

From a Site Compromise to Full Root Access - Symlinks to Root - Part I

When an attacker manages to compromise and get access to a website, they won't likely stop there, they will aim to gain full root (admin) access to the entire server. If there are more websites hosted on the server being attacked, it is likely they will attempt to compromise every single one of them.

How can an attacker escalate their privileges? How can they go from FTP-only access to getting root on the server? In this series of articles we will show some techniques that attackers are using to go from confined FTP/web access, to full root level access on a server.

1 - Symlinking to / (root)

Most shared hosts only provide restricted FTP access to their clients. Within these restricted FTP accounts an end-user can only see the content of their own websites, or shared space. This segmentation provides an additional level of security because end-users cannot use shell access to poke around the server.

However, there is an old technique that we still see often. What attackers will do is link a sub directory within the website to the root "/" directory allowing them to browse the whole server. All they do to execute this is run a simple PHP command:

```
symlink("/", "./symroot");
```

What you see here is the directory symroot linking to the root "/" directory. What some hosts do is attempt to thwart this type of activity by employing tighter controls on their PHP settings by restricting file access to only the user home directory. This can be bypassed if the hosting provider supports custom cron jobs, custom procmail rules or even cgi-bin scripts (yes, we have seen it all). In some cases, the attackers can create a custom php.ini file to overwrite the default and more restrictive rules set by the server provider.

In the end, an attacker will take whatever you give them. On most shared servers (Even servers on some of the more popular hosting services), attackers can create a symlink like like we showed above to escalate access. In other cases it's a bit more work to do but still very much possible.

Navigating through the file system

It doesn't matter how the attacker manages to link the root, but once they do, they have the privileges to navigate through the file system using a plain old browser. In most of the attacks we're seeing, attackers are also adding a .htaccess file that treats all files as plain text. The danger is that all files

can then be viewed remotely.

Here is an example from a compromised server where the directory /sym was created and the file named root was added linking to /:

```
ncftp /www/ > cd sym
OK. Current directory is /www/sym
ncftp /www/sym > ls -la
drwxr-xr-x 2 userX userX 4096 May 19 10:47 .
drwxr-xr-x 36 userX userX 4096 May 19 10:47 ..
-rw-r--r- 1 userX userX 175 May 19 10:47 .htaccess
lrwxrwxrwx 1 userX userX 1 May 19 10:47 root -> /
```

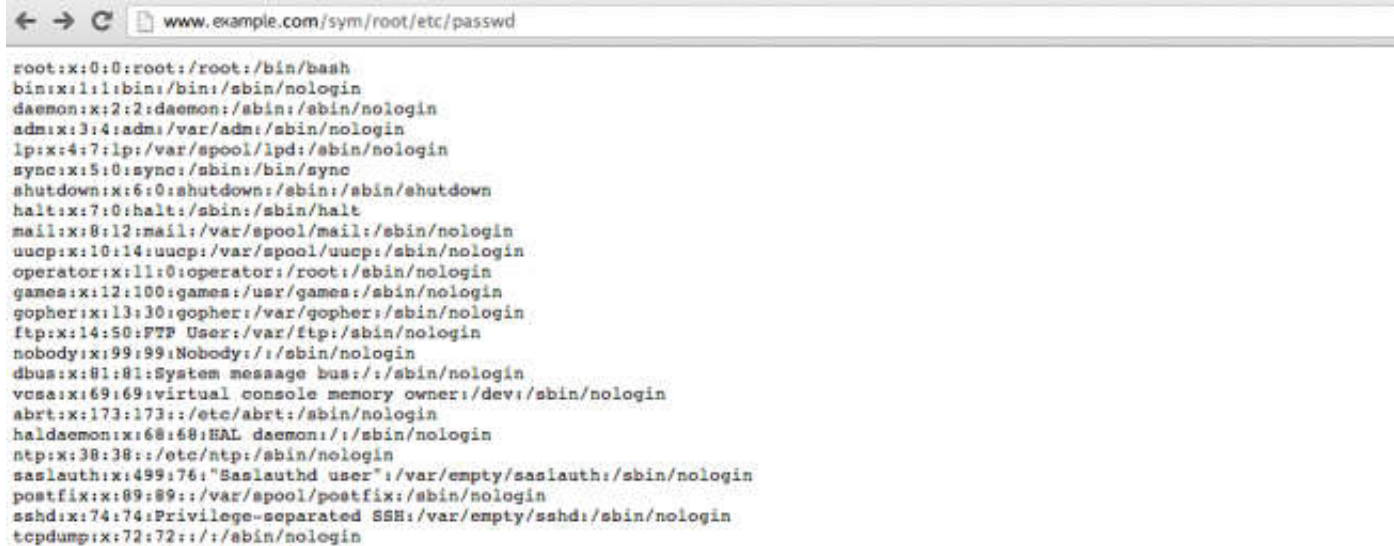
If you look at the .htaccess file, it has custom rules to treat all PHP, Conf, .log and .sql files as plain text, so the content is visible in a regular browser:

```
Options all
DirectoryIndex Sux.html
AddType text/plain .php
AddType text/plain .conf
AddType text/plain .sql
AddType text/plain .log
AddHandler server-parsed .php
AddHandler txt .html
Require None
Satisfy Any
```

If you try to visit the /sym/root directory via the browser, this is what you get:

Pretty awesome to see that you get access to all the directories on the server.

The user can even go to /etc/passwd to see the list of users:



```

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:system message bus:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
abrt:x:173:173:/:/etc/abrt:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
ntp:x:38:38:/:etc/ntp:/sbin/nologin
saslauthd:x:499:76:"Saslauthd user":/var/empty/saslauthd:/sbin/nologin
postfix:x:89:89:/:var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/ssh:/sbin/nologin
tcpdump:x:72:72:/:/sbin/nologin

```

That's not even the worse part. They can query configuration files, backup files and use this to try to steal the password for other users in the server.

Protecting against Symlink attacks

A quick solution for this specific attack is to prevent Apache from following symlinks (Options -FollowSymLinks). You can also prevent end-users from overwriting the default security rules (AllowOverride None). If you really need links, you can use the "SymLinksIfOwnerMatch" option to only allow links from within the same user (not to /).

This alone doesn't solve the whole problem. The client still has PHP access and can just opendir("/") and access all directories. To prevent PHP from accessing any file outside of their directory, you need to specify the open_basedir setting to only have access to their directory.

Even then, this still doesn't cover all angles if the client still has SSH access, can create custom cron jobs, or if you allow shell exec inside PHP (system, exec, " , etc).

Permissions!

The real solution against this problem is setting very tight permissions on your servers so that even if a bypass is found, attackers won't be able to query or access any critical directories and/or files, or user information.

In the conclusion of this series, we will show our recommended permissions and setup to help address these issues.