Group Project Final Report

Eric Walker, Mariano Pennini, Xiaoyu Shi

December 3, 2018

1   Introduction

Our code: **https://github.com/xiaoyu-iid/Simplog-Exploit**

Our group was assigned CVE-2009-4092. The vulnerability exists in the user.php file of Simplog 0.9.3.2, an application used for blogging. Attackers can use cross-site request forgery (CSRF) to hijack the user authentication process by sending a malicious request to change the password. Upon following a malicious URI crafted by the attacker, given that the victim has an active session on the target website, the victim's password will be changed. Since the application cannot tell the difference between a same-site and cross-site request, it leaves itself vulnerable to this type of attack.

In order to patch the vulnerability, we will add a mechanism to the simplog code to distinguish same-site and cross-site requests and reject those requests which are cross-site and unauthenticated. Using a unique security token generated by the application at the beginning of each new session for each user, every request within the application can be sent with this token to verify that the request is a same-site request. By making the secret token randomly generated and of sufficient length, there is no way for attackers to correctly guess the secret token with noticeable probability.

2 Requirements

- PHP 5.6

- MySQL 5.7

- SEED VM: Ubuntu 16.04

- Apache 2.4.18

Notes on requirements:

The Simplog code is quite old, circa 2006. Consequently, it requires an antiquated version of PHP (5.6) in order to run most of the MySQL commands. The difference is mainly a result of PHP's switch from `mysql_*` commands to `mysqli_*` commands. The `mysql` PHP library was deprecated in PHP 5.5 and dropped in PHP 7.0. The `mysqli` library was introduced to patch a number of vulnerabilities in the old way of processing MySQL queries. The Simplog code does not take advantage of these patches and thus is also vulnerable to a number of SQL injections, documented in https://nvd.nist.gov/vuln/detail/CVE-2005-3076.

The figure below shows one such `mysql` error associated with trying to run the Simplog code using PHP 7.x.

```
[Mon Nov 26 17:34:35.170514 2018] [:error] [pid 12776] [client 127.0.0.1:40736]
PHP Fatal error:  Call to undefined function mysql_connect() in /var/www/simplog
/install/install2.php on line 10, referer: http://localhost/simplog/install.php
[11/26/18]seed@VM:.../apache2$
```

3   Installation

The code for Simplog 0.9.3.2 can be found on our github repository. The code can be cloned onto Ubuntu 16.4 VMs with the Web URL provided by github:

```
$ git clone https://github.com/xiaoyu-iid/Simplog-Exploit.git
```
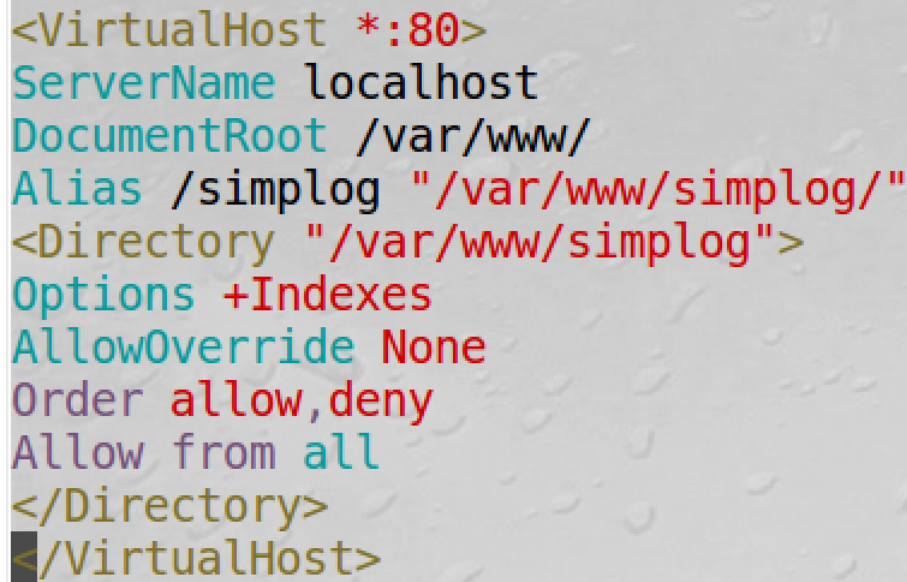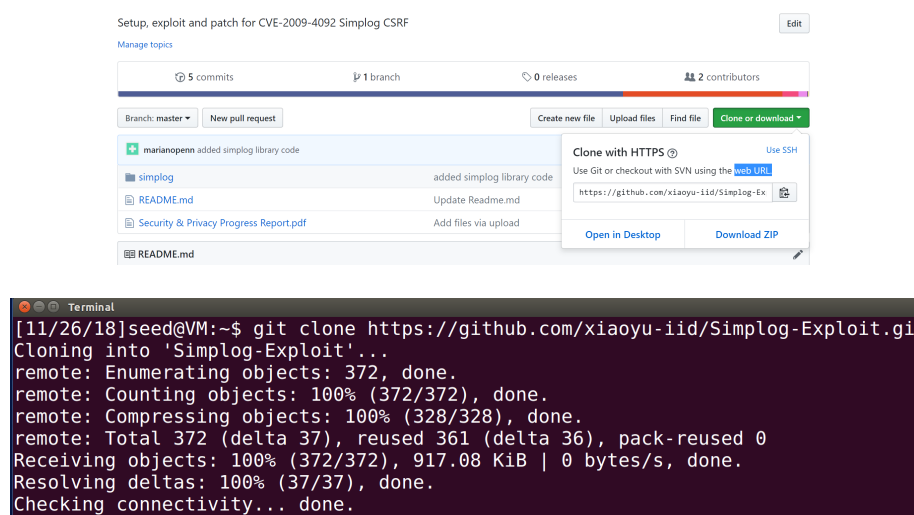

