

Securing your Drupal site

Ivica Puljic

28. Nov. 2009

- **Goal:** raise security awareness of Drupal Community
 - Target audience: clients&managers, coders, themers, site admins
- **Type of attacks**
 - Learn about the sources of danger & how to protect yourself
- **Howto:**
 - Administration of Drupal in Secure Way
- **Practical examples: bad practice**
 - Things that you should not try home alone

- It is **easy** to open a security hole with **just couple of clicks**
- If security **is** your concern, you should learn more about:
 - Drupal security API
 - User permissions and menu system
 - “Never trust user inputs“: database sanititation, filtering content, Form API
 - Safe Drupal theming
 - Drupal security modules
 - Automated Security Testing: Drupal modules / Penetration testing tools

Why should I care?



MONTENASOFT

Some serious numbers first

- Gartner group
 - ♦ 75% of hacks are at the web app level
 - ♦ Result: out of 300 audited sites, 97% are vulnerable to attacks
- **WhiteHat** study (2006)
 - ♦ security of largest/popular websites according to WASC threat classif.
 - ♦ Result: 7 out of 10 websites have serious vulnerabilities
- Partner company www.netsec.rs (security auditors)
 - ♦ **Recent audit of major Serbian banks** (34 tested banks, in Serbian)
 - ♦ Result: after 10 minutes only 8 of 34 did not show severe vulnerabilities
 - ♦ All vulnerabilities that they found are listed in TOP 10 OWASP list

Which means (in “security haiku”)...

If you develop
a web site
it probably
can be hacked.



But hey... why should I care?

Most of us are not creating ebay.com or an bank applications...

... so we are not interesting as a target for crackers?

- Even small blog, forum or small company site are hacked
 - **they** want your emails, passwords, usernames: for spamming or to try to get into your bank
- *they* want to create a spam relay from your site →
 - **you are** source of malware software for your site visitors
 - **your box** is used for attacking other sites
 - ...
 - ...

That horrible sneaking fealing



Why webapps are gold for crackers?

- Web is **everywhere** and it's here to stay
- **Money:** more hackers and crackers are working for money now
- Hacking web apps is **easy** compared to hacking os / other apps
- **Anonymity:** easy to hide an attack / international laws situation is bad
- It **bypasses firewall:** port 80 is always open
- Developers have **poor knowledge** about security
- **Immature protocol:** http don't even implement basic session handling
- **Constant change:** in a life of an webapp people come... and go away: developers, system admins, site admins, content editors...
- Web apps are **very complex - plus multiple layers:** os, web server, web platforms, frameworks, database, client side

Unfortunately, at the end of a day...

Nobody wants
to take care
about
security.



Week spot of web applications

For Drupal developer who wants to deliver an applications, security do not ends with proper use of Drupal security API:

- ♦ **os platform**
- ♦ **web server** - apache/iis
- ♦ **web platform** - asp.net, php
- ♦ **other services** – or you don't care if anonymous ftp login is enabled?
- ♦ **web applications** - attacks against authentication & authorization, site structure, input validation, app logic
- ♦ **database** - sql injection
- ♦ **web client** - active content execution, xss, phishing...
- ♦ **Transport** - interfere in client/server communication and ssl redirections
- ♦ **availability** - DoS attacks

Who knows what is owasp?

- ♦ **OWASP** - Open Web Application Security Project
- ♦ WebGoat
- ♦ WebScarab

- ♦ 1. XSS
- ♦ 2. Injection flaws - mostly sql injections
- ♦ 3. RFI - remote file execution
- ♦ 4. Insecure Direct Object Reference
- ♦ 5. CSRF
- ♦ 6. Information Leakage and Improper Error Handling
- ♦ 7. Broken Authentication and Session Management
- ♦ 8. Insecure Cryptographic Storage: example drupal & md5 pwd encryption
- ♦ 9. Insecure Communications
- ♦ 10. Failure to Restrict URL Access

- ♦ 1. Injections flaws
- ♦ 2. XSS
- ♦ 3. Broken Authentication and Session Management
- ♦ 4. Insecure Direct Object Reference
- ♦ 5. CSRF
- ♦ 6. Security misconfiguration
- ♦ 7. Failure to Restrict URL Access
- ♦ 8. Unvalidated redirects and forwards
- ♦ 9. Insecure Cryptographic Storage
- ♦ 10. Insecure Communications

