

---

# **pwny Documentation**

***Release 0.7.0***

**Author**

March 07, 2016



<b>1</b>	<b>pwny package</b>	<b>3</b>
<b>2</b>	<b>pwnypack package</b>	<b>5</b>
2.1	pwnypack.asm – (Dis)assembler . . . . .	5
2.2	pwnypack.codec – Data transformation . . . . .	6
2.3	pwnypack.elf – ELF file parsing . . . . .	10
2.4	pwnypack.flow – Communication . . . . .	14
2.5	pwnypack.fmtstring – Format strings . . . . .	19
2.6	pwnypack.packing – Data (un)packing . . . . .	20
2.7	pwnypack.php – PHP related functions . . . . .	22
2.8	pwnypack.rop – ROP gadgets . . . . .	23
2.9	pwnypack.target – Target definition . . . . .	24
2.10	pwnypack.util – Utility functions . . . . .	25
<b>3</b>	<b>Indices and tables</b>	<b>27</b>
	<b>Python Module Index</b>	<b>29</b>



*pwnypack* is the official CTF toolkit of Certified Edible Dinosaurs. It aims to provide a set of command line utilities and a python library that are useful when playing hacking CTFs.

The core functionality of *pwnypack* is defined in the modules of the *pwnypack* package. The *pwny* package imports all that functionality into a single namespace for convenience.

Some of the functionality of the *pwnypack* package is also exported through a set of commandline utilities. Run `pwny help` after installing *pwnypack* to get a list of available utilities. You can create convenience symlinks for all the included apps by running `pwny symlink`. Each app has a help function that is accessible using the `-h` parameter.

For some example of how to use *pwnypack*, check the write-ups on the official [Certified Edible Dinosaurs](#) website.

Package contents:



---

## pwny package

---

The `pwny` package provides a convenience metapackage that imports the entire public API of *pwnypack* into a single namespace:

```
>>> from pwny import *
>>> enhex(asm('mov rax, 0xcd', target=Target(arch=Architecture.x86_64)))
u'b8ed0c0000'
```

For details about what exactly is made available, please consult the documentation of the individual [pwnypack modules](#).





---

## pwnypack package

---

All the functionality of *pwnypack* is implemented in the modules of this package.

### 2.1 *pwnypack.asm* – (Dis)assembler

This module contains functions to assemble and disassemble code for a given target platform.

Currently, the only supported architecture is x86 (both 32 and 64 bits variants). Assembly is performed by the *nasm* assembler (only supports *nasm* syntax). Disassembly is performed by *ndisasm* (*nasm* syntax) or *capstone* (*intel* & *att* syntax).

**class** *pwnypack.asm.AsmSyntax*

Bases: *enum.IntEnum*

This enumeration is used to specify the assembler syntax.

*pwnypack.asm.asm* (*code*, *addr=0*, *syntax=AsmSyntax.nasm*, *target=None*)

Assemble statements into machine readable code.

#### Parameters

- **code** (*str*) – The statements to assemble.
- **addr** (*int*) – The memory address where the code will run.
- **syntax** (*AsmSyntax*) – The input assembler syntax.
- **target** (*Target*) – The target architecture. The global target is used if this argument is *None*.

**Returns** The assembled machine code.

**Return type** *bytes*

#### Raises

- *SyntaxError* – If the assembler statements are invalid.
- *NotImplementedError* – In an unsupported target platform is specified.

#### Example

```
>>> from pwny import *
>>> asm('''
...     pop rdi
...     ret
... ''', target=Target(arch=Target.Arch.x86, bits=64))
b'\xc3'
```

`pwnypack.asm.disasm` (*code*, *addr*=0, *syntax*=*AsmSyntax.nasm*, *target*=None)

Disassemble machine readable code into human readable statements.

#### Parameters

- **code** (*bytes*) – The machine code that is to be disassembled.
- **addr** (*int*) – The memory address of the code (used for relative references).
- **syntax** (*AsmSyntax*) – The output assembler syntax.
- **target** (*Target*) – The architecture for which the code was written. The global target is used if this argument is None.