We will use the default installation of Apache, 2.4.18, to serve the Simplog files. Once we have cloned the vulnerable code, we need to first place the entire `simplog/` directory into `/var/www/simplog`. To use the patched version of the code, place the `simplog_patched/` directory into `/var/www/simplog` instead.
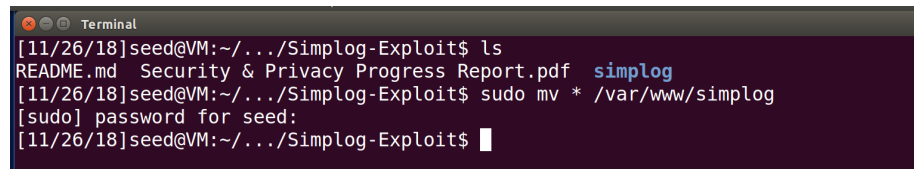
Next, we add the following entry to `/etc/apache2/sites-available/000-default.conf` (you will need root access to modify this file):

```
<VirtualHost *:80>
    ServerName localhost
    DocumentRoot /var/www/
    Alias /simplog "/var/www/simplog/"
    <Directory "/var/www/simplog">
        Options +Indexes
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```


This entry ensures that we have registered a virtual host on our machine for the Simplog server. Apache virtual hosts allow users to host multiple websites on the same machine.

We fix the compatibility issues with PHP7.x by first purging all php packages with the following lines:

```
$ sudo apt-get install aptitude
$ sudo aptitude purge `dpkg -l | grep php| awk '{print $2}' |tr "\n" " "`
```

Note: the first and last ticks should be backticks while the inner ticks are simply single quotes. We then add a Personal Package Archive (PPA) which

gives us compatibility with PHP5 commands so that the `mysql_*` functions will be found. We install php5.6 and the mysql client for it as well:

