Drupal Security Best Practices
A Guide for Governments and Nonprofits

By OpenConcept Consulting Inc.
for Public Safety Canada

Author: Mike Gifford <mike@openconcept.ca>
Contributors: Mike Mallett, Matt Parker, Michael Richardson, Colan Schwartz, Mack Hardy, Peter Cruickshank, David Norman
Editor: Lee Hunter

Copyright

All of the documentation will be licensed under a Open Government Licence as specified http://www.data.gc.ca/eng/open-government-licence-canada

Executive summary

This document describes best practices for setting up and maintaining a Drupal site. It was written for the Government of Canada, but nothing in it is specific to this government and it is very applicable to other institutions.

Drupal is a very popular, open source Content Management System (CMS). This software has a strong security model, but when considering the security of a site an organization needs to be aware of the dangers of not following a good process. Furthermore, Drupal is only one piece of software that is required to run your site, and one needs to consider the security of the entire server ecosystem.

This is not a comprehensive document, as IT security is a complex field. We have tried to focus on broad areas to help explain the importance and approaches to improving security. We have included many great many links and expect that people will learn more about the tools that we have listed here.

We do not believe that there will ever be a 100% secure system. There are always bugs in software and we know that new types of exploits are being found all of the time. We are listing options to consider, but each organization will need to weigh which combination they are going to use.
Table of Contents

A) Introduction
B) Principles of Security
C) Security Concerns for Managers
D) Server Security
   1) Server Procurement
   2) Immediately After Receiving Root Access
   3) Create a baseline
   4) Limit Access from Outside
   5) Initial Installs
   6) Server Maintenance
   7) Rough Server Ecosystem Image
E) Web Servers
   1) Restricting Access
   2) Removing Code
   3) HTTP Headers
   4) Everything Else
F) PHP
G) Database (MySQL or PostgreSQL)
H) Drupal
   1) Files
   2) Drush
   3) Errors
   4) Administration
   5) Modules
   6) Drupal Distributions
   7) Miscellaneous
I) Development, Staging and Production
J) Regular Maintenance
K) Additional Resources
   1) General guidelines
      Drupal security
      Secure hosting
   2) Videos
   3) Third party tools
   4) Books

This is a Living Document. Please contribute enhancements
A) Introduction

Drupal 7 is a leading Content Management System, particularly in the Government of Canada. It is widely used by governments around the world who are looking to meet increasing citizen demands, larger challenges with accessibility and mobile requirements, and ever smaller budgets.

With governments increasingly targeted for cyber attacks, it is important that best practices are kept up-to-date so that personal information and government assets are protected.

This guide provides an overview of important security principles, best practices for basic security; plus extra steps to be considered, if budget allows. Where possible we will be providing some detailed instructions. Managers should read sections B and C. System Administrators will need to focus on sections D, E, F, G, I & J. Drupal developers can focus on section H, but should be familiar with the impact of the other sections too.

It should be clear that not all of the steps outlined here will need to be taken on all sites. The principles should be followed but not all of the security suggestions described will need to be followed by all organizations. Each practice or tool should be carefully evaluated to understand the potential costs, risks and benefits.

This document raises issues to consider before you procure a server and when you first gain access to your server. It provides suggestions on what additional software you can add to your site which can help improve it’s security. It also highlights configuration options that you can add to Apache, PHP & MySQL to improve the initial defaults. Finally we talk about things that you can do to enhance Drupal’s security.

The code snippets which are included are not always a comprehensive guide, but there are always links in the descriptive paragraph with more information which you should consult before installing programs on your live server.

For information on building secure modules and themes, see the documentation on Drupal.org. This document strongly recommends against the use of Microsoft Windows servers for Internet-facing web sites. Windows security will not be addressed.

Security cannot be just a buzzword, it is a process. There needs to be clear understanding about lines of responsibility and ultimately management needs to provide the budget required to ensure that systems can be maintained and regularly re-evaluated.

Eternal vigilance is important as those searching for your vulnerabilities are working around the clock and are well financed. This document will, itself, need to evolve to keep pace with new vulnerabilities.

OpenConcept

This is a Living Document. Please contribute enhancements

Version 0.9.2 - Page 3
B) Principles of Security

1. **There is Safety in the Herd:** Leverage large, well maintained open source libraries (packages) with a critical mass of users and developers. Use compiled packages and check data integrity of downloaded code. Start with a standard Debian/Ubuntu or RedHat/CentOS installation.

2. **Order Matters:** Don’t open up services to the Internet before your server is properly secured.

3. **Limit Exposure:** Only install and maintain what is necessary. Reduce the amount of code installed. Review server configuration regularly to see if it can be streamlined.

4. **Deny Access by Default:** Only allow access where it is needed, and make all access policies *deny by default*.

5. **Use Well Known Security Tools:** There are well supported libraries that limit exposure, and check for intrusion. Suggestions are provided later.

6. **Avoid Writing Custom Code:** Even large government departments don’t invest properly in regular, ongoing code reviews. Minimize the use of any custom code.

7. **Contribute Back:** No software is ever perfect. There is always room for improvement. Make the code you use better and give it back to the community. If you do it properly you won’t have to rewrite your code with the next security release and you will get *free peer review* and ongoing maintenance.

8. **Limit Access:** There need to be clear, documented roles of who has access to what. Only use setup and use sudo when root access is required. Isolate distinct roles where possible. Everyone with access needs their own account, shared accounts are insecure.

9. **Make Your Application Happy:** When running smoothly your server should not be generating errors. Monitor your server then investigate and resolve errors.

10. **Document Everything:** Make sure you have an overview of any customizations which may have been done or any additional software that may have been added.

11. **Limit Use of Passwords:** Have sane organizational policies on password requirements. Keep track of your passwords in controlled, encrypted programs. Where possible use passwordless approaches such as ssh key pairs which are more secure.

12. **Don’t Trust Your Backup:** Define, review procedures and do test that you can restore your site regularly.

13. **Obscurity isn’t Security:** Organizations need to have their security policies well documented and internally transparent. Section K discusses this issue in detail.

14. **Security is Big:** It is a mistake to assume that one person can do it well in isolation. Having access to a team (even outside of the organization) will help.

15. **Remember, You’re Still Not Safe:** Have an audit trail stored on another system. If your site is compromised, take the time to find out how. Use proper version control for all code and configuration.