**Returns** The disassembled machine code.

**Return type** list of str

#### Raises

- `NotImplementedError` – In an unsupported target platform is specified.
- `RuntimeError` – If `ndisasm` encounters an error.

#### Example

```
>>> from pwny import *
>>> disasm(b'\xc3', target=Target(arch=Target.Arch.x86, bits=64))
['pop rdi', 'ret']
```

## 2.2 pwnypack.codec – Data transformation

This module contains functions that allow you to manipulate, encode or decode strings and byte sequences.

`pwnypack.codec.xor` (*key*, *data*)

Perform cyclical exclusive or operations on data.

The *key* can be a an integer ( $0 \leq key < 256$ ) or a byte sequence. If the key is smaller than the provided *data*, the *key* will be repeated.

#### Parameters

- **key** (*int or bytes*) – The key to xor data with.
- **data** (*bytes*) – The data to perform the xor operation on.

**Returns** The result of the exclusive or operation.

**Return type** bytes

## Examples

```
>>> from pwny import *
>>> xor(5, b'ABCD')
b'DGFA'
>>> xor(5, b'DGFA')
b'ABCD'
>>> xor(b'pwny', b'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
b'15==51)19=%5=9!)!%=-%!9!)-'
>>> xor(b'pwny', b'15==51)19=%5=9!)!%=-%!9!)-')
b'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

`pwnypack.codec.rot13(d)`

Rotate all characters in the alphabets A-Z and a-z by 13 positions in the alphabet. This is a *caesar()* shift of 13 along the fixed alphabets A-Z and a-z.

**Parameters** *d* (*str*) – The string to the apply the cipher to.

**Returns** The string with the rot13 cipher applied.

**Return type** *str*

## Examples

```
>>> rot13('whax')
'junk'
>>> rot13('junk')
'whax'
```

`pwnypack.codec.caesar(shift, data, shift_ranges=('az', 'AZ'))`

Apply a caesar cipher to a string.

The caesar cipher is a substitution cipher where each letter in the given alphabet is replaced by a letter some fixed number down the alphabet.

If *shift* is 1, *A* will become *B*, *B* will become *C*, etc...

You can define the alphabets that will be shift by specifying one or more shift ranges. The characters will than be shifted within the given ranges.

**Parameters**

- **shift** (*int*) – The shift to apply.
- **data** (*str*) – The string to apply the cipher to.
- **shift\_ranges** (*list of str*) – Which alphabets to shift.

**Returns** The string with the caesar cipher applied.

**Return type** *str*

## Examples

```
>>> caesar(16, 'Pwnypack')
'Fmdofqsa'
>>> caesar(-16, 'Fmdofqsa')
'Pwnypack'
>>> caesar(16, 'PWNYPack', shift_ranges=('AZ',))
```

```
'FMDOpack'
>>> caesar(16, 'PWNYPack', shift_ranges=('Az',))
'`g^iFqsA'
```

`pwnypack.codec.enhex(d, separator='')`

Convert bytes to their hexadecimal representation, optionally joined by a given separator.

**Parameters**

- **d** (*bytes*) – The data to convert to hexadecimal representation.
- **separator** (*str*) – The separator to insert between hexadecimal tuples.

**Returns** The hexadecimal representation of d.

**Return type** str

**Examples**

```
>>> from pwny import *
>>> enhex(b'pwnypack')
'70776e797061636b'
>>> enhex(b'pwnypack', separator=' ')
'70 77 6e 79 70 61 63 6b'
```

`pwnypack.codec.dehex(d)`

Convert a hexadecimal representation of a byte sequence to bytes. All non-hexadecimal characters will be removed from the input.

**Parameters** **d** (*str*) – The string of hexadecimal tuples.

**Returns** The byte sequence represented by d.

**Return type** bytes

**Examples**

```
>>> from pwny import *
>>> dehex('70776e797061636b')
b'pwnypack'
>>> dhex('70 77 6e 79 70 61 63 6b')
b'pwnypack'
```

`pwnypack.codec.enb64(d)`

Convert bytes to their base64 representation.

**Parameters** **d** (*bytes*) – The data to convert to its base64 representation.

**Returns** The base64 representation of d.

**Return type** str

**Example**

