# CHMOD TUTORIAL AND XOOPS FILE SECURITIES

The Xoops Site Security Guide has dealt with securities issues almost exclusively about possible intrusions or threats from outside of your Xoops site or server.

This tutorial provides information on how to secure your files and information from inside using file permissions.

Permissions are the way you regulate access to all your web files. Unix allows you to set different levels of access to a file or a directory for different groups of users.

Xoops default setup and the suggested security measure are merely making sure everyone can read the file while preventing them from overwriting it. For example, "**chmod 444 mainfile.php**" essentially prevents anyone from writing and executing the file, but it allows reading the file by you, web server and **everyone else with whom you share the web server.**

Notice that **everyone** can read the file. If they can read your mainfile.php then they can gain access to your database and do whatever they wish.

We will cover this later in the guide on you should change it. But first onto a short chmod tutorial.

## Unix Groups and Users

There are three categories under Unix file system.
**User** - consists only of the owner of the file, in most cases, your account;
**Group** – include other users on the server;
**Other** – consists of everyone else. Web server falls into "Other" categories.

Please note that you could remove permissions to Group entirely if you think it is necessary and this is the area we will be focusing on.

## Unix Permissions

There are three types of permissions under Unix files/directory:
**Read** – allows a user or a program to read the data in a file or directory
**Write** - allows a user or program to write new data into a file, and to remove data from it
**Execute** - allows a user or program the ability to execute a file, if it is a program or a script

## How To Tell File/Directory Permissions

If you do a directory listing, "ls –l", you will see an output similar to the following

```
drwxrwxrwx   2   username   users   2560 Apr  5 14:56 templates_c
-rw----r--   1   username   users   4891 Feb  1 10:58 mainfile.php
```

First column is the type of the permission; the third column is the user who owns the files or directory (it should be your username)

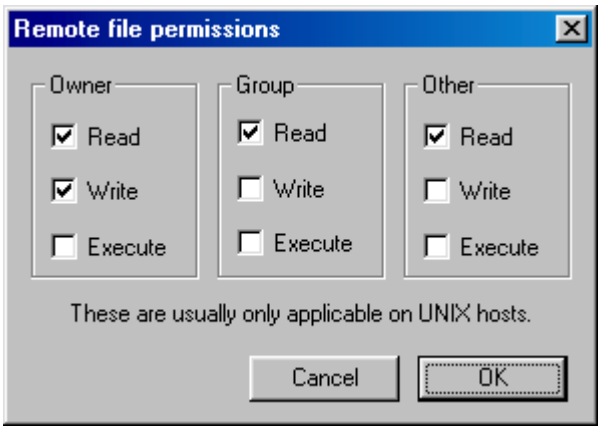
The UNIX permission listing consists of 10 letters or symbols "-". The first one indicates if it is a directory or a file – d means a directory; symbol "-" a file; l, a link. We can tell that templates_c is a directory while mainfile.php is a file.

The next nine letters or symbols are the permission assigned to the file or directory.
1st 3 – permission assigned to **User**
2nd 3 – permission assigned to **Group**
3rd 3 – permission assigned to **Other**

In our example, templates_c has a permission (rwxrwxrwx) allows **User** (you), **Group** (everyone that shares your server), and **Other** (web server) to read, write and execute the directory.

### Setting Permissions Via FTP Clients

Many FTP clients support a visual method of setting file permissions. Below is a screenshot of the visual file permissions setup for WS-FTP client ('right' clicking on the file and then selecting CHMOD from the menu that pops up):




(Equivalent to chmod 644)

## Setting Permission Via Telnet or SSH

You can use chmod command to set or change file permissions. chmod has two distinct methods of operation.

First and maybe easier to understand method – text method: the letters u (for **User**), g (for **Group**), and o (for **Other**), along with the letters r (for read permission), w (for write permission), and x (for execute permission) are used with + (plus), - (minus) and = (equals) to alter permissions from a file.

