Pyrasite Documentation

Release 2.0beta

Luke Macken

January 27, 2017

Pyrasite is a library and a set of tools for injecting code into running Python programs.

homepage http://pyrasite.com

download http://pypi.python.org/pypi/pyrasite

source http://github.com/lmacken/pyrasite

mailing list https://fedorahosted.org/mailman/listinfo/pyrasite

jenkins http://ci.csh.rit.edu/view/Pyrasite

irc #pyrasite on Freenode

CHAPTER 1

Contents

1.1 Installing

1.1.1 Requirements

- gdb (version 7.3+)

1.1.2 Python Compatiblity

Pyrasite works with Python 2.4 and newer. Injection works between versions as well, so you can run Pyrasite under Python 3 and inject into 2, and vice versa.

1.1.3 Installing

You can download the latest tarballs, RPMs, and debs from PyPi. Installing the package specific to your distribution is recommended. However, you can also install it using `pip` if you wish

```
pip install pyrasite pyrasite-gui
```

See also:

pyrasite-gui for instructions on installing the graphical interface

1.1.4 Additional installation notes

Fedora

If you're using Fedora 17 or later, you'll need to disable an SELinux boolean to allow ptrace.

```
sudo setsebool -P deny_ptrace=off
```

Mac OS X

If you don't want to override Apple's default gdb, install the latest version of gdb with a prefix (e.g. gnu)

```
$ ./configure --program-prefix=gnu
$ pyrasite <PID> payloads/reverse_python_shell.py --prefix="gnu"
```

Arch Linux

You can install pyrasite from the Arch User Repository If you want python debugging symbols, you may have to self compile python2.

Ubuntu

Since version 10.10, Ubuntu ships with a controversial patch that restricts the scope of ptrace, which can be disabled by running:

```
echo 0 > /proc/sys/kernel/yama/ptrace_scope
```

You can make this change permanent by setting `ptrace_scope` to `0` in `/etc/sysctl.d/10-ptrace.conf`.

1.2 `pyrasite` - Inject arbitrary code into a running Python process

```
usage: pyrasite [-h] [--gdb-prefix GDB_PREFIX] [--verbose] pid [filename]

pyrasite - inject code into a running python process

positional arguments:
  pid                   The ID of the process to inject code into
  filename              The second argument must be a filename

optional arguments:
  -h, --help            show this help message and exit
  --gdb-prefix GDB_PREFIX
                        GDB prefix (if specified during installation)
  --verbose             Verbose mode

For updates, visit https://github.com/lmacken/pyrasite
```

See also:

Example Payloads

1.3 `pyrasite-shell` - Give it a pid, get a shell

You can easily drop into a shell and execute commands in a running process using the `pyrasite-shell`.

```
$ pyrasite-shell
Usage: pyrasite-shell <PID>
```

```
$ pyrasite-shell $(pgrep -f "ipython")
Pyrasite Shell 2.0beta9
Connected to 'ipython'
Python 2.7.2 (default, Oct 27 2011, 01:40:22)
[GCC 4.6.1 20111003 (Red Hat 4.6.1-10)] on linux2
>>> print(x)
foo

>>> globals()['x'] = 'bar'
```

1.3.1 Source

```
import sys
import pyrasite


def shell():
    """Open a Python shell in a running process"""

    if not len(sys.argv) == 2:
        print("Usage: pyrasite-shell <PID>")
        sys.exit(1)

    ipc = pyrasite.PyrasiteIPC(int(sys.argv[1]), 'ReversePythonShell')
    ipc.connect()

    print("Pyrasite Shell %s" % pyrasite.__version__)
    print("Connected to '%s'" % ipc.title)

    prompt, payload = ipc.recv().split('\n', 1)
    print(payload)

    try:
        import readline
    except ImportError:
        pass

    # py3k compat
    try:
        input_ = raw_input
    except NameError:
        input_ = input

    try:
        while True:
            try:
                input_line = input_(prompt)
            except EOFError:
                input_line = 'exit()'
                print('')
            except KeyboardInterrupt:
                input_line = 'None'
                print('')

            ipc.send(input_line)
            payload = ipc.recv()
            if payload is None:
                break
            prompt, payload = payload.split('\n', 1)
            if payload != '':
                print(payload)
    except:
        print('')
        raise

    ipc.close()
```

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

1.4 pyrasite-memory-viewer - View the largest objects in your process

Pyrasite provides a tool to view object memory usage statistics, and the live value, of largest objects in your process. This requires urwid and meliae to be installed.

```
$ pyrasite-memory-viewer <PID>
```

This tool automatically injects the following payload:

```
import os, meliae.scanner
meliae.scanner.dump_all_objects('/tmp/pyrasite-%d-objects.json' % os.getpid())
```

You can easily dump the object memory usage JSON data by hand, if you wish:

```
$ pyrasite <PID> pyrasite/payloads/dump_memory.py
```

1.5 pyrasite-gui - A graphical interface for Pyrasite

The pyrasite-gui is a graphical interface for Pyrasite that lets you easily monitor, analyze, introspect, and alter running Python programs.

source https://github.com/lmacken/pyrasite-gui

download http://pypi.python.org/pypi/pyrasite-gui

1.5.1 Requirements

- Python debuginfo (needed for live object inspection)
- PyGObject3 Introspection bindings
- WebKitGTK3
- meliae (easy_install/pip may not work for this install. If not, use the tarball from the distribution website. You may need to install Cython in order to get meliae to build)
- pycallgraph
- psutil

Distribution-specific instructions

```
# Fedora
yum --enablerepo=updates-testing install python-psutil python-debuginfo python-pycallgraph pygobject3

# Ubuntu:
apt-get install python-dbg python-pycallgraph python-gobject-dev gir1.2-webkit-3.0 python-meliae pyth
```

```
# Arch
pacman -S python2-psutil python2-gobject python2-pycallgraph libwebkit3 python2-meliae
```

1.5.2 Screenshots

1.6 Example Payloads

These payloads can be found in the pyrasite/payloads directory.

