Browser-based shell access

Muscle Play

Firewalls often block shell access for remote users on a corporate network or at an Internet cafe. Luckily, tools like PHP Shell and Shell in a Box put the shell in a browser window.

By Wolfgang Dautermann

Thanks to Secure Shell (SSH), managing a server externally is quite easy, and X forwarding means you can even use graphics programs for these management tasks on your computer at home. Unfortunately, in a corporate environment or an Internet cafe, you probably won't be allowed to install additional software, or the firewall settings will be so restrictive that nothing but HTTP/HTTPS will get through. PHP Shell and Shell in a Box give you options for accessing your server.

PHP Shell

PHP Shell supports shell access to servers for cases in which either the server administrator or the firewall blocks it. A PHP-capable web server is all you need to run shell commands. PHP Safe Mode, which is a (legacy) attempt to make PHP safer on web servers that host various sites by restricting various commands, must be disabled for this to work.

The installation is quite simple: Download the latest version from the PHP Shell website [1] and unpack the ZIP archive in a directory in webspace. You need to set a password for this; to do so, go to *http://www.example.com/phpshell/pwhash.php* (taking care to replace *www.example.com* with your site name) and enter your choice of username/password.

You then need to add the line returned to you to the `[users]` section of your `config.php` file. If needed, you can add several users. After you've completed this step, PHP Shell is ready for use.

Next, you can go to *http://www.example.com/phpshell/phpshell.php* and log in with your username/password combination (Figure 1) to launch your shell session in the web browser.

Now you can type a shell command at the `$` prompt shown in the bottom of the shell window. Either press Enter or click on *Execute Command* to run the command and view the results in the shell window (Figure 2). Certain restrictions to the shell exist, as PHP Shell shows you in the browser.

Commands can't require additional user input; that is, you can't call interactive programs. A command must fit into one line; PHP Shell doesn't know when

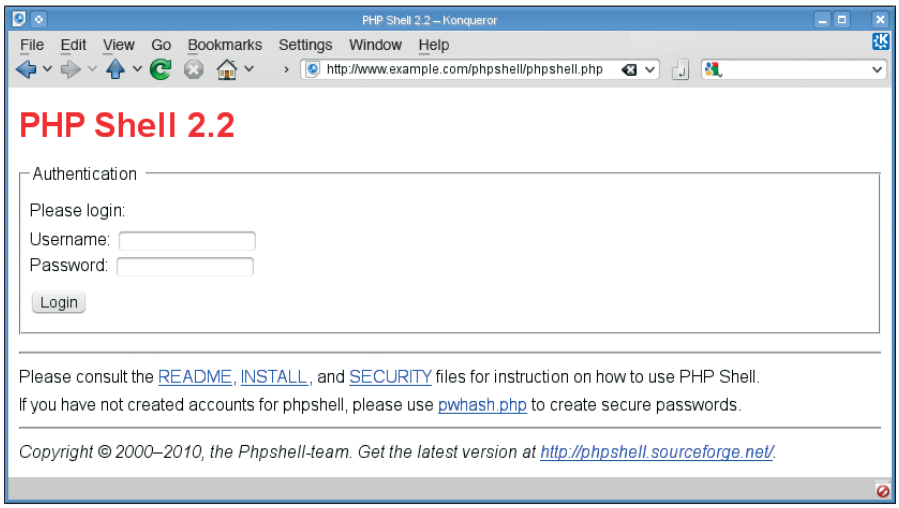

Figure 1: PHP Shell login window.

a command needs to be continued. For example, you can't input a `for` loop in multiple lines, which you can do in the standard shell like this:

```
$ for i in a b c ; do
> echo $i
> done
```

However, you can input this command as a single line:

```
for i in a b c ; do echo $i ; done
```

A command must complete within a certain amount of time (typically 30 seconds). This requirement isn't a PHP Shell restriction; rather, configurable limits used by the Apache web server (the `Timeout` directive) and PHP (the `max_execution_time` setting in `php.ini`) time out the execution.

The commands are executed with the web server's UID/GID, which you can check by entering the `id` command. This feature can be handy because FTP access typically uses a different UID.

For example, if you need to create a directory in which the web server writes, unless you happen to be root, you can only create a globally writable directory with FTP or SSH. In comparison, PHP Shell lets you create a directory in which write access is restricted to the web server.