```
>>> from pwny import *
>>> enb64(b'pwnypack')
'cHdueXBhY2s='
```

`pwnypack.codec.deb64(d)`

Convert a base64 representation back to its original bytes.

**Parameters** `d` (*str*) – The base64 representation to decode.

**Returns** The bytes represented by `d`.

**Return type** `bytes`

#### Example

```
>>> from pwny import *
>>> deb64('cHdueXBhY2s=')
b'pwnypack'
```

`pwnypack.codec.enurlform(q)`

Convert a dictionary to a URL encoded query string.

**Parameters** `q` (*dict*) – The query to encode.

**Returns** The urlencoded representation of `q`.

**Return type** `str`

#### Example

```
>>> from pwny import *
>>> enurlform({'foo': 'bar', 'baz': ['quux', 'corge']})
'foo=bar&baz=quux&baz=corge'
```

`pwnypack.codec.deurlform(d)`

Convert a URL encoded query string to a dictionary.

**Parameters** `d` (*str*) – The URL encoded query string.

**Returns** A dictionary containing each key and all its values as a list.

**Return type** `dict`

#### Example

```
>>> from pwny import *
>>> deurlform('foo=bar&baz=quux&baz=corge')
{'foo': ['bar'], 'baz': ['quux', 'corge']}
```

`pwnypack.codec.enurlquote(v, plus=False)`

Percent encode a string for use in an URL.

**Parameters**

- `v` (*str*) – The value to percent encode.
- `plus` (*bool*) – Use a plus symbol for spaces, otherwise use `%20`.

**Returns** The percent encoded string.

**Return type** `str`

### Example

```
>>> from pwny import *
>>> enurlquote('Foo Bar/Baz', True)
'Foo+Bar/Baz'
```

`pwnypack.codec.deurlquote(d, plus=False)`  
Decode a percent encoded string.

#### Parameters

- **d** (*str*) – The percent encoded value to decode.
- **plus** (*bool*) – Parse a plus symbol as a space.

**Returns** The decoded version of the percent encoded of d.

**Return type** str

### Example

```
>>> from pwny import *
>>> deurlquote('Foo+Bar/Baz')
'Foo Bar/Baz'
```

`pwnypack.codec.frequency(v)`  
Perform a frequency analysis on a byte sequence or string.

**Parameters** **d** (*bytes or str*) – The sequence to analyse.

**Returns** A dictionary of unique elements in d and how often the occur.

**Return type** dict

### Example

```
>>> frequency('pwnypack')
{'a': 1, 'c': 1, 'k': 1, 'n': 1, 'p': 2, 'w': 1, 'y': 1}
```

## 2.3 pwnypack.elf – ELF file parsing

This module contains a parser for, and methods to extract information from ELF files.

**class** `pwnypack.elf.ELF` (*f=None*)

Bases: `pwnypack.target.Target`

A parser for ELF files. Upon parsing the ELF headers, it will not only fill the ELF specific fields but will also populate the inherited *arch*, *bits* and *endian* properties based on the values it encounters.

**Parameters** **f** (*str, file or None*) – The (path to) the ELF file to parse.

### Example

```

>>> from pwny import *
>>> e = ELF('my-executable')
>>> print(e.machine)
>>> print(e.program_headers)
>>> print(e.section_headers)
>>> print(e.symbols)

```

#### class Machine

Bases: `enum.IntEnum`

The target machine architecture.

#### class ELF.OSABI

Bases: `enum.IntEnum`

Describes the OS- or ABI-specific ELF extensions used by this file.

#### class ELF.ProgramHeader(*elf*, *data*)

Bases: `object`

Describes how the loader will load a part of a file. Called by the `ELF` class.

##### Parameters

- **elf** (`ELF`) – The ELF instance owning this program header.
- **data** – The content of the program header entry.

#### class Flags

Bases: `enum.IntEnum`

The individual flags that make up `ELF.ProgramHeader.flags`.

#### class ELF.ProgramHeader.Type

Bases: `enum.IntEnum`

The segment type.

`ELF.ProgramHeader.align = None`

`ELF.ProgramHeader.filesz = None`

`ELF.ProgramHeader.flags = None`

`ELF.ProgramHeader.memsz = None`

`ELF.ProgramHeader.offset = None`

`ELF.ProgramHeader.paddr = None`

`ELF.ProgramHeader.type = None`

`ELF.ProgramHeader.type_id = None`

`ELF.ProgramHeader.vaddr = None`

#### class ELF.SectionHeader(*elf*, *data*)

Bases: `object`

Describes a section of an ELF file. Called by the `ELF` class.

##### Parameters

- **elf** (`ELF`) – The ELF instance owning this section header.
- **data** – The content of the section header entry.