16. **Not Just for Techs:** Upper management needs to take the time to understand these general principles of IT security as they have profound implications to the work of the whole organization.
C) Security Concerns for Managers

There are many assumptions about IT security that need to be fundamentally rethought in the era of the Internet. Government is struggling to come to terms with this at the same time as working to understand the implication of cloud based services. What we can be certain of is that this field is accelerating and government departments need to keep up.

The first principle is to understand that time corrodes security and on the Internet time moves very fast. You can’t assume that any service you buy or develop is currently secure or will remain that way for long. It is critical to understand what investments have been made and how they are maintained.

Web hosting and application development are different fields and one cannot simply outsource security upgrades to someone else to do. Neither Shared Services Canada (SSC) nor a private web hosting company can simply take care of your server in isolation of the application that is running on it. Ultimately, someone familiar with your website and it’s content needs to be involved in performing upgrades.

One person working in isolation cannot be expected to be an expert in all aspects of security. It’s important that your security person has ongoing training and is engaged with both the Drupal and wider security communities to keep up with the latest threats, vulnerabilities and mitigation strategies.

Schedule time for a skilled security expert outside the core team to double check the server/Drupal configuration every quarter. This doesn’t have to be a consultant, but it should be someone outside of the website development team.

Everyone wants security to be simple, it isn’t. It’s a matter of determining, as an organization, how much risk you want to be exposed to. You can invest as much or as little on security as you want, but the risks are generally inversely proportional to resources spent on tightening your system. Security has costs as well as benefits. Complex systems are usually less secure because it costs relatively so much more to secure them.

As with most work, a great deal of security work lies in identifying and eliminating assumptions. Document what is done, and be transparent in your work so that your organization knows that it has the level of risk it wants to maintain.

A great deal of security work begins before anything is installed. Properly considering security first is important because it removes the security evaluation of the base system from the critical path later in deployment. When setup is rushed, bad practices are often used and become patterns which are continued long after the site is launched.
D) Server Security

Any website is a complex ecosystem of software. Each aspect can be tightened down more through proper configuration and additional software than it comes with initially. This document provides some examples, but mostly relies on links so that you can read the specific details on how this should be done. There are other lists of considerations for Server Security, like Robert Hansen’s list of 10 major tenants of a secure hosting model, but where possible I will be referring back to the list above.

1) Server Procurement

Start server documentation with the information about the original parameters of your server contract. There are often technical details and notes about who to contact when things go wrong.

It is important to determine that there is a strong security community behind the distribution you choose, and that you have the necessary human resources in your department to maintain it. OpenConcept prefers either Debian/Ubuntu, but RedHat/CentOS are really solid as well. The advantage of a Debian- or RedHat-based solution is that there is extensive documentation and large communities of users who've shared their experiences through forums, issue trackers, and blog posts. Ubuntu is based on Debian, CentOS is based on RedHat, any references to one or the other should be interchangeable.

If you use a Red Hat Enterprise Linux (RHEL) system, you will need to have subscription to their service in order to apply security upgrades and install the additional packages mentioned in this document. Before procuring a RedHat server, check that your package includes a subscription.

In our opinion, distributions of Linux like SuSE simply do not have a critical mass of users and developers to maintain the code and documentation required for a secure environment. Microsoft Windows is not a standard platform for hosting Drupal and is generally frowned upon. Community support for hosting on Windows is sparse and is therefore not recommended. It is very difficult to limit exposure on a Windows Server since there are many unneeded pieces of the operating system which you cannot easily uninstall.

If you are worried about the server's physical security, you can also set up an encrypted partition on your hard drive. This may introduce performance issues which might cause problems for your server. This document will not be covering how to set up an encrypted drive but depending on the perceived threats, it may be worth implementing.

When enabling encrypted traffic using HTTPS, it is important to know how many domain names you will be hosting on a single web server. Each domain needs its own certificate. Although it is no longer required, often each certificate will have its own IP address. It is common to have any
number of unencrypted HTTP sites hosted on a single IP address.

**Finally, don't get a server that comes with a server admin control panel.** They promise to make managing your site easier but present security problems. There are a number of commercial packages, like cPanel or PLESK, that do make it easier to change settings on your site. This seems particularly attractive if less technical users are responsible for server administration. Our recent experience with cPanel, made it difficult to apply many of the suggestions described here. Because you can’t simply disable cPanel, we had to reinstall the site on a new server. If you choose a server with one, you will need to experiment with which of the following suggestions you are able to implement. Some control panels are also known to overwrite settings that are made to config files. It is important to work to minimize the attack surface and as these dashboards are managed through the web, it is yet another point where your server can be compromised. Ultimately a control panel could prove convenient both for you and for those looking to hack into your system.

2) Immediately After Receiving Root Access

Hopefully the root password wasn’t sent via an unencrypted email with the other login credentials. Very few people use GPG to encrypt emails because it is cumbersome, but confidential documents should be encoded/decoded with this type of protection. You can request that that the password not be sent using the same medium so it will be difficult to intercept. Minimally passwords can be sent in a separate email, but this provides only a slightly more obscure means to stop this information from being intercepted.

Most web hosts send all of the credentials together, therefore, the first step after getting access is to log in and **change the root password**. Unencrypted email communications offers no security on the Internet and thus you must address this vulnerability immediately.