To change the size of the shell window, use the scaling tool on the bottom right; just enter the required values and then run the next command. PHP Shell includes a simple editor (`editor` *`filename`*) that lets you edit files – they must be writable for the web server's UID. It also includes a simple history function that lets you scroll backward and forward to the previous and next commands by pressing the Up and Down arrows; however, it does not support more advanced features such as history searches.

Besides specifying the user/password (the password could also be stored in the clear, although that's not recommended), you can set up shell aliases and a home directory for PHP Shell in the `config.php` configuration file.

Storing an encrypted password (with `pwhash.php`) will only help prevent a login attempt if an attacker manages to compromise the configuration file. The security of PHP Shell depends on whether access occurs via the encrypted web server (HTTPS) – if it does not, an attacker could sniff any commands entered as well as their output.

Shell in a Box

Shell in a Box (Figure 3) can be very useful if you have shell access to the server and can install and run your own programs there, but you prefer, or are forced, to use a web browser as a client. An overview and a demo of this web-based Ajax tool can be found on the project website [2].

Shell in a Box is not included in the standard repositories. Debian/Ubuntu users can download the package from the project website and install it. Users with other Linux distributions will need to build and install the project themselves with the typical `./configure`, `make`, and `make install`, which will drop Shell in a Box into `/usr/local`.

In contrast to PHP Shell, Shell in a Box comes with its own web server, which listens on port 4200 by default. For your first experiments with Shell in a Box on a local machine, you can probably do without encryption; however, adding Secure Sockets Layer (SSL) security for production deployment on a web server is definitely a good idea. SSL is an encrypting intermediate layer that builds on the TCP/IP protocol, thus supporting the encryption of existing protocols (HTTP, POP3, IMAP, etc.). Shell in a Box also can provide various services. The

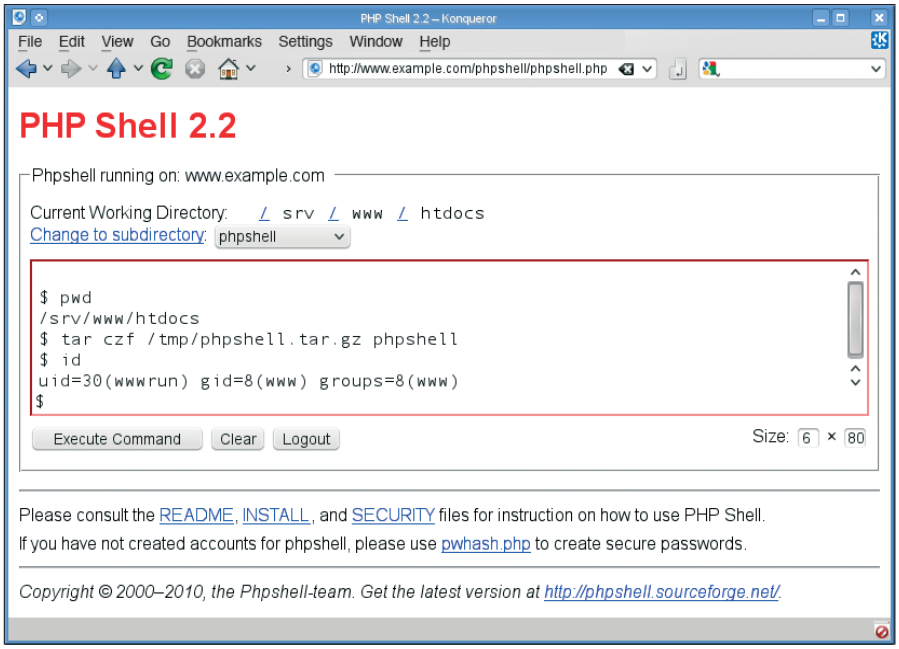

Figure 2: PHP Shell in action.

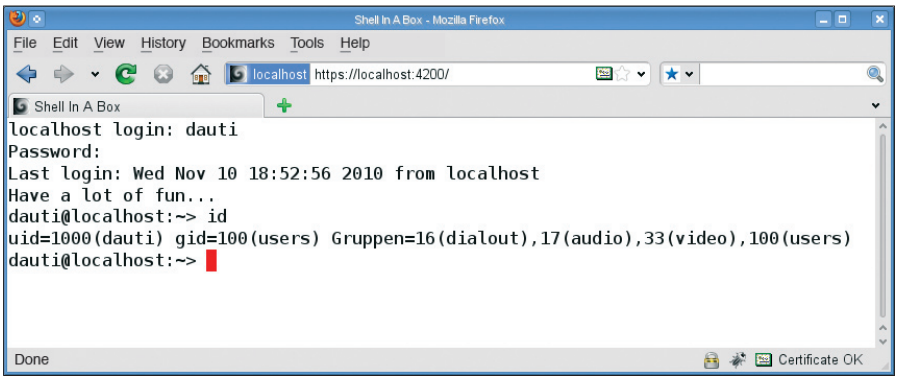

Figure 3: Shell in a Box in action.