```
class Flags
    Bases: enum.IntEnum

class ELF.SectionHeader.Type
    Bases: enum.IntEnum

    Describes the section's type

ELF.SectionHeader.addr = None
ELF.SectionHeader.addralign = None
ELF.SectionHeader.content
    The contents of this section.
ELF.SectionHeader.elf = None
ELF.SectionHeader.entsize = None
ELF.SectionHeader.flags = None
ELF.SectionHeader.info = None
ELF.SectionHeader.link = None
ELF.SectionHeader.name = None
ELF.SectionHeader.name_index = None
ELF.SectionHeader.offset = None
ELF.SectionHeader.size = None
ELF.SectionHeader.type = None
ELF.SectionHeader.type_id = None

class ELF.Symbol (elf, data, str)
    Bases: object

    Contains information about symbols. Called by the ELF class.

    Parameters

    • elf (ELF) – The ELF instance owning this symbol.
    • data – The content of the symbol definition.
    • strs – The content of the string section associated with the symbol table.

class Binding
    Bases: enum.IntEnum

    Describes a symbol's binding.

class ELF.Symbol.SpecialSection
    Bases: enum.IntEnum

    Special section types.

class ELF.Symbol.Type
    Bases: enum.IntEnum

    Describes the symbol's type.

class ELF.Symbol.Visibility
    Bases: enum.IntEnum

    Describes the symbol's visibility.
```



```

    ELF.Symbol.content
        The contents of a symbol.
        Raises TypeError – If the symbol isn't defined until runtime.

    ELF.Symbol.elf = None

    ELF.Symbol.info = None

    ELF.Symbol.name = None

    ELF.Symbol.name_index = None

    ELF.Symbol.other = None

    ELF.Symbol.shndx = None

    ELF.Symbol.size = None

    ELF.Symbol.type = None

    ELF.Symbol.type_id = None

    ELF.Symbol.value = None

    ELF.Symbol.visibility = None

class ELF.Type
    Bases: enum.IntEnum

    Describes the object type.

    ELF.abi_version = None

    ELF.entry = None

    ELF.f = None

    ELF.flags = None

    ELF.get_program_header(index)
        Return a specific program header by its index.

        Parameters index (int) – The program header index.

        Returns ~ELF.ProgramHeader: The program header.

        Return type :class

        Raises KeyError – The specified index does not exist.

    ELF.get_section_header(section)
        Get a specific section header by index or name.

        Parameters section (int or str) – The index or name of the section header to return.

        Returns ~ELF.SectionHeader: The section header.

        Return type :class

        Raises KeyError – The requested section header does not exist.

    ELF.get_symbol(symbol)
        Get a specific symbol by index or name.

        Parameters symbol (int or str) – The index or name of the symbol to return.

        Returns The symbol.

        Return type ELF.Symbol

```

**Raises** `KeyError` – The requested symbol does not exist.

`ELF.hsize = None`

`ELF.machine = None`

`ELF.osabi = None`

`ELF.parse_file(f)`

Parse an ELF file and fill the class' properties.

**Parameters** `f` (*file or str*) – The (path to) the ELF file to read.

`ELF.phentsize = None`

`ELF.phnum = None`

`ELF.phoff = None`

`ELF.program_headers`

A list of all program headers.

`ELF.section_headers`

Return the list of section headers.

`ELF.shentsize = None`

`ELF.shnum = None`

`ELF.shoff = None`

`ELF.shstrndx = None`

`ELF.symbols`

Return a list of all symbols.

`ELF.type = None`

## 2.4 pwnypack.flow – Communication

The Flow module lets you connect to processes or network services using a unified API. It is primarily designed for synchronous communication flows.

It is based around the central `Flow` class which uses a `Channel` to connect to a process. The `Flow` class then uses the primitives exposed by the `Channel` to provide a high level API for reading/receiving and writing/sending data.

### Examples