```
$ sudo add-apt-repository ppa:ondrej/php
$ sudo apt-get update
$ sudo apt-get install php5.6
$ sudo apt-get install mysql-server mysql-client php5.6-mysql
```

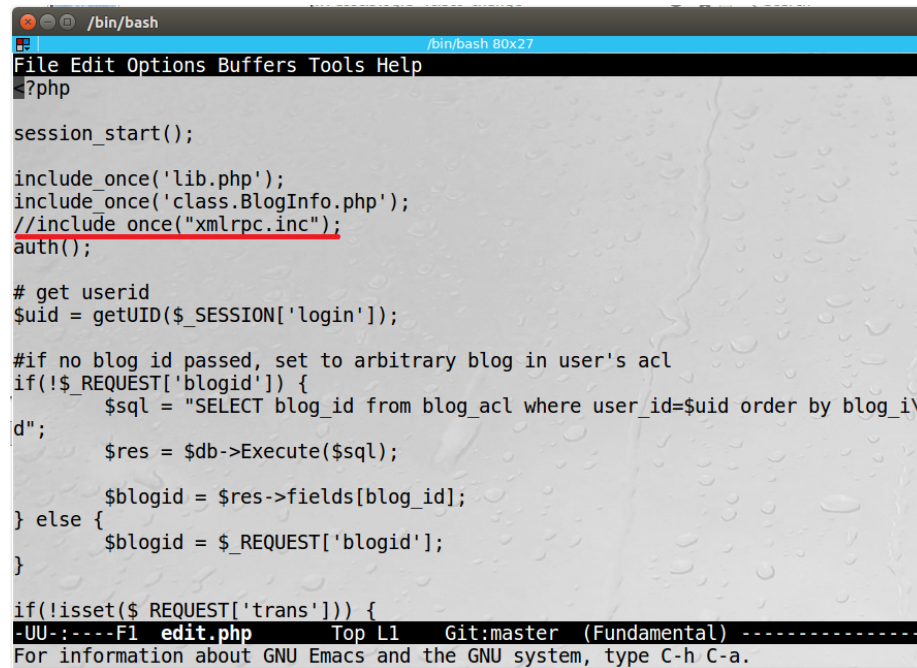Follow the procedure shown in the screenshots below to completely uninstall PHP 7.0:
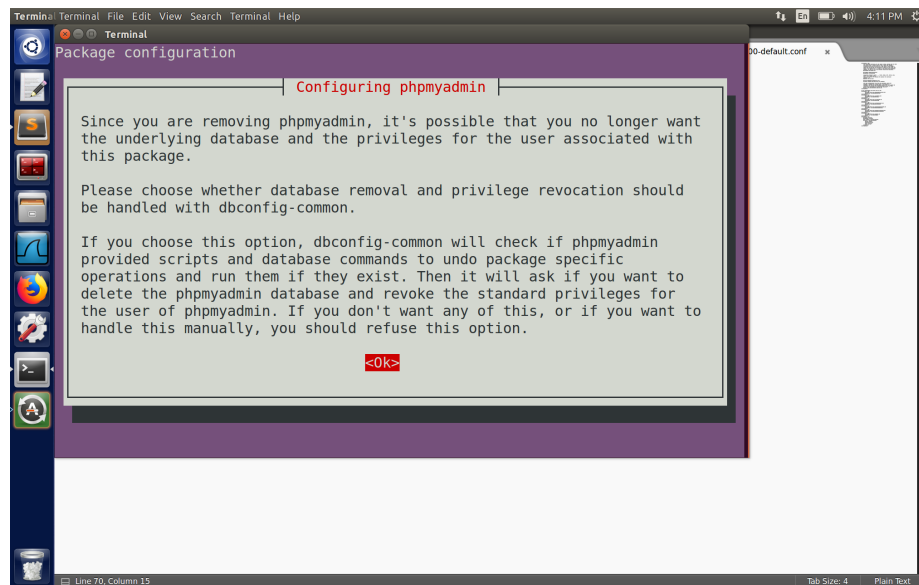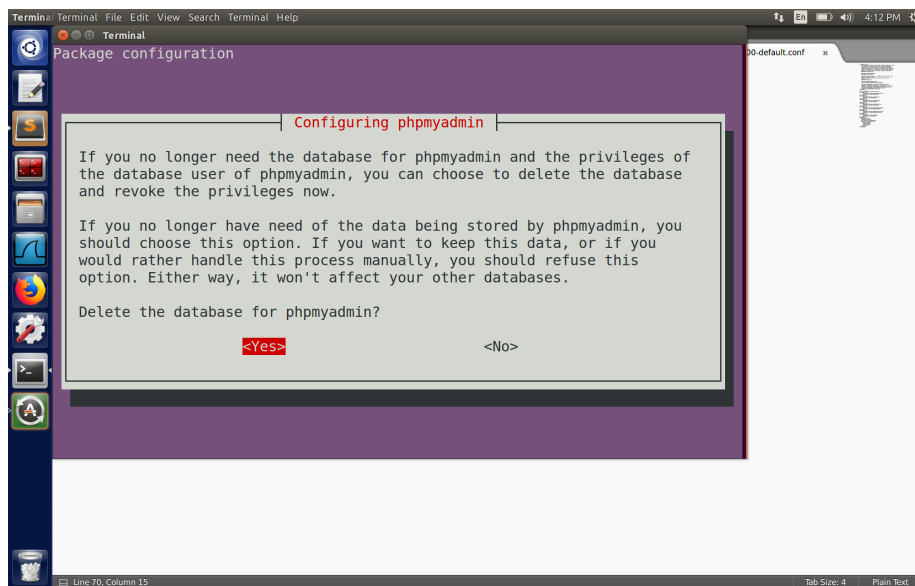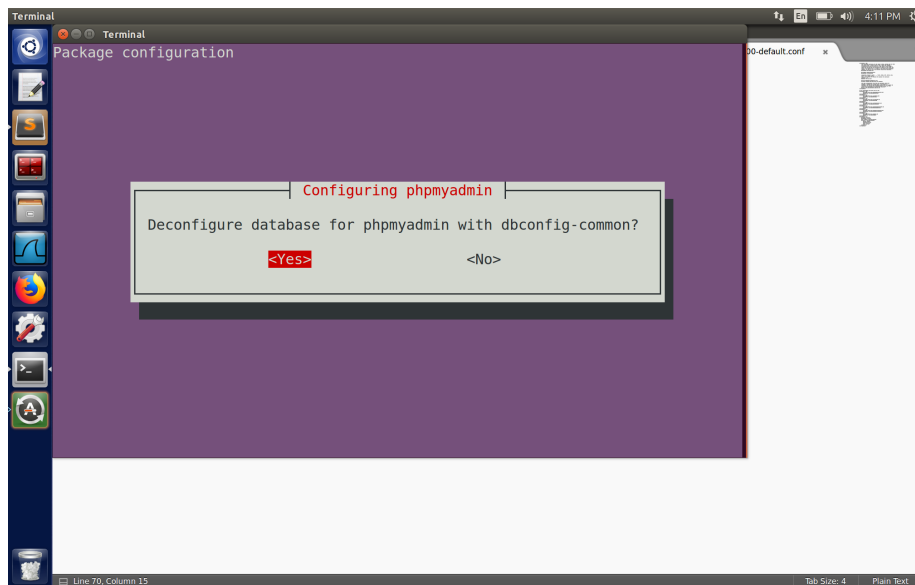
Finally, we need to restart our Apache server and mysql to allow all our changes to take effect:

```
$ sudo service apache2 restart
$ sudo service mysql restart
```

In the `edit.php` file, we comment out the line `include_once("xmlrpc.inc");` because it was causing issues upon login (due to its use of the `dl()` function

among other things) and is irrelevant to the task at hand.

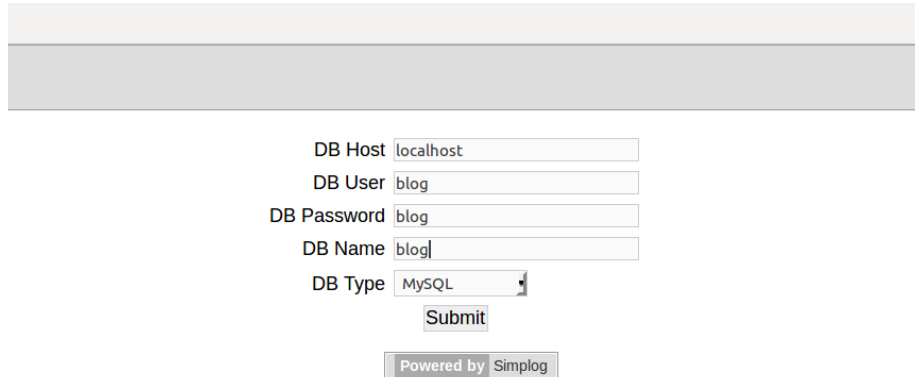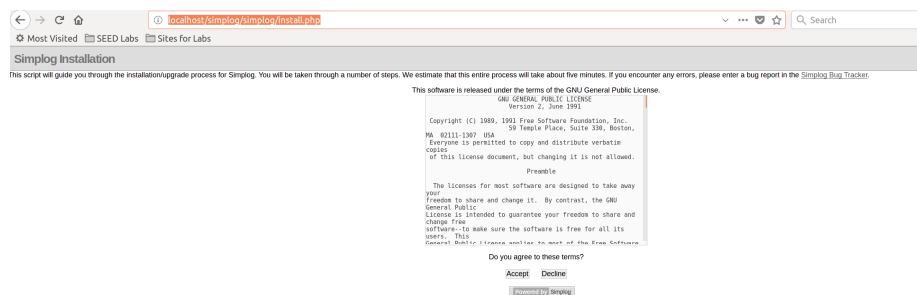After installing PHP5.6 and restarting the Apache server, installation instructions of Simplog can be found at URL `localhost/simplog/install.php`, which is reachable by `Firefox`.

Checkpoint: Be absolutely certain that PHP 5.6 is installed an active AND that there exists an empty database called blog in your mysql console before continuing with the installation.

We click on "Accept" button to accept Simplog terms and arrive at a page to specify the database we want to use with the Simplog application. In our setup, we specify out user to be `blog` with password `blog`, and the database we

use will be a MySQL database called `blog`, which is hosted locally.

After clicking the "Submit" button, we are notified to edit the following lines in `config.php`.


