pymem Documentation

Release alpha

Author

Dec 29, 2020

Contents

1 User's Guide 3

2 API Reference 15

3 Additional Notes 17

Python Module Index 21

Index 23

Welcome to Pymem's documentation. Get started with *Installation* and then get an overview with the *Quickstart*. There is also a more detailed *Tutorials* section that shows how to write small software with Pymem. The rest of the docs describe each component of Pymem in detail, with a full reference in the *API* section.

Except for running tests or buliding the documentation, Pymem does not require any library it only manipulate ctypes and more precisely WinDLL.

The structure of this documentation is based on Flask.

CHAPTER 1

User's Guide

This part of the documentation, which is mostly prose, begins with some background information about Pymem, then focuses on step-by-step instructions for reversing with Pymem.

1.1 Foreword

Read this before you get started with Pymem. This hopefully answers some questions about the purpose and goals of the project, and then why you should and should not be using it.

1.1.1 Why Pymem ?

I decided to build pymem after some reading of the wonderfull book Gray Hat Python by Justin Seitz, which I recommend as a first reading before even starting using Pymem. The book covers the win32api and important aspects of debuggers. As I wanted to learn more on debugging, hooking and the windows API, I figured out that writing a library was the perfect project.

1.1.2 Pymem history

So back in 2010, with my little knowledge of Python I wrote the first version of this library (which has been entirely rewritten since). I figured out that most of the resources you can find covering C, C++, C# of the windows API works "as it" using python ctypes without any effort, so I decided to wrap some of them into Pymem.

In 2015, I decided to rebirth the library, and to rewrite it using python3. The library is a toolbox for process memory manipulations, it supports memory reads, writes and even assembly injection (thanks to pyfasm).

In 2020, the support for pyfasm was dropped because of its incompatibility with x64 processes. It now includes testing, and the documentation as been totally rewritten with tutorials.

1.1.3 Why and when using Pymem

Pymem has been built to reverse games such as Worlf of Warcraft, so if you plan to write a bot for this kind of game, you're in the right place. You can also use pymem to do injections, assembly, memory pattern search and a lot more.

You should head over the *Tutorials* section and see what Pymem is capable of!

Continue to *Installation*, the *Quickstart* or *Tutorials*.

1.2 Installation

Pymem has no dependencies and works on both x86 and x64 architecture.

You will need Python 3 or newer to get started, so be sure to have an up-to-date Python 3.x installation.

If you are familiar with pyenv, it is highly recommended to sandbox pymem installation within a custom virtualenv.

1.2.1 Path

In order to use all pymem fonctionalities you have to first make sure that system python directory is configured within windows system PATH.

In a PowerShell window type:

```
$env:PATH
```

This PATH should contain the directory where python is installed system wide or at least have access to pythonXX.dll If you don't find python in your PATH, then it is recommended to add it.

```
- Open the Start Search, type in "env", and choose "Edit the system environment␣
↪variables"
- Click the "Environment Variables..." button
- Under the "System Variables" section (the lower half), find the row with "Path" in␣
↪the first column, and click edit.
- The "Edit environment variable" UI will appear. Here, you can click "New" and type␣
↪in the new path you want to add.
- Add your python path and close the windows (something like:␣
↪C:\Users\xxx\AppData\Local\Programs\Python\Python38)
```

1.2.2 Virtual environments

Use a virtual environment to manage the dependencies for your project, both in development and in production.

What problem does a virtual environment solve? The more Python projects you have, the more likely it is that you need to work with different versions of Python libraries, or even Python itself. Newer versions of libraries for one project can break compatibility in another project.

Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system's packages.

Python comes bundled with the `venv` module to create virtual environments.

Create an environment

Create a project folder and a `venv` folder within:

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv venv
```

On Windows:

```
$ py -3 -m venv venv
```

Activate the environment

Before you work on your project, activate the corresponding environment:

```
$ . venv/bin/activate
```

On Windows:

```
> venv\Scripts\activate
```

Your shell prompt will change to show the name of the activated environment.

1.2.3 Install Pymem

Within the activated environment, use the following command to install Pymem:

```
$ pip install pymem
```

Pymem is now installed. Check out the *Quickstart* or go to the *Documentation Overview*.

1.3 Quickstart

Eager to get started? This page gives a good introduction to Pymem. Follow *Installation* to set up a project and install Pymem first.

1.3.1 A Minimal Application

A minimal Pymem application looks something like this:

```
from pymem import Pymem

pm = Pymem('notepad.exe')
print('Process id: %s' % pm.process_id)
address = pm.allocate(10)
print('Allocated address: %s' % address)
pm.write_int(address, 1337)
value = pm.read_int(address)
print('Allocated value: %s' % value)
pm.free(address)
```

So what did that code do?

1. First we imported the `Pymem` class. An instance of this class will be our win32api wrapper
2. Next we create an instance of this class. The first argument is the name of the windows process we want to hook into.

   Be aware that after creating an instance of Pymem with the process name as an argument, the process will be opened with debug mode flags.
3. We then allocate 10 bytes into given _notepad.exe_ process with `allocate()`.
4. For the example we then write an integer with `write_int()` and read it with `read_int()`.
5. We then free memory from the current opened process at the given address with `free()`.

Save it as `hello.py` or something similar. Make sure to not call your application `pymem.py` because this would conflict with Pymem itself.

To run the application, first start *notepad.exe* be sure to have pymem installed within your current python environment and simply execute your script.

```
$ python hello.py
  Process id: 2345
  Allocated address: 123456789
  Allocated value: 1337
```

1.4 Tutorials

1.4.1 Listing process modules

Pymem comes with somes process utilities like listing loaded modules.

Here is a snippet that will list loaded process modules

```
import pymem

pm = pymem.Pymem('python.exe')
modules = list(pm.list_modules())
for module in modules:
    print(module.name)
```

So what did that code do?

1. we hook pymem with python.exe process
2. we retrieve the list of loaded modules
3. for every module listed, we display its name

note: every *module* is an instance of `MODULEINFO()`

1.4.2 Injecting a python interpreter into any process

Pymem allow you to inject *python.dll* into a target process and then map *py_run_simple_string* with a single call to `inject_python_interpreter()`.

```
from pymem import Pymem

notepad = subprocess.Popen(['notepad.exe'])

pm = pymem.Pymem('notepad.exe')
pm.inject_python_interpreter()
filepath = os.path.join(os.path.abspath('.'), 'pymem_injection.txt')
filepath = filepath.replace("\\", "\\\\")
shellcode = """
f = open("{}", "w+")
f.write("pymem_injection")
f.close()
""".format(filepath)
pm.inject_python_shellcode(shellcode)
notepad.kill()
```

So what did that code do?

1. we start notepad process and get its handle
2. we hook pymem with notepad process
3. we call `inject_python_interpreter()` which will:
   - dynamically finds the correct python dll and inject it
   - register **py_run_simple_string**
4. then we inject some python code with `inject_python_shellcode()` which will:
   - **VirtualAllocEx** some space for the code to be written
   - write the actual payload into allocated space
   - execute **py_run_simple_string** so the python code gets interpreted within the notepad process
5. finally we get rid of notepad process

1.5 Examples from the community

No support will be provided for any community related examples.

Here is a list of programs / scripts made by the community:

1.5.1 External glow ESP for CS:GO

No support will be provided for any community related examples.

Credits goes to Snacky.

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with VAC.

Use this code at your own risk and be aware that **using any sort of hack will resolve in having your steam_id banned**

Snippet

```
import pymem
import pymem.process

dwEntityList = (0x4D4B104)
dwGlowObjectManager = (0x5292F20)
m_iGlowIndex = (0xA428)
m_iTeamNum = (0xF4)


def main():
    print("Diamond has launched.")
    pm = pymem.Pymem("csgo.exe")
    client = pymem.process.module_from_name(pm.process_handle, "client.dll").
↪lpBaseOfDll

    while True:
        glow_manager = pm.read_int(client + dwGlowObjectManager)

        for i in range(1, 32):  # Entities 1-32 are reserved for players.
            entity = pm.read_int(client + dwEntityList + i * 0x10)

            if entity:
                entity_team_id = pm.read_int(entity + m_iTeamNum)
                entity_glow = pm.read_int(entity + m_iGlowIndex)

                if entity_team_id == 2:  # Terrorist
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x4, float(1))␣
↪  # R
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x8, float(0))␣
↪  # G
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0xC, float(0))␣
↪  # B
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x10,␣
↪float(1))  # Alpha
                    pm.write_int(glow_manager + entity_glow * 0x38 + 0x24, 1)        ␣
↪  # Enable glow

                elif entity_team_id == 3:  # Counter-terrorist
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x4, float(0))␣
↪  # R
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x8, float(0))␣
↪  # G
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0xC, float(1))␣
↪  # B
                    pm.write_float(glow_manager + entity_glow * 0x38 + 0x10,␣
↪float(1))  # Alpha
                    pm.write_int(glow_manager + entity_glow * 0x38 + 0x24, 1)        ␣
↪  # Enable glow


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

1.5.2 AssaultCube External ESP (Pygame, Pymem)

No support will be provided for any community related examples.

