Python-Remote-code-exec

Python Remote code exec

Author: Zhang Junyu

Description

When we use python to develop distributed process projects, it is possible to share code between modules that looks like the following

```
from multiprocessing.managers import BaseManager

class Q():
    pass

class QManager(BaseManager):
    pass

q = Q()
def get_q():
    return q

QManager.register('get_q', callable=get_q)
m = QManager(address=('', 8500), authkey=b'')

s = m.get_server()
s.serve_forever()
```

But I found that pickle was used by default to serialize and deserialize RPC communication, as shown

```
class BaseManager(object):
    '''
    Base class for managers
    '''
    _registry = {}
    _Server = Server

    def __init__(self, address=None, authkey=None, serializer='pickle',
                 ctx=None):
        if authkey is None:
            authkey = process.current_process().authkey
        self._address = address     # XXX not final address if eg ('', 0)
        self._authkey = process.AuthenticationString(authkey)
        self._state = State()
        self._state.value = State.INITIAL
        self._serializer = serializer
```

This is extremely insecure, because when the server accepts the request, pickle.loads will be called for deserialization, which will give the attacker a chance to RCE

The details are shown below

```
245
246     def recv(self):
247         """Receive a (picklable) object"""
248         self._check_closed()
249         self._check_readable()
250         buf = self._recv_bytes()
251         return _ForkingPickler.loads(buf.getbuffer())
252
```

This is the function of ConnectionBase.recv, Here it calls the loads function of the _ForkingPickler module, And this class, as a parent class, is inherited by the others, and the receiving function is completely output from this function

```
class Connection(_ConnectionBase):
    """
    Connection class based on an arbitrary file descriptor (Unix only), or
    a socket handle (Windows).
    """

    if _winapi:
        def _close(self, _close=_multiprocessing.closesocket):
            _close(self._handle)
        _write = _multiprocessing.send
        read = multiprocessing.recv
```

```
class ForkingPickler(pickle.Pickler):
    '''Pickler subclass used by multiprocessing.'''
    _extra_reducers = {}
    _copyreg_dispatch_table = copyreg.dispatch_table

    def __init__(self, *args):
        super().__init__(*args)
        self.dispatch_table = self._copyreg_dispatch_table.copy()
        self.dispatch_table.update(self._extra_reducers)

    @classmethod
    def register(cls, type, reduce):
        '''Register a reduce function for a type.'''
        cls._extra_reducers[type] = reduce

    @classmethod
    def dumps(cls, obj, protocol=None):
        buf = io.BytesIO()
        cls(buf, protocol).dump(obj)
        return buf.getbuffer()

    loads = pickle.loads
```

It ends up calling pickle.loads to deserialize the remote data, which results in RCE

So I also think that Pickle is Forking

> But no one seems to pay attention to this problem, so I hope to apply for a CVE-ID to get the attention, which will be very common when using python to develop distributed tasks.

POC:

poc

demonstration