```
27   ##########################################################
28
29   $blogurl = "localhost/simplog/index.php"; //change this
30   $baseurl = "/simplog";  //change this - omit the trailing slash
31   $basepath = "/Library/WebServer/Documents/simplog";  //change this
32
33   $host = "localhost";
34   $dbase = "blog";
35   $user = "blog";
36   $passwd = "blog";
37   $dbtype = "mysql";   // if using PostgreSQL, set to 'postgres'
38
```

The same webpage also notifies that we need to create a new MySQL database `blog`.

We add the skip-grant-tables option to the mysql configuration file to avoid user privilege issues. We can open my.ini with the following command:

```
$ sudo subl /etc/mysql/my.cnf
```

Then we an add the following lines to the end of `my.ini`:

```
[mysqld]
skip-grant-tables
```

```
     config.php        x      my.cnf           x
 1   #
 2   # The MySQL database server configuration file.
 3   #
 4   # You can copy this to one of:
 5   # - "/etc/mysql/my.cnf" to set global options,
 6   # - "~/.my.cnf" to set user-specific options.
 7   #
 8   # One can use all long options that the program supports.
 9   # Run program with --help to get a list of available options and with
10   # --print-defaults to see which it would actually understand and use.
11   #
12   # For explanations see
13   # http://dev.mysql.com/doc/mysql/en/server-system-variables.html
14
15   #
16   # * IMPORTANT: Additional settings that can override those from this file!
17   #   The files must end with '.cnf', otherwise they'll be ignored.
18   #
19
20   !includedir /etc/mysql/conf.d/
21   !includedir /etc/mysql/mysql.conf.d/
22   [mysqld]
23   skip-grant-tables
```

MySQL needs to be restarted with `sudo service mysql restart`. Then we can run MySQL using the following command:

```
$ sudo mysql -u blog -p
```

When prompted for the password, enter `blog`.

```
[11/26/18]seed@VM:.../simplog$ sudo service mysql restart
[11/26/18]seed@VM:.../simplog$ sudo mysql -u blog -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.
```

Now, create a new database called `blog` using `create database blog`.

```
+--------------------+
8 rows in set (0.06 sec)


mysql> create database blog;
Query OK, 1 row affected (0.04 sec)
```

Then we go back to the Simplog setup on `Firefox`. We click on the "New Install" button on the website, then we arrive at a page to specify the username and password for the admin user. In our installation, we specified the username to be `admin` with password `admin`, and with a few other entries:
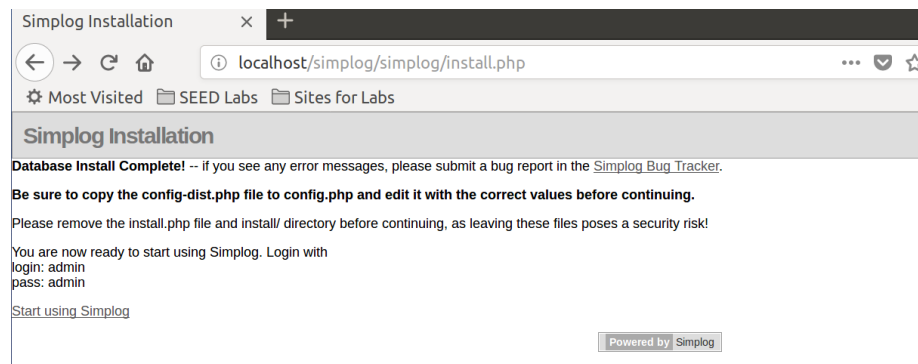
Admin Login: admin
Password: •••••
Re-Enter Password: •••••
Admin Name: admin
Admin Email: admin@email.com
Admin URL: www.admin.com

Simplog will now install the database.
Install!

Powered by Simplog

We can now start using Simplog.

Simplog Installation   +

localhost/simplog/simplog/install.php

⚙ Most Visited   📁 SEED Labs   📁 Sites for Labs

Simplog Installation

Database Install Complete! -- if you see any error messages, please submit a bug report in the Simplog Bug Tracker.

Be sure to copy the config-dist.php file to config.php and edit it with the correct values before continuing.

Please remove the install.php file and install/ directory before continuing, as leaving these files poses a security risk!

You are now ready to start using Simplog. Login with
login: admin
pass: admin

Start using Simplog

Powered by Simplog

4 Exploit

If the user is signed in and follows a crafted link (e.g. via email, instant messaging, etc.) from some outside malicious source, their password will automatically be changed. In our example, the user is sent the following link:

```
localhost/simplog/user.php?pass1=newPass&pass2=newPass&blogid=1&act=change
```

Upon following the link, this submits a password change request on behalf of the logged in user in order to set their new password to "newPass". We can see that the password has been updated:
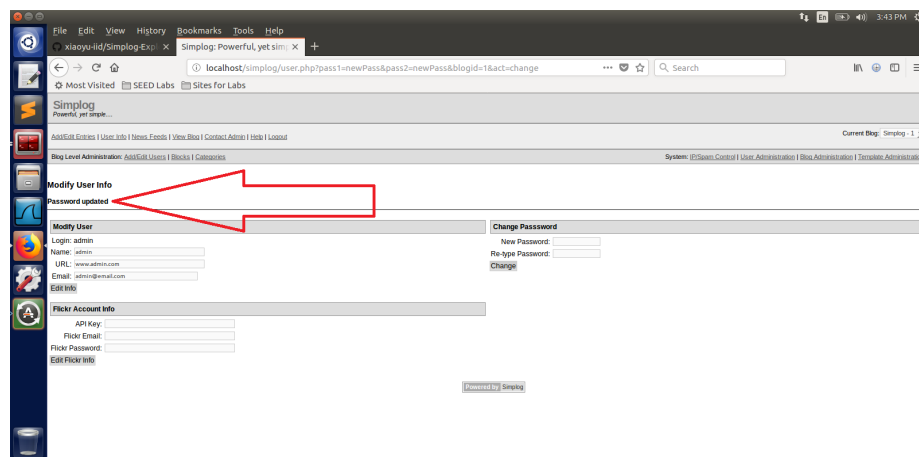
By checking the MySQL database before and after this link has been clicked, we can further prove that the password has changed. Note the different hashed passwords, and that we can even successfully change the password of `admin`:
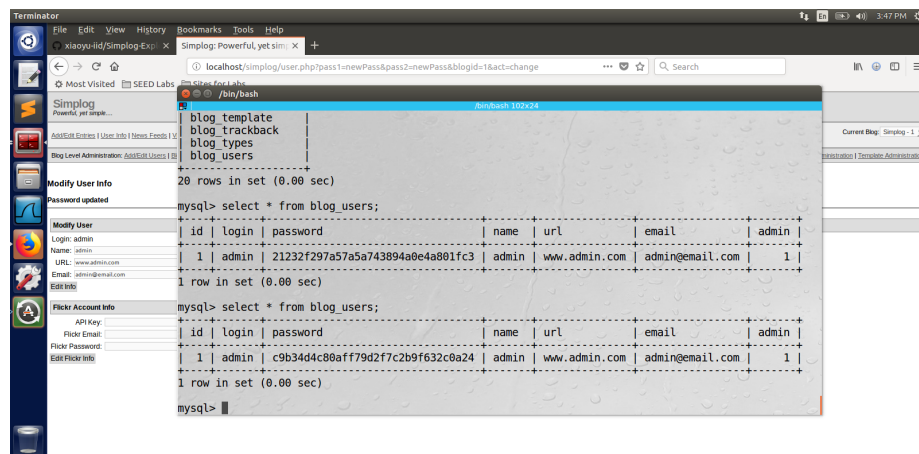
Let's examine the relevant code in `user.php` to see why this works:

```php
} elseif($_REQUEST['act'] == "change") {

    if(($_REQUEST['pass1'] == "") or ($_REQUEST['pass2'] == "") or ($_REQUEST[
        'pass1'] != $_REQUEST['pass2'])) {
        $err = "<font color=red><b>Passwords must match!</b></font><P>";
    } else {
        $enc = md5($_REQUEST['pass1']);
        $sql = "UPDATE blog_users set password='$enc' where login='$_SESSION[
            login]'";
        $res = $db->Execute($sql);
        echo "<b>Password updated</b><br><hr><p>\n";
    }
```

Once `user.php` identifies the action to be "change" ($REQUEST['act'] ==
"change"), it checks that `pass1` and `pass2` are nonempty and match each other
for confirmation. If so, the new password is hashed and a SQL command is ex-
ecuted to update the password of the current blog user. Therefore, when these
arguments are provided in the malicious URI, the user's password is changed
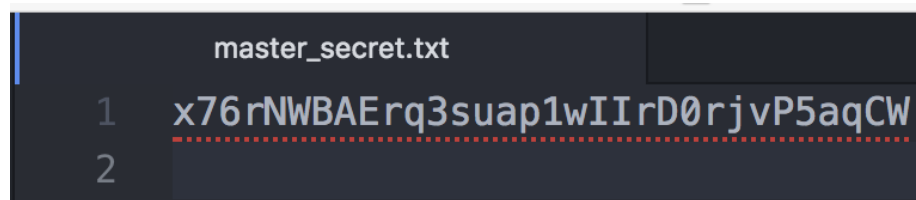without their consent.

5 Patch

In this section, we will demonstrate how the CSRF vulnerability could be prevented with validation tokens. To use our patched version of Simplog, place the `simplog_patched/` directory from Github into `/var/www/simplog` instead.

We first create a file `master_secret.txt` which we fill with the result of a call to the following function:

```
$ head /dev/urandom | tr -dc A-Za-z0-9 | head -c 32
```
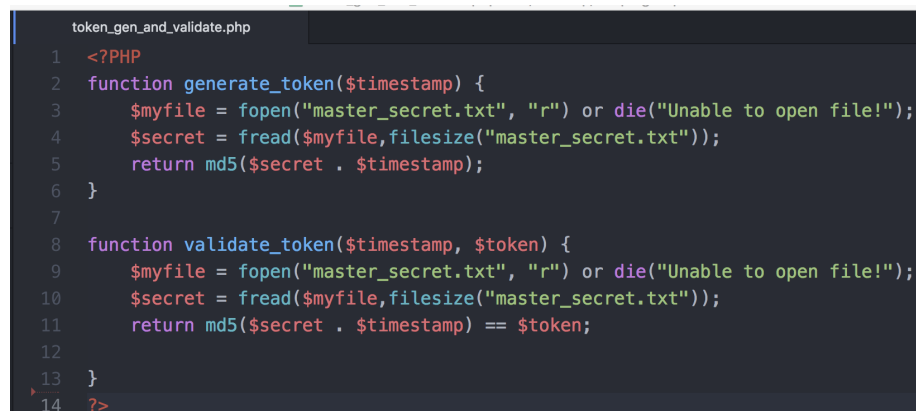
This generates a random string of length 32 using characters within the set of {A-Z, a-z, 0-9}. This master secret will be used in our MD5 hash to create the CSRF token. It should be updated on a regular basis by system administrators to ensure variability in the hashes:

```
master_secret.txt
1    x76rNWBAErq3suap1wIIrD0rjvP5aqCW
2
```

The following functions are for token creation and validation. In the function `generate_token()`, tokens are created to be the MD5 hash value of `master_secret.txt` and `timestamp` concatenated. To provide even additional security, more values from the session could be concatenated here, but we found the master secret and timestamp to be sufficient.

Tokens are validated in the function `validate_token()`. The function recalculates the MD5 hash value of `master_secret.txt` and `timestamp` concatenated, and does an equality comparison between the recalculated token and the `token` passed by the request, returning the boolean result:

```php
     token_gen_and_validate.php
1    <?PHP
2    function generate_token($timestamp) {
3        $myfile = fopen("master_secret.txt", "r") or die("Unable to open file!");
4        $secret = fread($myfile,filesize("master_secret.txt"));
5        return md5($secret . $timestamp);
6    }
7
8    function validate_token($timestamp, $token) {
9        $myfile = fopen("master_secret.txt", "r") or die("Unable to open file!");
10       $secret = fread($myfile,filesize("master_secret.txt"));
11       return md5($secret . $timestamp) == $token;
12
13   }
14   ?>
```

13

In the `login.php` file, we require our new `token_gen_and_validate.php` file. We then make a call to the `time()` function and place this in our timestamp variable `ts`. We call our `generate_token()` function, passing it the timestamp. We then save both the generated token and the timestamp in the `_SESSION` data structure:

