What is pwntools?

It is a set of security libraries allowing to:

- connect to a distant server;
- read and send information;
- manipulate bits, hex, strings easily;
- access command-line arguments easily;
- do many security-related complex things (interact with gdb, create shellcodes, etc.)

Installation

Start checking if you already have pwntools:

```
$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
```

if it does not raise an error you have it. If there is an error you can install it with:

```
$ sudo -H pip3 --proxy=https://proxy.isae.fr:3128 install 'pwntools>=4.0.0beta0'
```

removing the proxy option if you are not inside ISAE.

By the way, pwntools is not installed by default in the computer executing this notebook. Execute the next cell to install it(no need of sudo or proxy in this notebook).

```
!pip3 install 'pwntools>=4.0.0beta0'
```

```
Collecting pwntools>=4.0.0beta0
  Downloading https://files.pythonhosted.org/packages/7e/a2/b8a5eb3d57cd7b
     |████████████████████████████████| 10.0MB 9.9MB/s
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/
Collecting pyelftools>=0.2.4
  Downloading https://files.pythonhosted.org/packages/6f/50/3d7729d64bb233
     |████████████████████████████████| 153kB 48.7MB/s
Collecting pyserial>=2.7
  Downloading https://files.pythonhosted.org/packages/07/bc/587a445451b253
     |████████████████████████████████| 92kB 6.7MB/s
```

```
Collecting unicorn<1.0.2rc4,>=1.0.2rc1
  Downloading https://files.pythonhosted.org/packages/4a/75/a833bcda9a1bd3
     |████████████████████████████████| 8.1MB 28.8MB/s
Requirement already satisfied: pygments>=2.0 in /usr/local/lib/python3.6/d
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.6/dis
Collecting mako>=1.0.0
  Downloading https://files.pythonhosted.org/packages/5c/db/2d2d88b924aa46
     |████████████████████████████████| 481kB 51.1MB/s
Requirement already satisfied: requests>=2.0 in /usr/local/lib/python3.6/d
Collecting paramiko>=1.15.2
  Downloading https://files.pythonhosted.org/packages/95/19/124e9287b43e6f
     |████████████████████████████████| 215kB 68.2MB/s
Collecting capstone>=3.0.5rc2
  Downloading https://files.pythonhosted.org/packages/38/85/647d512c2c2e29
     |████████████████████████████████| 2.1MB 31.2MB/s
Requirement already satisfied: pip>=6.0.8 in /usr/local/lib/python3.6/dist
Requirement already satisfied: pysocks in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/di
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3
Collecting cryptography>=2.5
  Downloading https://files.pythonhosted.org/packages/c9/de/7054df0620b541
     |████████████████████████████████| 2.6MB 32.1MB/s
Collecting bcrypt>=3.1.3
  Downloading https://files.pythonhosted.org/packages/26/70/6d218afbe4c735
     |████████████████████████████████| 71kB 6.8MB/s
Collecting pynacl>=1.0.1
  Downloading https://files.pythonhosted.org/packages/9d/57/2f5e6226a674b2
     |████████████████████████████████| 962kB 49.1MB/s
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.6/dist
Requirement already satisfied: pycparser in /usr/local/lib/python3.6/dist-
Building wheels for collected packages: intervaltree, mako
  Building wheel for intervaltree (setup.py) ... done
  Created wheel for intervaltree: filename=intervaltree-3.1.0-py2.py3-none
  Stored in directory: /root/.cache/pip/wheels/f3/f2/66/e9c30d3e9499e65ea2
  Building wheel for mako (setup.py) ... done
  Created wheel for mako: filename=Mako-1.1.4-py2.py3-none-any.whl size=75
  Stored in directory: /root/.cache/pip/wheels/ad/10/d3/aeb26e20d19045e2a6
```

Now we can import the pwntools functions without error. As pwntools is supposed to be executed in a terminal and not on a Jupyter notebook, we must first define an environment variable before importing the tools in pwntools.

```
import os; os.environ['PWNLIB_NOTERM']="True"
```

```
from pwn import *
```

```
from pwn import *
r = remote('google.com', 443, ssl=True)
# HTML hint for next line.
# The \r\n\r\n is just what HTML requires to understand that you have ended your
# command and not doing a newline. So basically you ask "GET /" and say "over".
r.send("GET /\r\n\r\n")
reply = r.readline()
print(f"Server reply: {reply}")
r.close()
```

```
[x] Opening connection to google.com on port 443
[x] Opening connection to google.com on port 443: Trying 74.125.20.139
[+] Opening connection to google.com on port 443: Done
Server reply: b'HTTP/1.0 200 OK\r\n'
[*] Closed connection to google.com port 443
```

The remote function opens a TCP connexion with a server (first argument) on a given port (second argument).