Meow, feel free to contact pymem author if you want this example to be removed as we didn't reached you to have your authorisation to post this

Credits goes to Meow.

Original source code: link

Warning

This comes, "as it" with no guarantees regarding its standing with any anti-cheat related.

Use this code at your own risk and be aware that **using any sort of hack may resolve in having your account banned**

Snippet

See **Original source code** above to see the whole source code.

Video

1.5.3 CS:GO Esp

No support will be provided for any community related examples.

pSilent, feel free to contact pymem author if you want this example to be removed as we didn't reached you to have your authorisation to post this

Credits goes to pSilent

Original post: link

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with any anti-cheat related.

Use this code at your own risk and be aware that **using any sort of hack may resolve in having your account banned**

Snippet

See **Original source code** above to see the whole source code.

Screenshot

1.5.4 No flash cheat for CS:GO

No support will be provided for any community related examples.

Credits goes to Snacky.

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with VAC.

Use this code at your own risk and be aware that **using any sort of hack will resolve in having your steam_id banned**

Snippet

```
import pymem
import pymem.process
import time

dwLocalPlayer = (0xD36B94)
m_flFlashMaxAlpha = (0xA40C)


def main():
    print("Emerald has launched.")
    pm = pymem.Pymem("csgo.exe")
    client = pymem.process.module_from_name(pm.process_handle, "client.dll").
↪lpBaseOfDll

    while True:
        player = pm.read_int(client + dwLocalPlayer)
        if player:
            flash_value = player + m_flFlashMaxAlpha
            if flash_value:
                pm.write_float(flash_value, float(0))
        time.sleep(1)
```

(continues on next page)

(continued from previous page)

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

1.5.5 Auto bunny hopper for CS:GO

No support will be provided for any community related examples.

Credits goes to Snacky.

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with VAC.

Use this code at your own risk and be aware that **using any sort of hack will resolve in having your steam_id banned**

Snippet

```
import keyboard
import pymem
import pymem.process
import time
from win32gui import GetWindowText, GetForegroundWindow

dwForceJump = (0x51F4D88)
dwLocalPlayer = (0xD36B94)
m_fFlags = (0x104)


def main():
    print("Ruby has launched.")
    pm = pymem.Pymem("csgo.exe")
    client = pymem.process.module_from_name(pm.process_handle, "client.dll").
↪lpBaseOfDll

    while True:
        if not GetWindowText(GetForegroundWindow()) == "Counter-Strike: Global␣
↪Offensive":
            continue

        if keyboard.is_pressed("space"):
            force_jump = client + dwForceJump
            player = pm.read_int(client + dwLocalPlayer)
            if player:
                on_ground = pm.read_int(player + m_fFlags)
                if on_ground and on_ground == 257:
                    pm.write_int(force_jump, 5)
                    time.sleep(0.08)
                    pm.write_int(force_jump, 4)
```

(continues on next page)

(continued from previous page)

```
        time.sleep(0.002)


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

1.5.6 Trigger bot for CS:GO

No support will be provided for any community related examples.

Credits goes to Snacky.

Original source code: github

Warning

This comes, "as it" with no guarantees regarding its standing with VAC.

Use this code at your own risk and be aware that **using any sort of hack will resolve in having your steam_id banned**

Snippet

```
import keyboard
import pymem
import pymem.process
import time
from win32gui import GetWindowText, GetForegroundWindow

dwEntityList = (0x4D4B104)
dwForceAttack = (0x317C6EC)
dwLocalPlayer = (0xD36B94)
m_fFlags = (0x104)
m_iCrosshairId = (0xB3D4)
m_iTeamNum = (0xF4)

trigger_key = "shift"


def main():
    print("Sapphire has launched.")
    pm = pymem.Pymem("csgo.exe")
    client = pymem.process.module_from_name(pm.process_handle, "client.dll").
↪lpBaseOfDll

    while True:
        if not keyboard.is_pressed(trigger_key):
            time.sleep(0.1)

        if not GetWindowText(GetForegroundWindow()) == "Counter-Strike: Global␣
↪Offensive":
            continue

        if keyboard.is_pressed(trigger_key):
```

(continues on next page)

(continued from previous page)