```
11    diff simplog/login.php simplog_patched/login.php
12    21a22,23
13    >       require('token_gen_and_validate.php');
14    >
15    23a26,29
16    >       $ts = time();
17    >       $token  = generate_token($ts);
18    >       $mysql = "";
19    >
20    28a35,36
21    >             $_SESSION['token'] = $token;
22    >             $_SESSION['timestamp'] = $ts;
```

In `user.php`, the file `token_gen_and_validate.php` is also required:

```
34    diff simplog/user.php simplog_patched/user.php
35    4,5d3
36    <
37    < require("lib.php");
38    6a5,6
39    > require("lib.php");
40    > require('token_gen_and_validate.php');
41    35c35
42    < } elseif($_REQUEST['act'] == "del") {
43    ---
44    > } elseif($_REQUEST['act'] == "del") {
```

When a request to change the user password is created, `user.php` includes the saved token from `$_SESSION['token']` as a hidden input of the form.

```
83   ---
84   >
85   173a177
86   > <input type=hidden name=token value="<?=$_SESSION['token']?>">
87   180c184
88   < <?php
89   ---
```

Once `user.php` identifies the action to be "change" ($REQUEST['act'] == "change"), we now call the `validate_token()` function to make sure that the CSRF token passed by the request matches the recalculation of the token. If the token matches, the password change proceeds as it did before. If the token does not match however, we add a "BAD TOKEN" line to the HTML:

```
53   48,56c48,59
54   <
55   <   if(($_REQUEST['pass1'] == "") or ($_REQUEST['pass2'] == "") or ($_REQUEST['pass1'] != $_REQUEST['pass2'])) {
56   <       $err = "<font color=red><b>Passwords must match!</b></font><P>";
57   <   } else {
58   <       $enc = md5($_REQUEST['pass1']);
59   <       $sql = "UPDATE blog_users set password='$enc' where login='$_SESSION[login]'";
60   <       $res = $db->Execute($sql);
61   <       echo "<b>Password updated</b><br><hr><p>\n";
62   <   }
63   ---
64   >     if (!validate_token($_SESSION['timestamp'], $_REQUEST['token'])) {
65   >         echo "<b>BAD TOKEN.</b><br>\n";
66   >     } else {
67   >        if(($_REQUEST['pass1'] == "") or ($_REQUEST['pass2'] == "") or ($_REQUEST['pass1'] != $_REQUEST['pass2'])) {
68   >            $err = "<font color=red><b>Passwords must match!</b></font><P>";
69   >        } else {
70   >            $enc = md5($_REQUEST['pass1']);
71   >            $sql = "UPDATE blog_users set password='$enc' where login='$_SESSION[login]'";
72   >            $res = $db->Execute($sql);
73   >            echo "<b>Password updated</b><br><hr><p>\n";
74   >        }
75   >     }
```

Now we test out our changes.

We see that we are still able to change the password in the normal fashion from the change password form as intended:

We can observe that the password hash in our database is changed after a normal password change request sent by the user:

However, we can see that upon following the malicious link, the user is now presented with a "BAD TOKEN" message:

By checking the MySQL database before and after this link has been clicked, we show that the hashed password has not been changed:

Our CSRF token has therefore successfully blocked the attack.

6   Conclusion

Simplog is an old but simple application used to add blogging features to a personal website. However, it requires limited checks and validations for client requests to interact with the server database. As a result, a CSRF vulnerability can be exploited to change user passwords. One simple fix for this vulnerability is to use a CSRF token to validate each client request to change the current password. We expect there to be a number of other endpoints within Simplog that are vulnerable to CSRF attacks. The patches for those exploits should be a straightforward application of the CSRF token solution we have provided in this report.

Disclaimer: Simplog 0.9.3.2 is created by Jeremy Ashcraft (`ashcraft@13monkeys.com`). It is free software, released under GNU GPL Licence version 2.0.

7   References

- https://nvd.nist.gov/vuln/detail/CVE-2009-4092

- https://stackoverflow.com/questions/36788873/package-php5-have-no-installation-candidate-ubuntu-16-04

- https://launchpad.net/ ondrej/+archive/ubuntu/php

- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

- http://www.cis.syr.edu/ wedu/seed/Labs_16.04/Web/Web_CSRF_Elgg/