1.6.1 Dumping thread stacks

```
import sys, traceback

for thread, frame in sys._current_frames().items():
    print('Thread 0x%x' % thread)
    traceback.print_stack(frame)
    print()
```

1.6.2 Viewing loaded modules

```
import sys

for name in sorted(sys.modules):
    print('%s: %s' % (name, sys.modules[name]))
```

1.6.3 Call Graph

Pyrasite comes with a payload that generates an image of your processes call graph using pycallgraph.

```
import pycallgraph
pycallgraph.start_trace()
```

```
import pycallgraph
pycallgraph.make_dot_graph('callgraph.png')
```

The callgraph is then generated using graphviz and saved to *callgraph.png*. You can see an example callgraph here.

1.6.4 Forcing garbage collection

```
import gc
gc.collect()
```

1.6.5 Dumping out object memory usage statistics

```
# "meliae" provides a way to dump python memory usage information to a JSON
# disk format, which can then be parsed into useful things like graph
# representations.
#
# https://launchpad.net/meliae
# http://jam-bazaar.blogspot.com/2009/11/memory-debugging-with-meliae.html

import os, meliae.scanner
meliae.scanner.dump_all_objects('/tmp/pyrasite-%d-objects.json' % os.getpid())
```

See also:

pyrasite-memory-viewer - View the largest objects in your process

1.6.6 Reverse Subprocess Shell

```
import pyrasite

class ReverseShell(pyrasite.ReverseConnection):

    reliable = False # This payload is designed to be used with netcat
    port = 9001

    def on_connect(self):
        uname = pyrasite.utils.run('uname -a')[1]
        self.send("%sType 'quit' to exit\n%% " % uname)

    def on_command(self, cmd):
        p, out, err = pyrasite.utils.run(cmd)
        if err:
            out += err
        self.send(out + '\n% ')
        return True

ReverseShell().start()
```

```
$ pyrasite <PID> pyrasite/payloads/reverse_shell.py
```

```
$ nc -l 9001
Linux tomservo 2.6.40.3-0.fc15.x86_64 #1 SMP Tue Aug 16 04:10:59 UTC 2011 x86_64 x86_64 x86_64 GNU/Li
% ls
```

1.6.7 Reverse Python Shell

Deprecated since version 2.0: Use the pyrasite-shell instead

This lets you easily introspect or alter any objects in your running process.

```
import sys
import pyrasite

class ReversePythonShell(pyrasite.ReversePythonConnection):
    port = 9001
    reliable = False
```

```
    def on_connect(self):
        self.send("Python %s\nType 'quit' to exit\n>>> " % sys.version)

ReversePythonShell().start()
```

```
$ python
>>> x = 'foo'
```

```
$ pyrasite <PID> pyrasite/payloads/reverse_python_shell.py
```

```
$ nc -l 9001
Python 2.7.1 (r271:86832, Apr 12 2011, 16:15:16)
[GCC 4.6.0 20110331 (Red Hat 4.6.0-2)]
>>> print x
foo
>>> globals()['x'] = 'bar'
```

1.7 API

pyrasite.**inject**(*pid*, *filename*, *verbose=False*, *gdb_prefix=''*)
Executes a file in a running Python process.

pyrasite.**inspect**(*pid*, *address*)
Return the value of an object in a given process at the specified address

class pyrasite.**PyrasiteIPC**(*pid*, *reverse='ReversePythonConnection'*)
Pyrasite Inter-Python Communication.

This object is used in communicating to or from another Python process.

It can perform a variety of tasks:

- Injection of the *pyrasite.ReversePythonConnection* payload via `PyrasiteIPC.connect()`, which causes the process to connect back to a port that we are listening on. The connection with the process is then available via *self.sock*.
- Python code can then be executed in the process using `PyrasiteIPC.cmd()`. Both stdout and stderr are returned.
- Low-level communication with the process, both reliably (via a length header) or unreliably (raw data, ideal for use with netcat) with a *pyrasite.ReversePythonConnection* payload, via `PyrasiteIPC.send(data)()` and `PyrasiteIPC.recv(data)()`.

The `PyrasiteIPC` is subclassed by `pyrasite.tools.gui.Process` as well as `pyrasite.reverse.ReverseConnection`.

cmd(*cmd*)
Send a python command to exec in the process and return the output

connect()
Setup a communication socket with the process by injecting a reverse subshell and having it connect back to us.

create_payload()
Write out a reverse python connection payload with a custom port

inject()
Inject the payload into the process.

listen()
Listen on a random port

recv()
Receive a command from a given socket

recv_bytes(*n*)
Receive n bytes from a socket

send(*data*)
Send arbitrary data to the process via self.sock

wait()
Wait for the injected payload to connect back to us

class pyrasite.**ReverseConnection**(*host=None*, *port=None*)
A payload that connects to a given host:port and receives commands

on_command(*cmd*)
Called when the host sends us a command

on_connect()
Called when we successfuly connect to *self.host*

class pyrasite.**ReversePythonConnection**(*host=None*, *port=None*)
A reverse Python connection payload.

Executes Python commands and returns the output.

1.8 Development

1.8.1 Running from git

```
git clone git://github.com/lmacken/pyrasite.git
cd pyrasite
python pyrasite/main.py
```

```
git clone git://github.com/lmacken/pyrasite-gui.git
cd pyrasite-gui
python pyrasite_gui/gui.py
```

1.8.2 Git flow

Use git-flow.

1.8.3 Style

- PEP8

C

I

L

O

P

R

S

W