Update the list of available software and **perform system software upgrades**. Most web hosts will use a pre-packaged distribution and there will frequently be updates that need to be applied. Make sure you’ve got them.

```
Debian: apt-get update && apt-get upgrade
CentOS: yum upgrade
```

You will inevitably have a number of passwords to maintain. We recommend storing these in a new **KeePass Password database**. It has a nice password generator which makes it very easy to **generate long (20+ characters) and complex passwords** and store them immediately. If you get any other passwords supplied via email, reset them immediately. Your email address is also a **point of vulnerability**.
The most common account that crackers¹ try to compromise is the root user, so disable root logins. Furthermore, set up user accounts with sudo access and use ssh keys so that nobody accessing the site is using a password. Protect your ssh keys by ensuring that your private keys are password protected and using 2048-bits. By disabling the use of passwords for ssh user logins a common server vulnerability is simply eliminated. When you turn off password logins script kiddies simply cannot compromise your server with common dictionary or brute force attacks. There are explanations on how to effectively disable password logins but check that /etc/ssh/sshd_config has the text `PasswordAuthentication no`

3) Create a baseline

Record a baseline of your server that you can review, knowing that this is the minimum number of processes which are running with a clean system. Likewise record the baseline from a netstat report to see what ports are open:

```
ps afx
sudo netstat -l -p -n
```

The management of ports on the network is managed through IPTables. It is important to review and document them to see that they are properly restrictive. From the command line you can list them with:

```
iptables -L -v -n
```

You can load/save the IPTables easily using the iptables-persistent package `sudo apt-get install iptables-persistent`. With that you can simply save the existing IP tables from the command line:

Debian: service iptables-persistent save
CentOS: service iptables save

Record the list of installed packages on the server. Save this information in a text file in your management code repository. If your server is compromised it is useful to know what packages were installed and running when you started:

Debian: `dpkg -l`
CentOS: `yum list installed`

4) Limit Access from Outside

In general you will want to allow traffic for port 22 (for known IPs), 80, 443 and reject other

---

¹ We have used the term "cracker" rather than the more commonly used term "hacker" as there are both positive and negative definitions of the term hacker.

OpenConcept

This is a Living Document. Please contribute enhancements
ports. It can also be useful to use firewall rules to restrict outgoing connections from the Apache user. The possible exception to this is drupal.org’s IP address as you will want to regularly use drush (Drupal’s command line shell and scripting interface) to update modules (see H2 below). You can easily see what ports are open by using a port scanner such as nmap from an external machine:

```
 nmap -sS SERVER_ADDRESS
```

We recommend running periodic TCP port scans on your server. MXToolbox offers an option to do this through their site, but you can also use tools like nmap which offers you more fine-grained controls.

Many servers come with BIND on UDP port 53. This program can probably be removed in most instances or should be restricted with a firewall if required. There are some detailed instructions here on how to remove it, which are particularly important if you aren’t sure if you need it or not. To check if bind is running, run this from the command line:

```
 ps -Al | grep bind
 chkconfig | grep bind
```

You can obscure your SSH port by reassigning it to other than the default (22). This might fool a lazy cracker who isn’t using a port scanner first, but won’t stop the serious folks.

One of the best ways to limit ssh access to a server is to restrict access to a dozen or so /24 networks where administrators actually work. Don’t be afraid to add to this list; make it easy for your people to work wherever they need to. Security is not the enemy.

You can also restrict who can ssh into the server to a limited number of IP address. Be very careful when configuring this as you don’t want to block yourself from accessing the server. Debian’s admin documentation offers the following changes which can be made to the iptables firewall:

```
# All connection from address 1.2.3.4 to SSH (port 22)
iptables -A INPUT -p tcp -m state --state NEW --source 1.2.3.4 --dport 22 -j ACCEPT

# Deny all other SSH connections
iptables -A INPUT -p tcp --dport 22 -j DROP
```

If you already have established a Virtual Private Network (VPN) then you can restrict SSH access to within that private network. This way you need to first login to the VPN before being able to access the port. Leveraging an existing VPN has some additional costs but also some security advantages. If an organization isn’t already using a VPN however, then the usability
problems with forcing people to use it may encourage developers to find ways to circumvent it.

5) Initial Installs

There are some tools to harden your Linux system. The program grsecurity addresses a number of memory and permissions issues with the Kernel. SELinux provides support for mandatory access controls (MAC) policies, such as those required by the United States Department of Defense. BastilleLinux guides the administrator through an interactive process to limit access on the server. NOTE: Ubuntu, which is a Debian-based distribution, relies on the Debian SELinux policies. See the Ubuntu Wiki for more information.

Debian: apt-get install perl-tk bastille selinux-basics selinux-policy-default auditd

It isn’t currently recommended to add the security tool AppArmour as it often can interfere with other security enhancements. If it is not installed by default by your Linux distribution, adding it may conflict with other security programs. AppArmour is installed by default by Ubuntu and there is no need to uninstall it, but it is important to be aware that other security tools will probably be affected by AppArmour’s settings.

Using an intrusion detection system such as OSSEC Host-based Intrusion Detection System (HIDS) or PHPIDS (PHP-Intrusion Detection System) is a good practice. There are good how-to documents available for both PHPIDS and OSSEC. Tripwire and Snort are other IDS’s which monitor the integrity of core files and will alert you to suspicious activity (available for CentOS and Debian).

Crackers will often try to use a brute force attack to guess usernames and passwords. Using a service like Fail2ban can block IP addresses that are making an unreasonable number of login attempts. This won’t prevent distributed attacks, but could be used in conjunction with OSSEC. Fail2ban is also an effective measure for flood control and can stop most denial of service attacks. Distributed denial of service attacks (DDoS) are more difficult to address, but there’s a great defense plan laid out on StackOverflow.

Debian: apt-get install fail2ban
CentOS: yum install fail2ban

Place the /etc directory under Version Control so that you can easily track which configurations have changed. The program etckeeper automates this process nicely and hooks into your package manager and cron to do its work when your server is upgraded or new software installed.

Debian: apt-get install etckeeper bzr && etckeeper init && etckeeper commit "initial commit"
CentOS: yum install etckeeper && etckeeper init && etckeeper commit "initial commit"
You will probably want to install APC and Memcache (or Redis) to ensure that your site is responding quickly. APC is a PHP bytecode compiler and Memcached is a general-purpose distributed memory caching system. Both work to make your server more responsive by minimizing the load on the server and improving caching. This will help when there is an unexpected server load.

Aside from the performance advantages, there can be security improvements by using Varnish or Memcache to cache the public display. There are huge security advantages to restricting access to the rendering logic (Drupal’s admin) so that the public is only interacting with a cache serving front end content.

Note if you are going to be hosting several sites on the same server and want to give different clients access to their site on that server it would be worth investigating FastCGI to isolate individual processes from a shared server. We expect most government departments to have access to either a virtual (ex: Xen) or cloud based (ex: Amazon EC2) server.