```
>>> from pwny import *
>>> f = Flow.connect_tcp('ced.pwned.systems', 80)
>>> f.writelines([
...     b'GET / HTTP/1.0',
...     b'Host: ced.pwned.systems',
...     b'',
... ])
>>> line = f.readline().strip()
>>> print(line == b'HTTP/1.0 200 OK')
True
>>> f.until(b'\r\n\r\n')
>>> f.read_eof(echo=True)
... lots of html ...
```

```
>>> from pwny import *
>>> f = Flow.execute('cat')
>>> f.writeline(b'hello')
>>> f.readline(echo=True)
```

**class** `pwnypack.flow.ProcessChannel` (*executable, argument..., redirect\_stderr=False*)

Bases: `object`

This channel type allows controlling processes. It uses python's `subprocess.Popen` class to execute a process and allows you to communicate with it.

#### Parameters

- **executable** (*str*) – The executable to start.
- **argument...** (*list of str*) – The arguments to pass to the executable.
- **redirect\_stderr** (*bool*) – Whether to also capture the output of stderr.

**close** ()

Wait for the subprocess to exit.

**fileno** ()

Return the file descriptor number for the stdout channel of this process.

**kill** ()

Terminate the subprocess.

**read** (*n*)

Read *n* bytes from the subprocess' output channel.

**Parameters** *n* (*int*) – The number of bytes to read.

**Returns** *n* bytes of output.

**Return type** `bytes`

**Raises** `EOFError` – If the process exited.

**write** (*data*)

Write *n* bytes to the subprocess' input channel.

**Parameters** *data* (*bytes*) – The data to write.

**Raises** `EOFError` – If the process exited.

**class** `pwnypack.flow.SocketChannel` (*sock*)

Bases: `object`

This channel type allows controlling sockets.

**Parameters** **socket** (*socket.socket*) – The (already connected) socket to control.

**close** ()

Close the socket gracefully.

**fileno** ()

Return the file descriptor number for the socket.

**kill** ()

Shut down the socket immediately.

**read** (*n*)

Receive *n* bytes from the socket.

**Parameters** *n* (*int*) – The number of bytes to read.

**Returns** *n* bytes read from the socket.

**Return type** bytes

**Raises** EOFError – If the socket was closed.

**write** (*data*)

Send *n* bytes to socket.

**Parameters** *data* (bytes) – The data to send.

**Raises** EOFError – If the socket was closed.

**class** pwnypack.flow.TCPClientSocketChannel (*host*, *port*)

Bases: [pwnypack.flow.SocketChannel](#)

Convenience subclass of [SocketChannel](#) that allows you to connect to a TCP hostname / port pair easily.

**Parameters**

- **host** (*str*) – The hostname or IP address to connect to.
- **port** (*int*) – The port number to connect to.

**class** pwnypack.flow.Flow (*channel*, *echo=False*)

Bases: object

The core class of *Flow*. Takes a channel and exposes synchronous utility functions for communications.

Usually, you'll use the convenience classmethods [connect\\_tcp\(\)](#) or [execute\(\)](#) instead of manually creating the constructor directly.

**Parameters**

- **channel** (Channel) – A channel.
- **echo** (*bool*) – Whether or not to echo all input / output.

**close** ()

Gracefully close the channel.

**static connect\_ssh** (\**args*, \*\**kwargs*)

Create a new connected SSHClient instance. All arguments are passed to [SSHClient.connect\(\)](#).

**classmethod connect\_tcp** (*host*, *port*, *echo=False*)

Set up a [TCPClientSocketChannel](#) and create a [Flow](#) instance for it.

**Parameters**

- **host** (*str*) – The hostname or IP address to connect to.
- **port** (*int*) – The port number to connect to.
- **echo** (*bool*) – Whether to echo read/written data to stdout by default.

**Returns** *Flow*: A Flow instance initialised with the TCP socket channel.

**Return type** :class

**classmethod execute** (*executable*, \**arguments*, \*\**kwargs*)

execute(executable, argument..., redirect\_stderr=False, echo=False):

Set up a [ProcessChannel](#) and create a [Flow](#) instance for it.

**Parameters**

- **executable** (*str*) – The executable to start.
- **argument...** (*list of str*) – The arguments to pass to the executable.

- **redirect\_stderr** (*bool*) – Whether to also capture the output of stderr.
- **echo** (*bool*) – Whether to echo read/written data to stdout by default.

**Returns** *Flow*: A Flow instance initialised with the process channel.

**Return type** :class

**classmethod** **execute\_ssh** (*command*, *arguments...*, *pty=False*, *echo=False*)

Execute *command* on a remote server. It first calls *Flow.connect\_ssh()* using all positional and keyword arguments, then calls *SSHClient.execute()* with the command and *pty* / *echo* options.

**Parameters**

- **command** (*str*) – The command to execute on the remote server.
- **arguments...** – The options for the SSH connection.
- **pty** (*bool*) – Request a pseudo-terminal from the server.
- **echo** (*bool*) – Whether to echo read/written data to stdout by default.

**Returns** *Flow*: A Flow instance initialised with the SSH channel.

**Return type** :class

**interact** ()

Interact with the socket. This will send all keyboard input to the socket and input from the socket to the console until an EOF occurs.

**classmethod** **invoke\_ssh\_shell** (*\*args*, *\*\*kwargs*)

*invoke\_ssh(arguments..., pty=False, echo=False)*