Some important remarks:

- The `ssl=True` option will add to the TCP negociation (which opens the connection) an SSL negociation (which secures the TCP connection), before the message exchanges start. This happens in the previous example as the Google webpage is accessible in HTTPS only (which is just HTTP messages inside an SSL-secured connectoin).
- To interact with a remote the `sendline` and `readline` functions are used in general. The input of `sendline` is a bytearray to which a newline is added before being sent to the remote. The function `readline` retrieves a bytearray up to the first newline character and returns it (without removing this newline character).
- HTTP being punctilious with newlines and carriage returns, the example uses the function `send` with sends the input bytearray as is.
- For `send` and for `sendline` if you use a string as input, the function will encode it as a bytearray before sending it.

Beware of the bytearray/string conversions

Let's expand a little bit the example above to focus a bit on the reply format.

```
[x] Opening connection to google.com on port 443: Trying 74.125.133.139
[+] Opening connection to google.com on port 443: Done
Server reply: b'HTTP/1.0 200 OK\r\n'
Server reply as a string: HTTP/1.0 200 OK

[*] Closed connection to google.com port 443
We are in the second if
```

This example shows that:

- The received bytearray can be decoded to a string using the `decode()` function to obtain a string
- We can use on a decoded bytearray, usual (and useful) string functions such as `startswith()`

Note: You should not try to build strings from bytearrays in any other way than with `decode()` (nor transform strings to bytearray with anything else than `encode()`). If you do, you will run into problems.

Beware of newlines

Le's show now a small issue. Note that in the following code

```
# Suppose we have a remote r
myline=r.readline()
r.sendline(myline)
```

we are not sending what we received. This is simply because `readline` keeps the final newline in the line read and `sendline` adds a second newline after it. So if `myline == b'Hello\n'` then we are sending `b'Hello\n\n'`.

```
# ./mypythonscript.py REMOTE=agivenremote PORT=agivenport

# We can allow script calls with default values
if not args.REMOTE: args.REMOTE='1.1.1.1'
if not args.PORT: args.PORT=80

try:
  rem = remote(args.REMOTE, args.PORT)
except:
  print('Remote connection failed!')

rem.send("GET /\r\n\r\n")
print(f'Server reply: {rem.recvall()}')
rem.close()
""")
  f.close()
```

Let's now execute the script, with or without command-line arguments.

```
!python3 /tmp/mypythonscript.py
```

```
    [x] Opening connection to 1.1.1.1 on port 80
    [x] Opening connection to 1.1.1.1 on port 80: Trying 1.1.1.1
    [+] Opening connection to 1.1.1.1 on port 80: Done
    [x] Receiving all data
    [x] Receiving all data: 0B
    [x] Receiving all data: 155B
```

Parse module

Installation

Before using it we first need to install the parse module.

```
!pip3 install parse
```

```
Collecting parse
  Downloading https://files.pythonhosted.org/packages/b8/49/85f19d9ff90881
Building wheels for collected packages: parse
  Building wheel for parse (setup.py) ... done
  Created wheel for parse: filename=parse-1.18.0-cp36-none-any.whl size=24
  Stored in directory: /root/.cache/pip/wheels/2a/53/09/869ca5781ede342254
Successfully built parse
Installing collected packages: parse
Successfully installed parse-1.18.0
```

- Note that, when parsing, the string `<html>{}</html>` must match entirely the string `reply_as_a_string`. In particular if there is something before `<html>` or after `</html>` in `reply_as_a_string` (e.g. a newline) it will not match.
- We define `reply_as_a_string` by first calling `strip()` to remove newlines that would prevent the format used in the parse function to match the result.
- We then call `decode()` to get a string out of the stripped bytearray.
- When the parsing fails no exception is lifted but the return value will be `None`, the check `if not parsed:` is important after each parsing to detect such events.
- The parsing results are in a list, as here we only have one value parsed we get it with `parsed[0]`. There can be more than a value parsed as it will be shown in the next example.
- In order to remove useless carriage return and newlines around the title and to print a single-line bytearray instead of a multiline string we print

Wildcards

Unfortunately there is no wildcards/regular expressions in the parse module. Something like `parse.parse('.*<head>{}</head>.*', mystring)` will not work. We can circumvent this problem by adding parsed variables that won't be used.

```
from pwn import *
```