general syntax for the Shell in a Box daemon takes the following form:

```
shellinaboxd -s WEBPATH:SERVICETYPE
```

In the following examples, note that the `-t` parameter will disable SSL temporarily:

- Provide a login shell on the local system with a path of *http://localhost:4200/* (only available to root):

```
shellinaboxd -t -s /:LOGIN
```

- Provide an SSH login on the local system with a path of *http://localhost:4200/*:

```
shellinaboxd -t -s /:SSH
```

This assumes that an SSH server is running (at least on the loopback device).

- Provide an SSH login for the remote computer `host.example.com` on the local system with a path of *http://localhost:4200/external-host/*:

```
shellinaboxd -t -s ➥
  /external-host/:SSH:host.example.com
```

An SSH server must be running for this process, which means that you can use `shellinabox` as a gateway to computers that would otherwise be unreachable.

- Run the `top` program on *http://localhost:4200/systemstate/* using the `dauti` UID and the `users` group (with a working directory of `/`):

```
shellinaboxd -t ➥
  -s /systemstate/:dauti:users:/:➥
    usr/bin/top
```

A word of warning: Being able to view the system status in a browser is great, but this feature also allows access to the `top` command's interactive options (such as pressing *k* to kill a process)!

- A `shellinaboxd` process can provide multiple services (e.g., a login on multiple computers):

```
shellinaboxd -t ➥
  -s /host1/:SSH:host.example.com ➥
  -s /host2/:SSH:host2.example.com
```

For example, you could use a URL like *http://localhost:4200/host1/* to connect with `host.example.com` and use a URL of *http://localhost:4200/host2/* to open an SSH connection to another machine (e.g., `host2.example.com`).

In contrast to PHP Shell (wherein each line entered is processed and the results are returned to the browser), Shell in a Box also supports (text-based) interactive activities, such as the use of editors like Vi, Joe, and so on.

Although the performance is not as fast as in a normal text terminal, you probably won't notice any major restrictions. (The `bb` demo for the `aalib` ASCII Art Library is fairly unimpressive in the browser, but you might want to watch it. On openSUSE/Ubuntu/Debian, you can install the bb package and view in a normal terminal.)

If you intend to use Shell in a Box on a production server, you should secure the service with SSL. Shell in a Box comes with this functionality built in. If you leave out the `-t` parameter (or `--disable-ssl`), Shell in a Box will default to providing the service via HTTPS.

Note that you will need a certificate, which must reside in the current directory (you can change the certificate directory using the `--cert=DIRECTORY` option). For more information about certificates, see the "Encryption and Certificates" box.

Conclusion

PHP Shell and Shell in a Box bring the shell to your browser window. You can try these handy tools if you need to access the shell from an Internet cafe or across a firewall. Remember, however, that shell access is a very powerful tool, and it demands close attention to security. You should always use encryption and take the necessary precautions to make sure these tools don't create an opening for intruders. ■■■

ENCRYPTION AND CERTIFICATES

Digital certificates are issued by certificate authorities, who typically provide this service for a fee, although community projects such as CAcert [3] also issue digital certificates. The browser automatically identifies the certificate as legitimate, because it recognizes and trusts the certificate authority, and it uses the server public key obtained through the certificate to encrypt communication with the server.

You also have the option of creating a self-signed certificate, which does not offer authoritative identification and thus is not as secure as working with a certificate authority. Shell in a Box is supposed to create self-signed certificates [4], but this process didn't work in my lab. Check the documentation or visit the website for more details on working with certificates in Shell in a Box.

INFO

[1] PHP Shell: *http://phpshell.sourceforge.net/*
[2] Shell in a Box: *http://code.google.com/p/shellinabox/*
[3] CAcert: *http://www.cacert.org/*
[4] Shell in a Box issues: *http://code.google.com/p/shellinabox/issues/detail?id=59#c11*

AUTHOR

Wolfgang Dautermann is a system administrator who has tamed many flavors of Linux and Unix, such as Solaris, Irix, and Tru64, in his career. He is a co-organizer of Linux Days in Graz, Austria.