Star a new shell on a remote server. It first calls *Flow.connect\_ssh()* using all positional and keyword arguments, then calls *SSHClient.invoke\_shell()* with the *pty* / *echo* options.

**Parameters**

- **arguments...** – The options for the SSH connection.
- **pty** (*bool*) – Request a pseudo-terminal from the server.
- **echo** (*bool*) – Whether to echo read/written data to stdout by default.

**Returns** *Flow*: A Flow instance initialised with the SSH channel.

**Return type** :class

**kill** ()

Terminate the channel immediately.

**classmethod** **listen\_tcp** (*host=''*, *port=0*, *echo=False*)

Set up a *TCPServerSocketChannel* and create a *Flow* instance for it.

**Parameters**

- **host** (*str*) – The hostname or IP address to bind to.
- **port** (*int*) – The port number to listen on.
- **echo** (*bool*) – Whether to echo read/written data to stdout by default.

**Returns** *Flow*: A Flow instance initialised with the TCP socket channel.

**Return type** :class

**read** (*n*, *echo=None*)

Read *n* bytes from the channel.

### Parameters

- **n** (*int*) – The number of bytes to read from the channel.
- **echo** (*bool*) – Whether to write the read data to stdout.

**Returns** *n* bytes of data.

**Return type** bytes

**Raises** `EOFError` – If the channel was closed.

**read\_eof** (*echo=None*)

Read until the channel is closed.

**Parameters** **echo** (*bool*) – Whether to write the read data to stdout.

**Returns** The read data.

**Return type** bytes

**read\_until** (*s, echo=None*)

Read until a certain string is encountered..

### Parameters

- **s** (*bytes*) – The string to wait for.
- **echo** (*bool*) – Whether to write the read data to stdout.

**Returns** The data up to and including *s*.

**Return type** bytes

**Raises** `EOFError` – If the channel was closed.

**readline** (*echo=None*)

Read 1 line from channel.

**Parameters** **echo** (*bool*) – Whether to write the read data to stdout.

**Returns** The read line which includes new line character.

**Return type** bytes

**Raises** `EOFError` – If the channel was closed before a line was read.

**readlines** (*n, echo=None*)

Read *n* lines from channel.

### Parameters

- **n** (*int*) – The number of lines to read.
- **echo** (*bool*) – Whether to write the read data to stdout.

**Returns** *n* lines which include new line characters.

**Return type** list of bytes

**Raises** `EOFError` – If the channel was closed before *n* lines were read.

**until** (*s, echo=None*)

Read until a certain string is encountered..

### Parameters

- **s** (*bytes*) – The string to wait for.
- **echo** (*bool*) – Whether to write the read data to stdout.

**Returns** The data up to and including *s*.

**Return type** bytes

**Raises** `EOFError` – If the channel was closed.

**write** (*data*, *echo=None*)

Write data to channel.

**Parameters**

- **data** (*bytes*) – The data to write to the channel.
- **echo** (*bool*) – Whether to echo the written data to stdout.

**Raises** `EOFError` – If the channel was closed before all data was sent.

**writeline** (*line=b' ', sep=b'\n', echo=None*)

Write a byte sequences to the channel and terminate it with carriage return and line feed.

**Parameters**

- **line** (*bytes*) – The line to send.
- **sep** (*bytes*) – The separator to use after each line.
- **echo** (*bool*) – Whether to echo the written data to stdout.

**Raises** `EOFError` – If the channel was closed before all data was sent.

**writelines** (*lines*, *sep=b'\n', echo=None*)

Write a list of byte sequences to the channel and terminate them with a separator (line feed).

**Parameters**

- **lines** (*list of bytes*) – The lines to send.
- **sep** (*bytes*) – The separator to use after each line.
- **echo** (*bool*) – Whether to echo the written data to stdout.

**Raises** `EOFError` – If the channel was closed before all data was sent.

## 2.5 pwnypack.fmtstring – Format strings

The `fmtstring` module allows you to build format strings that can be used to exploit format string bugs (`printf(buf);`).

`pwnypack.fmtstring.fmtstring` (*offset*, *writes*, *written=0*, *max\_width=2*, *target=None*)

Build a format string that writes given data to given locations. Can be used easily create format strings to exploit format string bugs.

*writes* is a list of 2- or 3-item tuples. Each tuple represents a memory write starting with an absolute address, then the data to write as an integer and finally the width (1, 2, 4 or 8) of the write.

`fmtstring()` will break up the writes and try to optimise the order to minimise the amount of dummy output generated.

**Parameters**

- **offset** (*int*) – The parameter offset where the format string start.
- **writes** (*list*) – A list of 2 or 3 item tuples.
- **written** (*int*) – How many bytes have already been written before the built format string starts.

- **max\_width** (*int*) – The maximum width of the writes (1, 2 or 4).
- **target** (*pwnypack.target.Target*) – The target architecture.

**Returns** The format string that will execute the specified memory writes.

**Return type** bytes

### Example

The following example will (on a 32bit architecture) build a format string that write 0xc0debabe to the address 0xdeadbeef and the byte 0x90 to 0xdeadbeef + 4 assuming that the input buffer is located at offset 3 on the stack.

