Two-Factor Authentication System for Apache and SSH

James Litton

Abstract

If you have personal publicly accessible Web sites and/or publicly accessible SSH services, you should take steps to limit your risks by adding a simple, yet effective two-factor solution.

If you run a publicly accessible Web server for your own use (and let's face it, if you're reading *Linux Journal*, there's a very good chance you do), how do you go about limiting the risk of someone accessing your site and doing bad things? How about SSH, an even bigger concern? In today's world, it's imperative to think about your exposure and take steps to limit as much risk as possible.

In this tutorial, I walk through the steps necessary to implement a home-grown two-factor authentication system for accessing your Web sites and for SSH access.

The Infrastructure and the "Challenge"

Running your own hardware can be a pain in the neck. After dealing with hardware failures, such as failed fans, failed power supplies, bad hard disks and the like, you finally may decide to dump your co-lo or bedroom closet and your hardware and jump into the world of elastic computing. One such option is Amazon's EC2 platform, which offers a variety of Linux flavors and has one of the most robust and mature cloud platforms available. I'm not an Amazon representative, but I'm the first to say *try it*. It's amazing stuff, and a micro instance is free for a year.

In the test scenario for this article, I use an Amazon EC2 server running Ubuntu 12.04 LTS to host a couple Web applications. If you use a different flavor of Linux, the instructions easily can be adapted to meet your specific needs. Let's assume the applications are, for the most part, for personal use only. If the sites were accessed only from work or home, you simply could secure the sites by creating firewall rules to allow Web traffic from only those IP addresses. This, incidentally, is exactly how one should secure SSH.

Let's assume though that this won't work for your Web apps because you do a fair amount of traveling and need to be able to access those applications while you're on the road, so a couple firewall rules won't help you. Let's also assume that your applications have their own security systems, but you still want an extra layer of security.

You could have set up a VPN server, but every once in a while, you might like to give a family member access to one of your sites, so a VPN approach wouldn't work.

Another consideration is Google Authenticator for true two-factor authentication. You certainly could go down this path, but you're looking for something you can do yourself—something that is self-contained and yours.

Just like so many things in the Linux world, where there's a will, there's a way! *It turns out you easily can set up your own, homegrown, two-factor solution and use it to control access to your Web apps and SSH, while also making it possible to allow occasional access to your sites by other users.*

Apache Authentication and Authorization

Since the Web server for this example is Apache, let's leverage the server's authentication and authorization capabilities to ask for a set of credentials before any of your sites are served up to a user.

In the interest of keeping things simple, and since you will follow best practice and allow only https traffic to and from your Web server, let's use the **mod_auth_basic** module for authentication.

Start by becoming root and installing Apache on your fresh Ubuntu install:

```
sudo su apt-get install apache2
```

Let's assume your Web applications run in subfolders off of the main www document folder. This allows you to take care of all your sites at once by creating a single .htaccess file in the http server root folder:

```
vim /var/www/.htaccess
```

Now, let's add a few lines that tell Apache to require authentication and where to look for the password file:

```
AuthType Basic AuthName "restricted area" AuthUserFile /home/ubuntu/.htpasswd require valid-user
```

With that in place, you now need to change the ownership of the file so the Apache process can read its contents:

```
chown www-data:www-data /var/www/.htaccess
```

Next, you need to create the .htpasswd file that you reference in your .htaccess file and configure its ownership so the Web server can read it:

```
htpasswd -cb /home/ubuntu/.htpasswd jameslitton test123 chown www-data:www-data /home/ubuntu/.htpasswd
```

Now you need to tell Apache to require authentication and to use the **mod_auth_basic** module for that purpose:

```
vim /etc/apache2/sites-available/default-ssl
```

Then you need to change **AllowOverride None** to **AllowOverride AuthConfig**:

```
Service apache2 restart
```

Visiting your site now prompts for a user name and password (Figure 1).

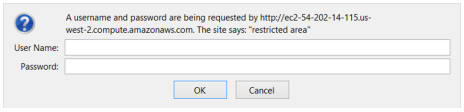

Figure 1. Authentication Request from **mod_auth_basic**

One-Time Day Password/PIN

The approach I'm going to take here is to have your secondary authentication password change daily instead of more frequently. This allows the **mod_auth_basic** approach described above to work. I won't go into the details here, but suffice it to say that every time the password changes, an immediate re-authentication is required, which is not the behavior you want.

Let's go with a six-digit numeric pin code and have that delivered to a mobile phone at midnight every night. I'm a big fan of Pushover, which is a service that pushes instant notifications to mobile phones and tablets from your own scripts and application.

To set this up, create a bash script:

```
vim /home/ubuntu/2fac.sh
```

Now add the following lines: *Garrick, shrink below.*

```
1 #!/bin/bash 2 ppwd="od -vAn -N4 -tu4 < /dev/urandom | tr -d '\n' | tail -c 6" 3 curl -s -F "token=id" -F "user=id" -F "message=$ppwd"  "https://api.pushover.net/1/messages.json 4 htpasswd -b /home/ubuntu/.htpasswd jameslitton $ppwd 5 echo $ppwd | base64 >/home/ubuntu/.2fac
```

Line 2 produces a random six-digit PIN code and assigns it to a variable called ppwd. Line 3 sends the PIN to the Pushover service for delivery to your mobile phone. Line 4 updates the .htpasswd file with the new password, and last but not least, Line 5 stores a copy of the PIN in a format that you can recover, as you will see later on.

