NOTE: SCENARIO IS WHAT THE PUZZLE TAKER SEES.

SCENARIO:

Imagine you are developing a file transfer application that allows data transmission with a “pause” feature. Its basic implementation consists of two parts: a server-side and a client-side. The server-side creates a socket to receive serialized data from a client, and deserialize it. The client side creates a socket to deserialize data, and send it to a server. (Serialization is the process of translating data structures or object states into formats that can be stored, or transmitted and reconstructed later. When the resulting series of bits is re-read according to the serialization format, it can be used to create a semantically-identical clone of the original object.) In the following code, the method `os.system(command)` executes the command, which is passed as an argument in a subshell. The method `cPickle.loads(data)` reads a pickled object (i.e. one of Python’s native data types) from a byte stream and returns the reconstituted object hierarchy. The method `cPickle.dumps(obj)` returns the pickled representation of the object as a string of bytes. (Note that Pickle is a standard Python library for serialization or deserialization of a Python object structure.)

Consider the snippet of code below and answer the following questions, assuming that the code has all required permissions to execute.

```
#CLIENT-SIDE
#import whatever is needed

class transport(object):
   def __reduce__(self):
       return (os.system, ('ls',))

data = cPickle.dumps(transport())
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('server', 55555)) # connecting to the below server
sock.send(data)
sock.close()
```

```
#SERVER-SIDE
import os, sys, socket, cPickle

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('localhost', 55555))
sock.listen(1)

while True:
  connection, client_address = sock.accept()
  data = connection.recv(1024)
```

```
    data = cPickle.loads(data)
    connection.close()

    # continue to process the data...
```

Questions:
1. What will the program (both server-side and client-side) do when executed?

2. Which of the following will happen on the server-side when the server receives data from the client?

a. The server will throw an error.
b. The server will return the list of files in the directory of the client-side.
c. The server will return the list of files in the directory of the server-side.
d. None of the above.

[Other statistical questions will be imported here while creating the survey.]

NOTE: ANSWER IS TO BE SHOWN TO THE PUZZLE TAKER AT THE END OF THE SESSION.

ANSWER:
1. The server opens up a socket, which will listen for data sent from the client.
2. d
In the above code, the data sent to the server is a command object, which the server will run as a shell command. It will print the list of files in the current directory of the server to the console. Note that nothing is returned to the client.

NOTE: THE REST OF THIS DOCUMENT CONTAINS EXTRA INFORMATION FOR THE PROJECT RESEARCHERS. IT IS NOT TO BE SHOWN TO THE PUZZLE TAKERS.

TAGS:
Python, pickle, code-injection

CATEGORIES:
Blindspot - YES
Type - Injection
Number of distinct functions - 11
Number of total functions - 12
Blindspot function - `cPickle.loads()`
Function call omitted - NO
Blindspot type - Validation missing

Number of parameters in the blindspot function - 0 parameters
Cyclomatic complexity - 3

NAME:
cPickle.load(file)

DESCRIPTION:
The pickle module implements binary protocols for serializing and de-serializing a Python object structure. The `load(file)` method reads a pickled object representation from the open file object file and returns the reconstituted object hierarchy specified therein.

BLINDSPOT:
The `pickle` module is not secure against erroneous or maliciously constructed data. It enables users to craft input that can execute arbitrary code on a remote machine. Never unpickle data received from an untrusted or unauthenticated source.

The blindspot is about the "pickle" module of Python, which provides data (basic) serialization/deserialization functionality. In the context of serialization tools, it is usually the deserialization functionality which may be vulnerable to maliciously constructed data. However, since deserialization cannot exist without the serialization part, its weakness makes the whole module dangerous.

CORRECT USE EXAMPLE:
#N/A

MORE INFORMATION:
#N/A

REFERENCES:
1. http://kevinlondon.com/2015/08/15/dangerous-python-functions-pt2.html