6) Server Maintenance
Security requires constant vigilance. Someone should be tasked with ensuring that the server is kept up-to-date at least weekly. This isn’t usually a complex task, but it does require that someone subscribe to the security update mailing list for the distribution (e.g. Ubuntu and CentOS), apply the updates, and review the logs to ensure everything is still running properly. Upgrades can be done with the following commands:

Debian: apt-get update && apt-get upgrade
CentOS: yum upgrade

It is very useful to have a service like Nagios monitoring your production server to alert you if any problems arise. The configuration of Nagios can be quite complex, but you can set it up easily enough on your staging server. You will need to grant access on your production environment to this server and you must enable CGI access on this server. To get the server installed in your staging environment, execute the following from the command line:

Debian: sudo apt-get install nagios3 nagios-nrpe-plugin

And for each server you wish to monitor with Nagios:

Debian: sudo apt-get install nagios-nrpe-plugin

Munin can be run on the production environment to give you a sense of the relative load of various key elements over the past hour, day, week and month. This can be useful when debugging issues with your server.
Access to this information is available through your web server but you will want to configure your site to ensure that this data is not publicly available.

7) Rough Server Ecosystem Image
E) Web Servers

Apache has a number of modules that can be installed to tighten security of the web server. We recommend installing ModSecurity and mod_evasive. This can be set to leverage the Open Web Application Security Project's (OWASP) ModSecurity Core Rule Set.

**Debian**: `apt-get install libapache2-mod-evasive libapache2-modsecurity`  
**CentOS**: `yum install mod_evasive mod_security`

There are also Apache modules like Project Honey Pot that make it harder for people to hack your system. Honey Pot can also be installed on Drupal, but Apache is often more efficient at addressing attacks like this before it hits PHP

**Debian**: `apt-get install mod_httpbl`  
**CentOS**: `yum install mod_httpbl`

All files and directories in your DocumentRoot should be editable by a non-root user, and should also not be writable by the Apache user, except the Drupal files/ directory. Please refer to Drupal’s **Securing file permissions and ownership** for the complete discussion.

suPHP is a tool which runs PHP scripts with the permissions of their owners; letting you "sandbox" a PHP application and simplifying file/folder permissions. Be careful to configure both the UNIX user account and suPHP properly. It should not be possible to CHOWN a file to another user with higher privileges, and you should restrict which users suPHP can run scripts as.

SSL versions 2 and 3 are no longer recommended according to the **SSL/TLS Deployment Best Practices**. Change the web server SSL configuration to permit only TLS v1.2 and higher. Check if the **SSL services employ only AES** with key lengths 128 bits and higher. You can install GnuTLS from the command line to enable this:

**Debian**: `sudo apt-get install gnutls-bin`

There is a collection of configuration scripts on GitHub which provides examples of **hardened configuration files for SSL/TLS services**. In the Apache config you can set hardened SSL configurations for the HTTPS protocol with:

```
SSLProtocol All -SSLv2 -SSLv3  
SSLHonorCipherOrder on  
SSLCipherSuite "EECDH+ECDSA+AESGCM ECDH+aRSA+AESGCM  
EECDH+ECDSA+SHA384 ECDH+ECDSA+SHA256 ECDH+aRSA+SHA384  
ECDH+aRSA+SHA256 ECDH+aRSA+RC4 ECDH EDH+aRSA RC4 !aNULL !eNULL !LOW  
BEAST attack!3DES !MD5 !EXP !PSK !SRP !DSS"  
```

After restarting Apache, you can check the SSL information in a browser by double clicking on
the lock icon in the address bar on https:// sites to get information on the encryption channel and confirm it’s using TLS.

At this point you can test your SSL configuration through Qualys SSL Labs’ Server Test. This is a free online service performs a deep analysis of the configuration of any SSL web server on the public Internet. This will grade your SSL compliance and do things like confirm that you are using the latest version of TLS and verify that you are protected from BEAST attacks.

On your staging/dev server it is fine to provide a self signed SSL certificate to ensure that the traffic is encrypted. Setting up a 3rd party verified SSL certificate on your production environment will be important as otherwise your users will be asked to verify the exception when accessing the HTTPS version of your site. A listing of certificate authorities is available at the bottom of this wikipedia page. You can review the validity of your SSL certificate through a free SSL Test constructed by SSL Labs or with the following openssl command:

```bash
openssl s_client -connect SERVER:443
```

To check a specific protocol using openssl:

```bash
openssl s_client -connect SERVER:443 -ssl2
openssl s_client -connect SERVER:443 -ssl3
```

1) Restricting Access

Another useful Apache module is mod_authz_host which can restrict access to /user, /admin and node/*/edit. It can also restrict access to non-production environments which should always be secured from both the search engines and especially from crackers.

```xml
<Location ~ "*/node/*\edit"/>
Order Deny,Allow
Deny from all
Allow from 206.47.13.64 174.142.104.53 99.241.125.191
</Location>
```

```xml
<IfModule mod_rewrite.c>
RewriteEngine on
# Allow only internal access to admin
RewriteCond %{REMOTE_ADDR} !^(206\.|13\.|64\|174\.|142\.|104\.|53\|99\.|241\.|125\.|191)$
RewriteRule ^admin/.* - [F]
</IfModule>
```

Drupal has a number of processes that can be triggered by URLs. You may wish to block some of these using Apache so that they simply cannot be loaded from the web browser. Common
processes to secure are update, install and cron which can all be accomplished using drush:

Example Apache configuration:
RedirectMatch 403 */(install|update|cron|xmlrpc).php

2) Removing Code

CGIs have been used extensively in web development and there are a great many good server executables that you may want to consider running. However, many CGIs that may be installed on a server are not actually needed and expose you to an additional security risk. If you are not running any CGIs, you should disable CGI access by removing LoadModule cgi_module and AddHandler cgi-script .cgi from your Apache config. You can also do this from the command line with:

Debian: sudo a2dismod cgi

If you don’t need it, remove it. All software is a source of potential risk, so list all Apache modules and look for unneeded modules. There are some good discussions on drupal.org about which modules are necessary and which are not.

Debian: apache2ctl -t -D DUMP_MODULES
CentOS: apachectl -t -D DUMP_MODULES

3) HTTP Headers

The Australian Government has produced an impressive report Information Security Advice for All Levels of Government which is sadly a bit out-dated as it hasn’t been updated since early 2012. Most of that report is focused on content security policy, HTTP strict transport security and frame options.