And don't think this is all

or almost all - there are
hundreds and hundreds of
attack vectors more on the list
...

those are just the top 10

TOP Drupal vulnerabilities

VULNERABILITY	OCCURRENCES	OCCURRENCES AS A PERCENT OF THE TOTAL
XSS	55	44
Access bypass	17	14
CSRF	12	10
SQL injection	12	10
Code execution	10	8
Clarifications and announcements	4	3
Session fixation	3	2
Privilege escalation	2	4
Arbitrary file upload	2	4
Mail header injection	2	4
CAPTCHA bypass	2	4
HTTP response splitting	2	4
File overwrite	1	2
Logging sensitive data	1	2
Session impersonation	1	2

Source: [Cracking a Drupal, A Drop In The Bucket](#)
by Greg James Knaddison

How do they do it - Tools of trade

- ♦ Web browser - just with a browser you can do a lot
- ♦ IE plugins: TamperIE - tampers get and post data;
IEWatch - monitoring http protocol...
- ♦ FF plugins: firebug, web development,
TamperData - trace and modify http&https
- ♦ HTTP proxies - OWASP WebSarab is a http proxy, crawled/spider, session analysis, automation, encoder/decoder for base64, md5
- ♦ command line tools
- ♦ penetration testing tools - [metasploit framework](#), [nesus](#)
- ♦ [backtrack](#) linux distribution



How do they do it – step 1 – Infrastructure Profiling

- ♦ internet registry search, ping sweeps, nmap port scanning - now they have your ip and your ports, and then they continue with
- ♦ advanced http fingerprinting
- ♦ automation - httpprint tool
- ♦ detecting load balancer - often code on different machine is not the same - old versions, test versions...
- ♦ many others methods

How do they do it – step 2 – Web App Profiling

- ♦ manual - most important, its like detective work
- ♦ search engine - discover many things about a site and its history
- ♦ automated crawling - saves time but can be of much help with forms, complex forms, client side code, broken html/http

How do they do it – Penetration testing tools

- ♦ metasploit framework
- ♦ nesus
- ♦ nikto - web scanner
- ♦ Grendel-Scan



General protections methods related to Drupal site

Keep up with security patches

MONTENASOFT

MontenaSoft e.U., Neilreichgasse 47 / 1 / 22, A-1100 Wien, T: +43 699 19084898, F: +43 1 9247976, office@montenasoft.com, <http://www.montenasoft.com>

General protections methods related to Drupal site

Hardening Apache

- Disable unneeded modules
- Disable next modules: mod_userdir, mod_info, mod_status, mod_include
- implement **ModSecurity** - web application firewall. It provides protection from a range of attacks against web applications and allows for HTTP traffic monitoring, logging and real-time analysis. Request Filtering, Anti-Evasion Techniques, HTTP Filtering Rules, Full Audit Logging, HTTPS Intercepting, Chroot Functionality, Mask Web Server Identity
- Chrooting Apache
- Document root restriction – allow apache to only go to /www/htdocs
- - automated tool - "**Apache Benchmark from the Center of Internet Security**"

General protections methods related to Drupal site (2)

Hardening PHP

- ♦ configure php properly
 - turn off register_globals
 - open_basedir - restrict php file access to only certain directories
 - disable_functions
 - expose_php - remove php info from http headers
 - display_errors
 - safe_mode - php can use only files which it is an owner
 - allow_url_fopen
- ♦ suhoshi
 - php engine protection with couple of patches
 - range of runtime protection, session protection, filtering features and logging features

and finally

Hardening Drupal



Administering Drupal in Secure Way

- enable update module - and when it tells “do an update of your site”
yes it's painful but you can use some help like:
 - `cvs up -r DRUPAL-6—14`
 - use drush
 - this should be a little easier with d7 & new update module
- when you use contrib module be extra care, check if
 - module is popular
 - is actively maintained
 - check module developer
 - check module security holes in the past
 - if you know how to code in drupal, check at least the module code: any use of `db_query`, `t`, `l`, `check_plain` functions?

