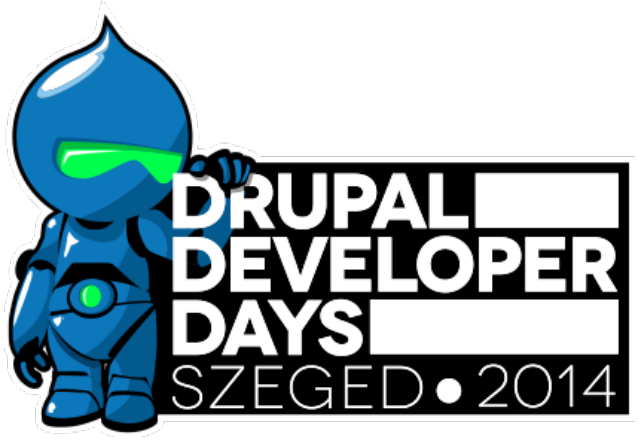


Cracking Drupal

Security concepts and pitfalls

Klaus Purer, Software Engineer at epiqo.com
http://klau.si Twitter: @_klausi_
Drupal Developer Days Szeged 2014

Security strategies

- **Trust** - who can do what
- **Principle of least privilege** - lock down permissions as far as possible
- **Defense in depth** - multi layered protection to have fallbacks
- **Software updates** - rule out obvious exploits in Drupal, PHP, operating system, browser etc.

OWASP Top 10

- Open Web Application Security Project
- List of most critical security risks
- Assessment of attack vector, weakness and impact

https://www.owasp.org/index.php/Top_10_2013

1. Injection

Attacker supplied parameters are passed to an interpreter

SQL injection:

```
<?php
db_query("SELECT uid FROM {users} u WHERE u.name = '" .
$_GET['user'] . "'");
?>
```

1. Injection (2)

Exploit: http://example.com/?user=x%27%3B%20DROP%20table%20node%3B%20--

Query:

```
SELECT uid FROM users u WHERE u.name = 'x'; DROP table node; --'
```

This will delete your node table => data loss + Drupal unable to bootstrap.

Preventing SQL injection

Correct usage:

```
<?php
db_query("SELECT uid FROM {users} u WHERE u.name = :name",
  array(':name' => $_GET['user']));
db_select('users', 'u')
    ->fields('u', array('uid'))
    ->condition('u.name', $_GET['user'])
    ->execute();
?>
```

Docs: https://drupal.org/writing-secure-code

1. Injection (3)

Remote code execution:

```php
<?php
// Obvious: never pass user supplied text to the PHP
// interpreter!
eval($_POST['some_field']);
// http://heine.familiedeelstra.com/security/unserialize
unserialize($_POST['some_field']);
?>
```

High impact vulnerabilities!

Injection (4)

```php
<?php
// Never use the /e modifier, deprecated since PHP 5.5
preg_replace('/^(.*)/e', 'strtoupper(\\1)', $_POST
['some_field']);
?>
```

2. Authentication & sessions

- Choose good passwords
- Hash your passwords
- Protect your session IDs

Drupal core covers this pretty well.

Configure **HTTPS** to not transmit unencrypted session IDs.

- https://drupal.org/project/securepages
- https://drupal.org/project/securelogin

3. Cross-Site Scripting (XSS)

- Attackers can inject Javascript tags
- all user provided text needs to be sanitized before printing to HTML
- (admin) user interaction is required

Reflected XSS example:

```
<?php
print 'You are on page number ' . $_GET['number'];
?>
```

Penetration test: `<script>alert('XSS');</script>`

Persistent XSS

Injected Javascript is stored in the database

Vulnerable, because of the node title:

```php
<?php
foreach ($nodes as $node) {
  $rows[] = array($node->nid, $node->title);
}
$render_array = array('#theme' => 'table','#rows' => $rows);
return $render_array;
?>
```