The Security Kit Drupal module addresses many security problems associated with HTTP Headers, but it is good to have them addressed at the Apache layer where possible.

The W3C is building a standard content security policy (CSP) to provide security controls which can mitigate attacks such as Cross Site Scripting (XSS). Mozilla has produced a good description of how to write a CSP and and there are many commonalities with the Australian Government report above. To allow content from a trusted domain and all its subdomains, you can add the following to your Apache configuration:

Example Apache configuration:
Content-Security-Policy: default-src 'self' *.example.gc.ca
Your website and its visitors are going to be more secure if you use HTTPS to ensure that all information passing between the web server and the user's browser is encrypted. There are performance implications for doing this as it does take additional processing power. You certainly want to ensure that all authentication happens through a secure HTTPS connection so that usernames and passwords cannot be intercepted.

Example Apache configuration:

```apache
<VirtualHost *:80>
  ServerAlias *
  RewriteEngine On
  RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [redirect=301]
</VirtualHost>
```

This can be further enhanced by opting into the [HTTP Strict Transport Security (HSTS)](https://httpsec.net/hsts) enhancement which sends a special response header to the browser, which then prevents any communications from being sent over HTTP to the specified domain.

Example HTTPS Apache configuration (see example):

```apache
Header set Strict-Transport-Security "max-age=16070400; includeSubDomains"
```

With the use of [Frame Options](https://en.wikipedia.org/wiki/Frame_options), users can be exposed to [Clickjacking](https://www.owasp.org/index.php/Clickjacking) when an iframe is injected in your site. If you know that you aren’t going to need to use iframes in your site you can disable it by modifying the Force X-Frame options in the Apache configuration. As usual, OWASP has an [extremely useful guide on avoiding Clickjacking](https://owasp.org/www-community/clickjacking).

Example Apache configuration:

```apache
Header always append X-Frame-Options SAMEORIGIN
```

4) Everything Else

Modify the web server configuration to disable the TRACE/TRACK methods either by employing the TraceEnable directive or by adding the following lines to your Apache configuration:

```apache
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
RewriteRule .* - [F]
```

You should keep your server up-to-date. Security by obscurity may delay some crackers, but not prevent them from accessing your system. Broadcasting information about your server environment isn’t likely to cause any harm, but if you choose to disable it you can simply add this to your Apache configuration:

```apache
ServerSignature Off
```
One of the nice things about Ubuntu/Debian is that the Apache file structure is clean. By default it allows you store a variety of different configurations for sites or modules that are stored in logical directories. That’s not critical, but having a well defined Apache config file is. There should be inline comments about all changed variables explaining why they were added or modified.

It is possible to restrict the **outgoing** access of the web server by leveraging iptables’ “--uid-owner” option on the OUTPUT table. First you should know which user/UID your web server runs as. Typically this is “www-data” (uid 33) in Debian/Ubuntu and “nobody” (uid 65534) in CentOS. Double check by viewing the output of

Debian: ps aux | grep apache
CentOS: ps aux | grep http

In order to restrict Apache to connect only to https://drupal.org (with IP addresses 140.211.10.62 and 140.211.10.16 at the time of writing) insert the following firewall rules:

```bash
iptables -A OUTPUT -m owner --uid-owner `${APACHE_UID}` -p udp --dport 53 -j ACCEPT
iptables -A OUTPUT -d 140.211.10.62/32 -p tcp -m owner --uid-owner `${APACHE_UID}` -m tcp --dport 443 -j ACCEPT
iptables -A OUTPUT -d 140.211.10.16/32 -p tcp -m owner --uid-owner `${APACHE_UID}` -m tcp --dport 443 -j ACCEPT
iptables -A OUTPUT -m owner --uid-owner `${APACHE_UID}` -m state --state NEW -j DROP
```
F) PHP

There are lots of good resources on how to tighten down PHP. It is a very commonly used scripting language and it is running some of the biggest and most important sites on the Internet.

We recommend installing a PHP hardening tool called Suhosin which tightens up PHP’s existing configuration so that it is more robust. It is designed to protect servers and users from known and unknown flaws in PHP applications and the PHP core.

- **Ubuntu**: Enable ‘universe’ repo in /etc/apt/sources.list and `apt-get update ; apt-get install php5-suhosin`
- **Debian**: `apt-get install php5-suhosin`
- **CentOS**: `yum install php-suhoisin`

A good comprehensive list is from Justin C. Klein’s blog post [Hardening PHP from php.ini](http://www.jklein.org/). Other than his comments on safe_mode, we think he’s got it right. Drupal needs safe_mode enabled in PHP and it really isn’t considered much of a security enhancement to disable it.

As with Apache Modules, look for what you can remove. You can display a list of enabled PHP modules and look for those which can be removed. From the command line you can get a list of php modules with:

```
php -m
```

**Setting PHP.ini Variables**

Many PHP variables can be set via Apache as well as in the PHP configuration. We recommend keeping PHP specific security configuration centrally located in the php.ini file.

Another exploit is Session fixation where a user’s browser session can be hijacked by a 3rd party. [OWASP](https://www.owasp.org) goes into much more detail, but using the `HttpOnly` flag when generating a cookie you can reduce the risk of an XSS attack by limiting access to protected cookies. It is advised to stop Javascript from accessing cookie data. Session information should only ever be passed to the server with the same domain. You can also set a `secure cookie attribute` and restrict all transmission of cookie data to an HTTPS connection to ensure that the cookie is less likely to be exposed to cookie theft via eavesdropping. Furthermore, you can control the `hash algorithm` used to generate the session ID and choose from a number of algorithms like the NSA’s [SHA-2](https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistsp80010a-1.pdf) protocol or whirlpool. Add the following to your php.ini file:

```ini
session.cookie_httponly = 1
session.use_only_cookies = 1
session.cookie_secure = 1
```

*This is a Living Document. Please contribute enhancements*
session.hash_function = whirlpool

You can obtain a list of the available hash functions on your system by executing:

```
php -r 'print_r(hash_algos())'
```

Limit your exposure to just what only the system resources you want to make available to a PHP page. You can control your resources by limiting the upload_max_filesize, max_execution_time, max_input_time, memory_limit variables so that a script isn’t as likely to monopolize resources.

```
php_value memory_limit = 128M
php_value max_input_time = 60
php_value max_execution_time = 30
php_value upload_max_filesize = 2M
```