Now save the script, and make it executable:

```
chmod +x /home/ubuntu/2fac.sh
```

To complete this solution, all you need to do is schedule the script to run, via cron, at midnight each night:

```
crontab -e 00 00 * * * /home/ubuntu/2fac.sh
```

Making It Web-Accessible

You certainly could leave it there and call it done, but suppose you didn't receive your code and want to force a change. Or, perhaps you gave someone temporary access to your site, and now you want to force a password change to ensure that that person no longer can access the site. You always could SSH to your server and manually run the script, but that's too hard. Let's create a Web-accessible PHP script that will take care of this for you.

To start, change the ownership of your 2fac.sh script so your Web server can run it:

```
chown www-data:www-data /home/Ubuntu/2fac.sh
```

Now you need to create a new folder to hold your script and create the PHP script itself that allows a new "key" to be run manually:

```
mkdir /raw/www/twofactor vim /var/www/twofactor/index.php 1 <?php 2 exec('/home/ubuntu/2fac.sh'); 3 header('Location: http://www.google.com'); 4 ?>
```

Because it's conceivable that you're needing to force a new key because you didn't receive the previous one, you need to make sure the folder that holds this script does not require authentication. To do that, you need to modify the Apache configuration:

```
vim /etc/apache2/sites-available/default-ssl
```

Now add the following below the Directory directive for /var/www:

```
<Directory /var/www/twofactor/>   satisfy any </Directory>
```

Now let's configure ownership and restart Apache:

```
chown -R www-data:www-data /var/www/twofactor Service apache2 restart
```

So thinking this through, it's conceivable that the Pushover service could be completely down. That would leave you in a bad situation where you can't access your site. Let's build in a contingency for exactly this scenario.

To do this, let's build a second script that grabs a copy of your PIN (remember the .2fac file that you saved earlier) and e-mails it to you. In this case, let's use your mobile carrier's e-mail to SMS bridge to SMS the message to you.

Start by installing mailutils if you haven't done so already, and be sure to select the Internet option:

```
apt-get install mailutils
```

Now create the second script:

```
vim /home/Ubuntu/2fac2.sh
```

Then add the code:

```
#!/bin/bash ppwd="cat /home/ubuntu/.2fac | base64 --decode" echo " " | mail -s $ppwd xxx5551212@vtext.com
```

Don't forget to change the file's ownership:

```
chown www-data:www-data /home/ubuntu/2fac2.sh chown www-data:www-data /home/ubuntu/.2fac
```

With that out of the way, now you need to modify the PHP script:

```
vim /var/www/twofactor/index.php
```

Replace line 2 with the following:

```
2 if (isset($_GET["sms"])) { 3 exec('/home/ubuntu/2fac2.sh'); 4 } else { 5 exec('/home/ubuntu/2fac.sh'); 6 }
```

Then create two bookmarks, so that any time you want to generate a new PIN and have it sent to you via Pushover, you simply can click the link and it's done. The second bookmark will send a copy of the existing PIN to the e-mail address of your choice in the unlikely event that the Pushover service is unavailable.

- 2Factor = https://www.thelittonfamily.com/twofactor/index.php
- 2Factor—SMS = https://www.thelittonfamily.com/twofactor/index.php?sms=1

Extending to SSH

Extending this solution to cover SSH is really pretty simple. The key is to use the little-known **ForceCommand** *directive in your sshd_config file.* This forces the SSH diemon to run a script before spanning the terminal session.

Let's start with the script:

```
vim /home/ubuntu/tfac-ssh.sh
```

Now add the following lines: *Garrick, shrink below.*

```
1 #!/bin/bash 2 code="cat .2fac | base64 --decode" 3 echo -ne "Enter PIN: " 4 while IFS= read -r -s -n1 pass; do 5 if [[ -z $pass ]]; then 6  echo 7  break 8 else 9  echo -n '*' 10  input+=$pass 11 fi 12 done 13 if [ $code = $input ]; 14 then 15 sleep 1 16 clear 17 /bin/bash 18 else 19 sleep 1 20 curl -s -F "token=id" -F "user=id" -F "message=$input"  "https://api.pushover.net/1/messages.json 21 fi
```

Line 2 loads the PIN into a variable. Lines 3–12 prompt for the PIN and echo a star back for each key press. Line 13 compares the user's input to the PIN. If they match, lines 14–17 clear the screen and start a bash session. If the user's input does not match the PIN, lines 18–21 send a notification to Pushover so you know a failure occurred and then ends the session.

Let's configure the SSH diemon to run the script:

```
vim /etc/ssh/sshd_config
```

Now add the following to the top of the file:

```
ForceCommand /home/ubuntu/tfac-ssh.sh
```

Figure 2. Two-Factor Request from SSH

This approach works great. The only limitation is no backspaces. If you press the wrong key, your session will be terminated, and you'll have to try again.

There you have it, a poor-man's two-factor authentication implementation with very little effort and from my experience, it's rock solid!

11619%1.qrk

Send comments or feedback via http://www.linuxjournal.com/contact or to <2-factor@linuxjournal.com>.