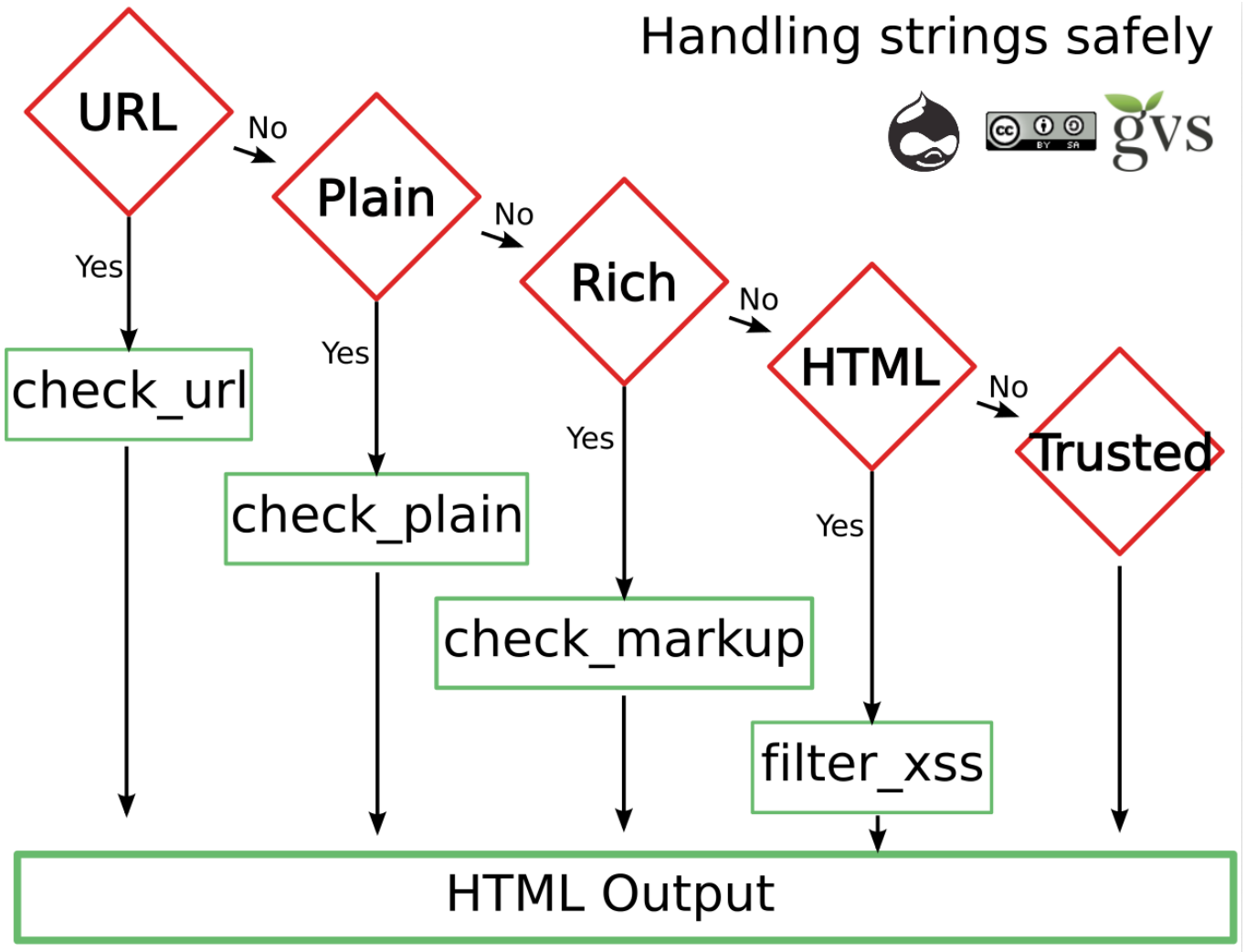
Preventing XSS

```php
<?php
foreach ($nodes as $node) {
  $rows[] = array($node->nid, check_plain($node->title));
}
$render_array = array('#theme' => 'table','#rows' => $rows);
return $render_array;
?>
```

Handling text securely: https://drupal.org/node/28984

Filtering on output

When handling data, the golden rule is to store exactly what the user typed. When a user edits a post they created earlier, the form should contain the same things as it did when they first submitted it. This means that **conversions are performed when content is output**, not when saved to the database.



http://drupalscout.com/knowledge-base/drupal-text-filtering-cheat-sheet-drupal-6

Mitigating XSS

What Drupal core does for us:

- Drupal sets the HTTPOnly flag on session cookies to prevent cookie stealing in JS
- User form: password change requires current password (since Drupal 7)
- Text formats for different user roles

We still need to rigorously escape user input.

Content Security Policy

- W3C Browser standard: no more inline JS execution!
- JS resources (domains) whitelist in HTTP headers
- supported by newer browsers, but not all (example: Android browser)

So we still need to rigorously escape user input (sigh).

4. Insecure Direct Object References

Category: Access bypass vulnerabilities

Happens rarely for Drupal, just use the user permission and access APIs.

5. Security misconfiguration

- Display of PHP error reporting

Disable at /admin/config/development/logging

- PHP filter module, disable at /admin/modules
- PHP files writeable by the web server

Remove write permissions for www-data

```
-rw-r-----  1 deployer   www-data  index.php
drwxr-x--- 32 deployer   www-data  modules/
drwxrwx---  7 www-data   deployer  sites/default/files/
```

Docs: https://drupal.org/security/secure-configuration

Permissions

- Careful with restricted, site-owning permissions (which roles do you trust?)
- Same for text formats (full HTML!)
- Do not use the user 1 account in your daily work, it has all permissions
- User 1 name should not be “admin”

Private files configuration

Move the private files directory outside of the docroot to avoid direct downloads:

```
example.com
|+ conf
|- docroot
  |- index.php
  |- ... other Drupal files ...
|- private
  |- secret_picture.png
  |- ... other private files ...
|+ tmp
```

PHP file execution

- Drupal uses the front controller pattern: almost everything goes through **index.php**
- Disallow execution of PHP files in subfolders
- Prevents PHP execution in files directory

Apache example:

```
RewriteRule "^.+/.*\.php$" - [F]
```

Nginx example:

```
location ~* ^.+/.*\.php$ { deny all; }
```

6. Sensitive Data Exposure

- **Encrypt sensitive data** such as credit card numbers in your database. Even better: don’t store them if you don’t have to.
- Again, use **HTTPS** for authenticated sessions to not send transmit data in plain text.
- User **passwords** are properly hash-salted by Drupal core.

