Pentesterlab Pickle Code Execution

Saikat Karmakar | SEPT 2 : 2021

Introduction

Serialisation of object is used by application to make their storage easier. If an application needs to store an instance of a class, it can use serialisation to get a string representation of this object. When the application needs to use the instance again, it will unserialise the string to get it.

Obviously, if the malicious user can tamper with a string that will be deserialised, they can potentially trigger unexpected behaviour in the application. Depending on the language and library used, this unexpected behaviour can go from arbitrary object creation to remote code execution. This exercise covers the Pickle library used by Python to perform serialisation and deserialisation of objects.

Pickle

The following code is used to create and serialise an instance of the class `Hack` (here with Python 2):

```python
import cPickle
class Hack:
  def __init__(self):
    self.test1 = "test"
    self.test2 = "retest"

h = Hack()
print cPickle.dumps(h)
```

language-python

The line `cPickle.dumps(Hack())` is used to transform the new instance `h` of the class `Hack` into a string.

We can now see what a pickled object looks like:

```bash
$ python test.py
(i__main__
Hack
(dp1
S'test1'
p2
S'test'
p3
sS'test2'
p4
S'retest'
p5
sb.
```

language-bash

Since the format is multi-lines and contain a fair-bit of special characters, it's unlikely that a web application uses it as it is. Most web applications dealing with pickle objects will encode them (for example using base64).

Code execution with Pickle

If an application unserialises data using pickle based on a string under your control, you can execute code in the application. To do so, you will need to create a malicious object. The following example creates an object that will bind a shell on port `1234` and run `/bin/bash`:

```python
class Blah(object):
  def __reduce__(self):
    return (os.system,("netcat -c '/bin/bash -i' -l -p 1234 ",))
```

language-python

We can use the code from the previous section to get the pickled object. Now, we need to find where the application uses Pickled data. By inspecting the application, we can see that the application uses Pickle as part of the `Remember me` function. We can now try to send our malicious object in the `remember_me` cookie. After Base64 encoding the payload, we can then send it to the application to obtain command execution. If we do this, we can see that it doesn't work. By tacking a step back and thinking like the person who developed the application, we can make a guess: the `remember_me` cookie will only be processed if we do not provide a session. Once you send your payload without a session, you should be able to connect to the shell using netcat or telnet.

If you use this payload, you will need to pickle the object on the same platform. Since the vulnerable system is a Linux system you will not be able to send an object pickled on Windows. You can bypass this limitation by using `subprocess` with `__import__`.

After the command gets executed, you should see a HTTP/500 error, this happens because you're sending a malicious object instead of a User object. The application unserialise your malicious object (the code execution happens), then the application tries to use the object but it crashes as your object doesn't have the right methods or attributes.