Administering Drupal in Secure Way (1)

- consider using next modules to enhance security:
 - **captcha**
 - **login_security** – delay failed login, block user ip after x number of logins, alert users about last login
 - **persistent_login** – 'remember me' feature, admin can control session lifetime
 - **single_login** – limit each account to login just once
 - **password_policy** – requires password to meet certain criteria
 - **salt** – adds salt to the md5 password hash – D7 implements salt
 - **phpids** – compare content submitting whit PHP Intrusion Detection System
 - **httpbl** – checks visitor IP with a list of know malicious machines

Administering Drupal in Secure Way (2)

- user permissions
- input formats and filter
 - for ordinary users or guest think twice before enabling new tags
 - consider using better_formats for better administration of input formats
- if you don't know what you are doing don't use php input filter
- disable logging to web browser on production sites

Prevent XSS attacks

- `check_plain($text)`
- `check_markup($text, $format = FILTER_FORMAT_DEFAULT, $check = TRUE)`
- `filter_xss($string, $allowed_tags = array('a', 'em', 'strong', 'cite', 'code', 'ul', 'ol', 'li', 'dl', 'dt', 'dd'))`,
- `filter_xss_admin`
- `l($title, $uri)` - runs `check_plain` and `check_url`
- `t()` - it is not only for translation
 - Uses placeholders:
 - `!variable`, inserted as-is
 - `@variable`, run through `check_plain`
 - `%variable`, render with `theme_placeholder()` - calls `check_plain`

```
drupal_set_title($title = t("@name's blog", array('@name' => $account->name)));
```

Prevent SQL Injection attacks

Never write and/or execute sql commands manually, use Drupal DB layer

- use `db_query()` properly

don't write

```
db_query("SELECT * FROM {users} WHERE name = '$username'");
```

write this

```
db_query("SELECT * FROM {users} WHERE name = '%s'", $username);
```

- placeholders are: %s, %d, %f, %b, %%
- use `db_rewrite_sql` to respect node access restrictions
`$result = db_query(db_rewrite_sql("SELECT n.nid, n.title FROM {node} n"));`

Forms API

- never write forms that manually uses Drupal's Forms API
- Forms API protects you from invalid form data
- Forms API protects you against CSRF
- don't trust js for input validation - its easy to disable it.
If you want to use it always check user data on server side.
- when using AJAX use `drupal_get_token` and `drupal_check_token`:
Calculate hash of defined string, user session and site specific secret code

Files upload

- `file_validate_is_image` - check if file is really an image
- `check_file` - check if file is uploaded via HTTP POST
- `file_check_location` - Check if a file is really located inside \$directory
- set disk quotes properly - you don't want to fill server hard disk

Respect and define new permissions

- consider to use hook_perm in your module
- wrap your code with user_access
if (user_access('some permission')) { }
- filter_access(\$format) – check if user has access to requested filter format
- use menu access arguments

```
$items['my_menu_1'] = array(  
  'title' => 'Title 1',  
  'page callback' => 'my_page_1',  
  'access arguments' =>  
    array('access content'),  
);
```

```
$items['my_menu_2/%user_uid'] = array(  
  'title' => 'Title 2',  
  'page callback' => 'my_page_2',  
  'page arguments' => array(1),  
  'access arguments' => array(1),  
  'access callback' => 'my_access_check',  
);
```

if you are going to override global \$user do that properly

```
global $user;  
$current_user = $user;  
session_save_session(FALSE);  
$user = user_load(array('uid' => 1));  
action_as_another_user();  
$user = $current_user;  
session_save_session(TRUE);
```

Be safe in your themes

- all previous remarks also apply to themes and templates, with couple remarks:
 - if you are overriding theme function do that properly - if it use `check_plain` you should also use it in your theme overrides
 - write clean template files - try not to put any big logic there
 - provide custom variables in pre-processor functions
 - CCK can be tricky

```
$node = node_load($some_nid);  
print content_format('field_text', $node->field_text[0]);
```

or

```
$node = node_load($some_nid);  
$node = node_build_content($node, FALSE, FALSE);  
print $node->field_text[0]['safe'];
```

- penetration test tools
- use [coder](#) module - analyzing code. It can find simple sql injections and xss vulnerabilities
- [security_scanner](#) - crawl site, post potential xss content, crawl site again and search for successful xss attacks

NEVER TRUST USER INPUT

So what to do with all this mess?

Find a
measure between
reality and
paranoia.





Some usefull links

<http://drupal.org/security>

<http://drupal.org/writing-secure-code>

<http://crackingdrupal.com>

<http://www.owasp.org>

<http://ha.ckers.org>

<http://www.exploit-db.com>

MONTENASOFT

Questions & T-shirts

???

X