```
            player = pm.read_int(client + dwLocalPlayer)
            entity_id = pm.read_int(player + m_iCrosshairId)
            entity = pm.read_int(client + dwEntityList + (entity_id - 1) * 0x10)

            entity_team = pm.read_int(entity + m_iTeamNum)
            player_team = pm.read_int(player + m_iTeamNum)

            if entity_id > 0 and entity_id <= 64 and player_team != entity_team:
                pm.write_int(client + dwForceAttack, 6)

            time.sleep(0.006)


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

1.6 Common issues

Here is a non-exhaustive list of common issues related to pymem usage:

- The token does not have the specified privilege

Pymem requires that the terminal or interpreter be ran with administrator privileges

- AttributeError: 'NoneType' object has no attribute or any other Python error

Make sure you are running at least Python 3.5 (earlier versions are unsupported, future versions may have breaking issues)

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API

This part of the documentation covers all the methods of Pymem. For parts where Pymem depends on external dlls, we document the most important right here and provide links to the canonical documentation.

2.1.1 Pymem

2.1.2 Structures

2.1.3 Pattern

2.1.4 Exceptions

CHAPTER 3

Additional Notes

Design notes, legal information and changelog are here for the interested.

3.1 License

3.2 How to contribute to Pymem

Thank you for considering contributing to Pymem!

3.2.1 Support questions

Please, don't use the issue tracker for this. The issue tracker is a tool to address bugs and feature requests in Pymem itself. Use one of the following resources for questions about using Pymem or issues with your own code:

- The `#general` channel on our Discord chat: https://discord.gg/xaWNac8
- Ask on Stack Overflow. Search with Google first using: `site:stackoverflow.com python pymem {search term, exception message, etc.}`

3.2.2 Reporting issues

Include the following information in your post:

- Describe what you expected to happen.
- If possible, include a minimal reproducible example to help us identify the issue. This also helps check that the issue is not with your own code.
- Describe what actually happened. Include the full traceback if there was an exception.
- List your Python, Pymem versions. If possible, check if this issue is already fixed in the latest releases or the latest code in the repository.

3.2.3 Submitting patches

If there is not an open issue for what you want to submit, prefer opening one for discussion before working on a PR. You can work on any issue that doesn't have an open PR linked to it or a maintainer assigned to it. These show up in the sidebar. No need to ask if you can work on an issue that interests you.

Include the following in your patch:

- Include tests if your patch adds or changes code. Make sure the test fails without your patch.
- Update any relevant docs pages and docstrings.

First time setup

- Download and install the latest version of git.
- Configure git with your username and email.

```
$ git config --global user.name 'your name'
$ git config --global user.email 'your email'
```

- Make sure you have a GitHub account.
- Fork Pymem to your GitHub account by clicking the Fork button.
- Clone the main repository locally.

```
$ git clone https://github.com/srounet/pymem
$ cd pymem
```

- Add your fork as a remote to push your work to. Replace `{username}` with your username. This names the remote "fork", the default Pymem remote is "origin".

```
git remote add fork https://github.com/{username}/pymem
```

- Create a virtualenv.

```
$ python3 -m venv env
$ . env/bin/activate
```

On Windows, activating is different.

```
> env\Scripts\activate
```

- Install Pymem in editable mode with development dependencies.

```
$ pip install -e .
```

Start coding

- Create a branch to identify the issue you would like to work on. If you're submitting a bug or documentation fix, branch off of the latest ".x" branch.

```
$ git fetch origin
$ git checkout -b your-branch-name origin/1.1.x
```

If you're submitting a feature addition or change, branch off of the "master" branch.

```
$ git fetch origin
$ git checkout -b your-branch-name origin/master
```

- Using your favorite editor, make your changes, committing as you go.
- Include tests that cover any code changes you make. Make sure the test fails without your patch. Run the tests as described below.
- Push your commits to your fork on GitHub and create a pull request. Link to the issue being addressed with `fixes #123` in the pull request.

```
$ git push --set-upstream fork your-branch-name
```

Running the tests

Run the basic test suite with pytest.

```
$ python -m pytest
```

This runs the tests for the current environment, which is usually sufficient. CI will run the full suite when you submit your pull request.

Running test coverage

Generating a report of lines that do not have test coverage can indicate where to start contributing. Run `pytest` using `coverage` and generate a report.

```
$ pip install -r requirements-test.txt
$ python -m pytest --cov=pymem
```

Building the docs

Build the docs in the `docs` directory using Sphinx.

```
$ cd docs/source
$ make clean
$ make html
```

Open `_build/html/index.html` in your browser to view the docs.

Read more about Sphinx.

Python Module Index

p

pymem, 15

Index

P

pymem (*module*), 15