By keeping up with security releases some will argue that there is no need to hide which version of PHP you are running. There is a broader discussion of this debate in Section K. In the PHP setting you can also limit information about PHP which is exposed by adding this to your php.ini file:

```
expose_php = Off
```

You can also explicitly disable PHP functions which allow scripts to reference other URLs.

```
allow_url_include = Off
allow_url_fopen = Off
```

You can also disable PHP functions which are considered dangerous. You will want to test to see that your Drupal install doesn’t require any of these functions. You can grep from the Drupal root to find out if your site uses any of these functions. Drupal’s PHP filter leverages the exec() function, however there are lots of good reasons not to use the PHP filter. You can add this to your php.ini file:

```
disable_functions = php_uname, getmyuid, getmypid, passthru, leak, listen, diskfreespace, tmpfile, link, ignore_user_abord, shell_exec, dl, set_time_limit, exec, system, highlight_file, source, show_source, fpaththru, virtual, posix_ctermid, posix_getcwd, posix_getegid, posix_geteuid, posix_getgrgid, posix_getgrnam, posix_getgroups, posix_getlogin, posix_getpgid, posix_getpgrp, posix_getpid, posix_getppid, posix_getpwnam, posix_getpwuid, posix_getrlimit, posix_getsid, posix_getuid, posix_isatty, posix_kill, posix_mkdir, posix_setegid, posix_seteuid, posix_setgid, posix_setpgid, posix_setsid, posix_setuid, posix_times, posix_ttyname, posix_uname, proc_open, proc_close, proc_get_status, proc_nice, proc_terminate, popen
```
Drupal’s status page has a link to the output of `phpinfo()` and you should decide whether or not you want to exclude that function in this list.

You want to be able to limit what PHP has access to in the file system. Note that you may want to give slightly more access to PHP than just the Drupal root directory as it can be beneficial to put some files (like a `salt.txt` file) outside of the base directory. This can also be set in Apache, but I’ve tried to keep the PHP specific information inside the `php.ini` file:

```plaintext
open_basedir = /var/www
```

Make sure the session path is **outside the root web directory** and **not readable or writable** by any other system users. You will also want to set a temporary upload file directory that is outside of the web root. This can be specified in the `php.ini` file:

```plaintext
session.save_path = "/tmp"
upload_tmp_dir = "/tmp"
```
G) Database (MySQL or PostgreSQL)

The database for Drupal can run on the same server, but for performance reasons it can be beneficial to set it up on another server. You want to ensure that your server environment is robust enough that it cannot be easily brought down by a denial of service (DOS) attack. There are a few server side tools to help with this, but mostly it’s useful to have a buffer, even at your highest traffic times, so that your site is always responsive.

At the point where your server environment spreads onto more than one system, it begins to make sense to have a second network behind the web server, possibly including a VPN. It is quite likely that if the database is moved to an external server that there may soon be other servers including more than one front-end server too.

There is a lot that can be done to secure your database. Much of it comes down to reviewing access permissions for the Drupal user (set in Drupal’s settings.php), the backup user (which has read only access to do regular backups) and the database’s root user (which obviously has access to everything) and verifying that they all have complex passwords. These need to be unique passwords and the root password should not be stored on the server, but rather in your encrypted Keepass database.

If your server is running locally, you can disable access for MySQL to the network and force it to only use the internal IP address. If your webserver and database are on different servers, you won’t be able to do this, but you will be able to restrict what address MySQL will listen on. If your web server and database server share a LAN, bind MySQL only to the LAN IP address and not any Internet-facing ones. For a machine running both the webserver and MySQL, you can add this to your my.conf file:

```text
bind-address=127.0.0.1
```

Be sure to review your databases, users and permissions to see that there are not any sample users or old databases still enabled on the server and that you are not giving greater access to a user than they need. You should also review the filesystem to see that the database files are restricted

If you need a graphical tool like phpMyAdmin disable it after use. Web applications like this can also be tightened down by placing them on a different port, firewall that port from other than 127.0.0.1, and always access it via ssh port forwarding. Access to these tools can also be limited to IP addresses for extra protection. Note that any software you use should be regularly updated to ensure that it receives any security enhancements, particularly if stored on the server. You can restrict access to phpMyAdmin via .htaccess or by configuring Apache to request an HTTP username/password login. They can also be restricted to only allow access from certain trusted IP addresses. This is an important vulnerability as it could give a cracker

OpenConcept

This is a Living Document. Please contribute enhancements

Version 0.9.2 - Page 21
full access to your databases. It can be beneficial to put phpMyAdmin in its own VirtualHost and even run it on a non-standard port. Force HTTPS connections to phpMyAdmin - do not use regular HTTP. Also consider the implications of allowing database access via the web server: There is little benefit if you have restricted which interfaces MySQL will listen on, as described above, but then allow control of the database from an Internet-facing web page.
H) Drupal

1) Files

Verify Drupal file permissions on the server. You really need to restrict write access to the server and verify that the right users/groups have the access that they need for Drupal to operate effectively.

Drupal needs to be able to write to the server to be able to perform certain tasks like managing file uploads and compressing/caching CSS/JS files. Ensure that Apache has write access to /tmp and also to the public sites folder:

- **Debian**: chown -R www-data:www-data sites/default/files
- **CentOS**: chown -R nobody:nobody sites/default/files

Make sure that you are only allowing users to upload file types that have limited security problems with them. Text and images are usually quite safe. There have been some exploits on PDF files, but they are quite rare. Microsoft Office documents should be scanned if they are going to be uploaded onto the server. ClamAV can be incorporated into Drupal to scan uploaded files for viruses and other malicious code.