7. Missing Function Level Access Control

Access bypass in hook_menu():

```php
<?php
function mymodule_menu() {
  $items['admin/mymodule/settings'] = array(
    'title' => 'Admin configuration',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('mymodule_admin_form'),
    'access callback' => TRUE,
  );
  return $items;
}?>
```

Using permissions

Protect your menu entries:

```php
<?php
function mymodule_menu() {
  $items['admin/mymodule/settings'] = array(
    'title' => 'Admin configuration',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('mymodule_admin_form'),
    'access arguments' => array('administer mymodule'),
  );
  return $items;
}?>
```

Node access bypass

Example: some expense report nodes are access restricted depending on the user role.

```
<?php
$records = db_select('node', 'n')
  ->fields('n')
  ->condition('type', 'expense_report')
  ->execute()
  ->fetchAll();
// ... load and render list of nodes somehow.
?>
```

Correctly using node access

Limit the list of nodes with the node_access tag:

```
<?php
$records = db_select('node', 'n')
  ->fields('n')
  ->condition('type', 'expense_report')
  ->addTag('node_access')
  ->execute()
  ->fetchAll();
// ... load and render list of nodes somehow.
?>
```

8. Cross-Site Request Forgery (CSRF)

```
function mymodule_menu() {
  $items['mymodule/pants/%/delete'] = array(
    'title' => 'Delete pants',
    'page callback' => 'mymodule_delete_pants',
    'page arguments' => array(2),
    'access arguments' => array('delete pants objects'),
  ); return $items;
}
function mymodule_delete_pants($pants_id) {
  db_delete('mymodule_pants')
    ->condition('pants_id', $pants_id)->execute();   }
```

Exploiting CSRF

Attacker posts a comment somewhere:

```
<img src="http://example.com/mymodule/pants/1337/delete">
```

Chain of an attack:

- Logged-in admin visits comment page
- Browser fetches the image src and sends cookies along
- Request is successfully authorized
- Delete query is executed
- Pants 1337 are gone.

http://epiqo.com/en/all-your-pants-are-danger-csrf-explained

Protecting against CSRF

Write operations need to be protected with:

- Confirmation forms or
- Security tokens in the URL

```
http://example.com/mymodule/pants/1337/delete?
token=tLBSLWTZVpRmp1cD_I4hCKd2vS-dJbv6xxTICKr3DHM
```

POST requests: always use the Form API!

Docs: https://drupal.org/node/178896

9. Using Components with Known Vulnerabilities

Widespread attack vector, often automated

- Update all your software regularly
- Monitor security mailing lists, RSS feeds etc.
- Enable Drupal's update status notifications
- Security advisories at https://drupal.org/security
- Disable software components that are not used

10. Unvalidated Redirects and Forwards

Vulnerability:

```
<?php
drupal_goto($_GET['target']);
```

Exploit example that redirects to evil.com:

```
http://example.com/cart?target=http%3A%2F%2Fevil.com
```

Perfect for phishing attacks. Correct:

```
<?php
if (!url_is_external($_GET['target'])) {
  drupal_goto($_GET['target']);
}
```

Do you see the pattern?

- Don’t trust any user provided data in the URL, the request or content in the database
- Attackers use browser features to perform actions behind the user’s back (XSS, CSRF, open redirects)
- Attackers use known vulnerabilities and automated tools to mass-hijack sites

Denial of Service (DoS)

Goal: bring the site down, make it unresponsive

How: execute expensive actions very often

Example from SA-CONTRIB-2013-053:

```php
<?php
// Too many login attempts, delay the response.
@sleep($sleep_time);
?>
```

sleep() will bind server process resources, too!

How to recover from an attack

- Determine what was compromised and when
- Restore from backup
- Update code (and server software)
- Change all passwords and keys
- Audit your code (custom modules first!)
- Scan logs for traces of the attacker (Drupal watchdog log, web server log, syslog etc.)

Useful security modules

- Security Review: check your site for misconfiguration https://drupal.org/project/security_review
- Paranoia: no PHP eval() from the web interface https://drupal.org/project/paranoia
- Seckit: Content Security Policy, Origin checks against CSRF https://drupal.org/project/seckit

Thank you!

Questions?

Klaus Purer, Software Engineer at epiqo.com
http://klau.si Twitter: @_klausi_