**chmod u=rwx file.html**
(**User** group has the right to read, write and execute on .html files)
**chmod g-rwx secret.txt**

(**Group** is removed/substracted the right to read, write and execute on file secret.txt)

**chmod o+rwx weblog.txt**
(**Other** is assigned read, write, execute rights on weblog.txt file)

**chmod u=rwx, g-rwx, o=r other.html**
(**User** is given read, write, execute right; **Group**'s read, write, execute right is taken away; **Other** is given read right)

Second method: numerical codes - special numeric codes are used in place of the letters system. Each permission level is assigned a value, as per the following chart:

**Permission Value**
Execute - 1
Write - 2
Read - 4
No permissions - 0

To determine the value of a set of permissions, their numbers are added. For example:
**5=1+4** - has 1 execute and 4 read rights;
**6=2+4** - has 2 write and 4 read rights;
**7= 1+2+4** - has 1 execute, 2 write, and 4 read rights

**Numeric Value Permissions**
0 -no permissions
1 - execute permission
2 - write permission
3 - write and execute permissions
4 - read permission
5 -read and execute permissions
6 - read and write permissions
7 - read, write, and execute permissions

To use chmod with numerical permissions, a three-digit number is formed. The first indicates the permissions for **User**, the second indicates **Group** permission and the last indicates **Other** permission.

Some examples:
**chmod 700 private.txt**
(only **User** has full permissions to the file private.txt)

**chmod 755 normal.txt**
(**User** has full permissions; **Group** and **Other** have read and execute permission)

**chmod 707 webfile.txt**
(**User** and **Other** have full permissions and **Group** has no permission to webfile.txt file)

Please note 707 can usually be substituted for 777, and is a more secure as it cuts out direct access by **Group** (other users).

In order for the web server to be able to serve it to your site viewers, **Other** (web server) must have at least **read** access to any normal files of your web and **read** and **execute** rights to any directories.

### General Web Settings

Text Files, Images and PHP Scripts - chmod 604
Directories - chmod 701
CGI Scripts - chmod 705

For more information on the chmod, check out the Unix Manual Pages. Or you can find it by going to a Unix prompt and typing **man chmod**.

### Xoops File Securities

Warning: Changing permissions to your web files could have a serious consequence of rendering your web site not functioning properly.
**You are proceeding at your own risk.**
Before you attempt to make a change, write down the permission of the particular file, so you can reverse back to the original permission, if your changes made your website stop working.

### Mainfile.php

Xoops stores your sensitive database information in mainfile.php file and with a recommended file permission of 444 or 644. This means everyone has the right to READ the file. **User** (you) and **Other** (web server) are fine, what about **Group**? Anyone sharing the same server could potentially read your database username and password!

**Suggested security measures:**
1. Move the sensitive information out of mainfile.php as suggested in the Security Guide and do the followings:
**chmod 604 xoops-auth.php** (**User** has read/write permission, **Other** has read permission)
**chmod 701 securedata** (grants full permission to **User** and only execute permission to **Other,** so the server will be able to access the protected directory and read the authentication file)

Notice we remove permission to **Group**, so other users on the same server can't enter the directory.

2. If you have not moved out the information and protected with an .htaccess. Change the permission by doing:
**chmod 404 mainfile.php**

chmod 404 or chmod 604 should apply to any of your web files storing sensitive information, not just mainfile.php file.

## Cache, Uploads, Template_c Directories

The suggested chmod 777 can be modified to 707, as there is not a single reason that **Group** need to access those directories.

## Securing Your Web Root

Be default, your web hosting company may assign full permission to your web root directory to anyone. The best and effective way of stopping other users from reading your web files is to change the permission and prevent other users from entering your web root directory or any other directory storing sensitive information (such as securedata directory in the example).

If they can't get into your directory, then they can't read your files regardless of the file permissions.

If your web root directory is public_html, you can do the following

**chmod 705 public_html**

**User** will have full permissions and **Other** will have read and execute permissions. By doing this, users on the same server will be prevented ever getting into your root directory while the web server will continue to function properly.

Please note that you may not be able to do this if some of your hosts' programs (if belong to Group) need to access your web root. If this is the case, you will need to secure your files individually (such as chmod 404 your files).

Attacks from the insider of the server are rare but they can happen potentially. The best safe guard is to make appropriate file permissions and to prevent your sensitive information from being read by others.