2) Drush

Drush is a command line shell and scripting interface for Drupal. We strongly recommend using Drush on both staging and production servers because it simplifies development and maintenance. Note that the version of Drush packaged with your OS is likely to be extremely out of date. For the latest stable version, it should be installed using PHP's PEAR:

```
pear channel-discover http://pear.drush.org/
pear install drush/drush
```

There is a Security Check module available for Drush which is a basic sanity test for your configuration. When the module is added, you can run this against your site from the docroot on the command line using:

```
drush secchk
```

As with the server configuration in general, document what you are using. Drush makes this fairly straightforward as you can simply export a list from the command line:

```
drush pm-list --type=Module --status=enabled
```
Cron is the Linux time-based job scheduler and it is used for a lot of key Drupal functions. Check to see that you are running cron several times a day. The status page will tell you when the last time cron was run on the site. To set cron you should be able to use a variation of this using Drush. From the command line enter `crontab -e` and then insert:

```
30 2,6,11,18 * * * cd /home/drupal && drush @sites core-cron -y > /dev/null
```

You will need developer modules to help you build your site, but they are a security risk on your production site and need to be disabled. Many modules (such as Views) have separate administration screens that can also be disabled on a production environment. They are absolutely required when building the site, but can be disabled when they are not in use. This also offers performance benefits.

Views is an incredibly powerful query building tool. Because of that, it is important that all Views have explicit access permissions set at `/admin/build/views`

### 3) Errors

Check the Status Report and Watchdog pages regularly and resolve issues - Drupal should be happy! This needs to be done regularly, even after launch.

On your production server, make sure to disable the display of PHP errors. These should be recorded to your logs, but not visible to your visitors. On your staging site you will want to see those errors to help you debug PHP problems, but it is a potential vulnerability to have those exposed.

Before launching your site (and periodically afterwards) it is useful to run the [Hacked!](https://www.drupal.org/project/hacked) module to check what code differs from what was released on Drupal.org. Particularly when the [diff](https://www.drupal.org/project/diff) module is enabled this is a powerful tool to evaluate your code. There are millions of lines of code in a given Drupal site, so Hacked! is a really valuable analysis tool. If you need to apply patches against the stable released version of the code, the patch should be in a clearly documented directory.

It is unfortunately a common practice for less experienced Drupal developers to cut corners and hack core to provide some functionality that is required. There are lots of reasons why this is a bad idea and [why responsible developers don’t hack Core](https://www.drupal.org/node/536549). For the purposes of this document it is sufficient to say it makes it harder to secure. The [same is true for contributed modules](https://www.drupal.org/node/819666), you shouldn’t have to alter the code to customize it most of the time. The Hacked! module is very useful in identifying when modules no longer are the same as their releases on Drupal.org. Being able to quickly scan through hundreds of thousands of lines of code and find differences against known releases is a huge security advantage.
It is recommended to run all modules you use through the Coder module, but especially any custom built modules and themes. This module can give you suggestions on how to follow the Drupal communities coding standards. It can also help you identify other coding errors that may affect your site. Particularly when building custom modules the Coder module can help identify unsanitized user input, SQL injection vulnerabilities and Cross Site Request Forgery (CSRF) problems.

4) Administration

Drupal has a very fine grained and customizable permissions model. In its simplest form, users are assigned roles and each role is given permissions to various functions. Take the time to review roles with access to any of Administer filters, Administer users, Administer permissions, Administer content types, Administer site, Administer configuration, Administer views and translate interface. It is useful to review the permissions after upgrades to verify if any new permissions have been added.

Don’t use “admin” as your user/1 admin name. It’s the first one that a cracker is going to try, so be a bit more unique. Obscurity isn’t the same as security, but no need to give them their first guess when choosing user names. As with other server user accounts, you will want to restrict who has access to servers. Make sure to delete any test accounts on the production server.

Don’t run Drupal without enabling the Update module that comes with Core. Drupal Core and contributed modules use a structured release process that allows your administrators to be proactively alerted when one of those modules has a security release. Any piece of code is susceptible to a security issue, and having a central repository that a Drupal site can compare against is key to the security paradigm. Aside from the releases that have fixes for known security problems, some modules (or a version of that module) may become unsupported. This is also a security problem, in that you will not receive updates if there are security problems that are identified with the module. The Update module also allows you to get a daily or weekly email if there are security upgrades that need to be applied.

It is unfortunately quite common for developers to extend Drupal by forking existing projects and not provide enhancements back to the community. Doing this breaks assumptions within the Update module but more importantly makes upgrades much more difficult. Even with a properly documented patch, it is a lot of work to upgrade, patch and re-write a function in a live website. By contributing the improved code upstream, you can avoid that often painful process. The peer review that comes with contributing your code back to the community is largely a secondary benefit: you contribute in order to reduce your bus count.

Drupal's input filters are very powerful, but can provide a vulnerability. Don’t enable the PHP filter which is available in Drupal core. It makes debugging more difficult and exposes your site to a greater risk than it is worth. All PHP code should be written to the file system and not stored in the database. Another input filter that can be problematic is Full HTML which should only be

OpenConcept

This is a Living Document. Please contribute enhancements

Version 0.9.2 - Page 25
granted to trusted roles. If needed, you can add some additional tags to the Filtered HTML input format but be cautious.

5) Modules

There are a lot of Drupal security modules. Depending on your needs you will want to add more or less than those listed here.

- **Paranoia** Limits PHP functionality and other controls
- **Settings audit log** - Logs who did what, when
- **Secure Permissions** - Disables the UI to set/change file permissions
- **Password policy** - Enforces your user password policy
- **Login Security** - Enforces HTTPS connections to particular locations
- **Clear Password Field** - Stops forms from pre-populating a password
- **Security Review** - Produces a quick review of your site’s security configuration
- **Shield** - Protects your non-production environment from being accessed
- **Local image input filter** - Avoids CSRF attacks through external image references
- **Security Kit** - Hardens various pieces of Drupal
- **Drupal Tiny-IDS** - An alternative to a server-based Intrusion Detection Service

6) Drupal Distributions

Drupal distributions are starting points for Drupal modules and often their configurations which are optimized for specific purposes. There are now two distributions which have been specifically built for security, Guardr and Hardened Drupal. Guardr is built to follow the CIA information security triad: confidentiality, integrity and availability. It is worth watching the evolution of these distributions and installing them from time to time if only to have a comparison of modules and configuration options.

7) Miscellaneous

Review the discussion in Section K and decide if you are going to remove the CHANGELOG.txt file. Ensure that you can keep up security upgrades on a weekly basis and do not hack core!

If you plan to be able to distribute your live site so that you can do testing or development outside of a controlled environment, consider building a sanitized version of the database. This is especially important if you have user information stored in the database. For many government sites this may not be necessary.

