

Hardening WordPress the geek way

2010-01-17 20:30:16

My Wordpress installation was hacked several times, and all because I was too lazy to use basic protection and hardening mechanisms. After some time I decided to investigate more on the security of my blog and applied some well known protection techniques as well as some really geeky dirty hacks right in the database.

For the first part of this series, I just want to show my really geeky dirty hackish database-level protection. Other not so geeky and more proper protection techniques can be found at:

- [Security whitepaper for wordpress incl. several very usefull plugins](#)
- [interesting article](#)
- [first wordpress antivirus plugin](#)
- [PHPIDS – PHP based intrusion detection and prevention system](#)

Perhaps I'll cover few of them in later articles.

The dirty hack

Usually the blogs on my dedicated system are owned and used by only one person as they are personal blogs. So there is only need for two user accounts on a Wordpress install: One for the person owning the blog and one for the admin for maintainance purposes (like updating plugins and the WP core).

When a blog is set up, changes on the wp_users are redundant for my blogs. Most hackers will try to create (by SQL-Injection) a new user with extended rights or modify the authentication data of existing users (like the one for admin). So I had the idea of preventing modifications on the wp_users. This only can be achieved within the database itself.

After experimenting with the user rights in mysql (GRANT, REVOKE) I found out that restricting the access to only the wp_users table would need to set access rights to all other tables. No option for me.

I investigated a little bit and found, that there is another possibility: TRIGGERS, which need at least mysql 5+ Version.

In my database I have some administrator user who may do everything in the database, I'll call this one "dba". For the wordpress I create another user as described in the security whitepaper with less rights. Then I place a trigger on the wp_users table which avoids DELETE and INSERT except by the dba:

```
delimiter // CREATE TRIGGER `mywordpressdb`.`root_insert_only` BEFORE INSERT ON
`mywordpressdb`.`wp_users` FOR EACH ROW BEGIN DECLARE dummy INT; IF
STRCMP(SUBSTRING(SESSION_USER(),1 , 4), 'dba@')<>0 THEN SELECT 'no' INTO dummy FROM
dual.id WHERE dual.id=NEW.id; END IF; END // CREATE TRIGGER
`mywordpressdb`.`root_delete_only` BEFORE DELETE ON `mywordpressdb`.`wp_users` FOR EACH
ROW BEGIN DECLARE dummy INT; IF STRCMP(SUBSTRING(SESSION_USER(),1 , 4),
'dba@')<>0 THEN SELECT 'no' INTO dummy FROM dual.id WHERE dual.id=12; END IF; END //
delimiter ;
```

This both triggers only allow inserts and deletes by the dba user (who has to login into mysql by a shell account). All other users get some sql-error on trying it.

As there was the admin-account reset bug in WP, and a SQL-injection could update an existing login

account, I also added a trigger to updates. Since a blog-user may want to change his email or other account data (except password and login name), the trigger for updates restricts changes to that fields:

```
delimiter // CREATE TRIGGER `mywordpressdb`.`restricted_update_only` BEFORE UPDATE ON
`mywordpressdb`.`wp_users` FOR EACH ROW BEGIN DECLARE dummy INT; IF
STRCMP(SUBSTRING(SESSION_USER(),1 , 4), 'dba@')<>0 THEN SET NEW.ID=OLD.ID; SET
NEW.user_login=OLD.user_login; SET NEW.user_pass=OLD.user_pass; SET
NEW.user_registered=OLD.user_registered; SET
NEW.user_activation_key=OLD.user_activation_key; SET NEW.user_status=OLD.user_status; END
IF; END // delimiter ;
```

The update trigger restricts changes to all fields but the ones set in the trigger itself. This hack prevents changing of login-accounts but to the cost of some comfort loss. If a user wants to change his password we now have to log into the database as user dba and execute some query. But at least, this gives some more security to our blog.

By the way: Since newer versions of WordPress use some different encryption for the passwords, you should use [a wordpress password hasher](#) instead of MD5 when updating user passwords manually.

Of course, this makes no sense if you use weak passwords and the intruder can apply some brute force or hash-lookup to find it, or if you have multiple users using one blog and you have to recently update login data or are no geek.