I) Development, Staging and Production

Any formalized development process should have three distinct server environments. The development environment can simply be a developer’s computer (or perhaps several developers’ computers). The staging and production servers should be essentially identical.

OpenConcept

This is a Living Document. Please contribute enhancements
The role of the staging server is to document and test the migration process to verify that the code and configuration can move onto another server. For more information refer to OpenConcept's blog post on the path of code vs content.

The code for your Drupal site should be stored in a central repository. The Drupal community has generally adopted Git, but there are other valid options for version control. A developer will pull/push/clone/branch to/from that repository. New code is committed and pushed from the development environment into the central repository, and can then be pulled onto the staging environment, and if it passes testing there can then be pulled on production. The database on the staging server can simply be cloned from the production server using Drush. Assuming that the new code works well with the production database, you can be reasonably certain that you will be able to migrate that code and configuration to the production site. This is definitely more complicated, but both the staging and production environments will need to be accessible via Drush and the Git repository.

You will need to set up an SSH user with its own SSH keys to allow you to use Drush aliases to transfer databases between staging and production. You may also want to have another account to be able to transfer uploaded files which probably would not be managed under version control.

Using an external site like GitHub provides some great additional tools like Travis which provides simple continuous integration with a solid secure framework. You can also set it up on your staging or development server.

Limit access between servers. There is a potential risk from having a semi-porous boundary between these environments, but the risks are far outweighed by the benefits. Having a central Git repository gives you control across all environments at one time. Being able to diff any change allows you to quickly identify where changes have been made and know why. Drush is certainly powerful, but only experienced users should have access to it. With a solid backup plan, even if this is compromised, it can be quickly restored.
J) Regular Maintenance

No security plan is foolproof. You need regular backups to ensure that you can restore your system quickly if required. With both the database and file system it is important to have both local and remote backups. You want the local backup because that allows you to quickly restore the site if there is a problem. You want a remote backup in case of total system failure. There are many ways to setup and configure this. Some helpful backup solutions include:

- Bacula
- rsync / rsnapshot
- mysqldump

Remember that a backup is only good if it can be restored. It’s a best practice to make use of RAID drives, but RAID should be used as a failsafe and not considered a backup strategy. Backups should be stored regularly locally, but there also need to be regular, long-term backups stored off-site. Make sure to evaluate your backup procedures and test your restores to verify that they are working effectively.

Drupal.org releases security updates on Wednesdays when needed which are broadcast by an email list, RSS feeds and Twitter. Subscribe to the security newsletter for updates (you will need a Drupal.org account and the instructions are on the sidebar of the previous link). It is also useful to check the Status page and Watchdog pages in your Drupal site.

SELinux provides auditing services which are worth monitoring. You should be watching your server logs, particularly your Apache error log:

```
tail -f /var/log/httpd/error_log
grep 'login.php' /var/log/httpd/error_log
egrep -i "denied|error|warn" /var/log/httpd/error_log
```

Security best practices are constantly changing. Earlier this year OWASP released their Top 10 for 2013 and it is somewhat similar to the 2010 list. The Top 10 for 2010 was leveraged to look at how it applies to Drupal. This needs to be updated, and reviewed, particularly if you are writing any custom code.
K) Points of Debate - Security by Obscurity

There is a bit of a division within the security community as to whether one should expose information about what versions of software are being used.

1) Make it Obscure

Leaving a CHANGELOG.txt file visible does nothing to improve security, rather it only helps inform an attacker how to focus their research efforts to find a zero day attack, a contrib module vulnerability even faster, or just disable any scripted attacks that might be designed to be Joomla or Wordpress specific. Justin C. Klein Keane in his blog Open source software security strongly recommends hiding both the Drupal and server identification.

2) Make it Transparent

In many cases where the CHANGELOG.txt has been removed, it is because the webmaster hasn't done a Drupal Core upgrade and they are looking for a way to obscure that fact. By keeping the CHANGELOG.txt up-to-date at the very least it indicates that someone is paying attention to security updates.

There are easy ways to fingerprint Drupal and the security team could hide access to this file in the .htaccess file that comes with Drupal Core if they were concerned.

By making it transparent, there is an additional reason for developers to make it a priority to upgrade Core when there is a security release.

3) Be consistent

Ultimately one has to know the organization and individuals that are maintaining the site determine if it is better to hide the CHANGELOG.txt or make it visible. What there is agreement on is that when security releases are announced, that developers apply them quickly such that the site cannot be compromised.

The Linux distribution, Apache & PHP also announce information by default which can be turned off in their configuration files. It is good to be consistent and have your reasoning documented so that it is clearly understood.
L) Additional Resources

1) General guidelines

Drupal security
- Standards, security and best practices - Drupal.org wiki
- Writing secure code - Drupal.org wiki
- Securing your site - Drupal.org wiki
- Drupal Security Group Discussion
- Drupal Security Report - Acquia
- Drupal Security - Acquia
- Security: How the world's largest open source CMS combines open & security - Acquia
- Drupal, SSL and Possible Solutions - Acquia
- Drupal Watchdog Magazine - Security Edition

Secure hosting
- Hardening an SSL server against the NSA - xin.at
- Security in a Box - Tactical Technology Collective
- LinuxSecurity.com
- COTS Security Guidance (CSG) (CSG-09\G) Intrusion Prevention System (IPS) - CSEC
- COTS Security Guidance (CSG) (CSG-10\G) Overview of OS Security Features - CSEC
- How to Deploy HTTPS Correctly - EFF.org

2) Videos
- Doing Drupal Security Right - DrupalCon London
- Building and Securing Government Drupal Sites in the Cloud - DrupalCon Denver
- Securing Drupal Sites for Government Agencies - Acquia
- Drupal Videos About Security on Archive.org
- Semantic Forgeries in Drupal's Form API - Greg Knaddison

3) Third party tools
- Retina Network Security Scanner - beyondtrust.com
- N-stalker - Web Application Security Scanner
- Syhunt - Web Security Audits
- Greensql - Database Security

4) Books
- Cracking Drupal by Greg Knaddison
- O'Reilly.com's Linux Server Security by Michael D. Bauer
- Hacking Linux Exposed by Bri Hatch & James Lee
- Announcement of New Cyber Security Books published by scitech
- SELinux System Administration by Sven Vermeulen

This is a Living Document. Please contribute enhancements