```
>>> from pwny import *
>>> fmtstring(3, [(0xdeadbeef, 0xc0debabe), (0xdeadbeef + 4, 0x90, 1)])
```

## 2.6 pwnypack.packing – Data (un)packing

`pwnypack.packing.pack` (*fmt, v1, v2, ..., endian=None, target=None*)

Return a string containing the values *v1*, *v2*, ... packed according to the given format. The actual packing is performed by `struct.pack` but the byte order will be set according to the given *endian*, *target* or byte order of the global target.

### Parameters

- **fmt** (*str*) – The format string.
- **v1, v2, ...** – The values to pack.
- **endian** (*Endian*) – Override the default byte order. If *None*, it will look at the byte order of the *target* argument.
- **target** (*Target*) – Override the default byte order. If *None*, it will look at the byte order of the global target.

**Returns** The provided values packed according to the format.

**Return type** bytes

`pwnypack.packing.unpack` (*fmt, data, endian=None, target=None*)

Unpack the string (presumably packed by `pack(fmt, ...)`) according to the given format. The actual unpacking is performed by `struct.unpack` but the byte order will be set according to the given *endian*, *target* or byte order of the global target.

### Parameters

- **fmt** (*str*) – The format string.
- **data** (*bytes*) – The data to unpack.
- **endian** (*Endian*) – Override the default byte order. If *None*, it will look at the byte order of the *target* argument.
- **target** (*Target*) – Override the default byte order. If *None*, it will look at the byte order of the global target.

**Returns** The unpacked values according to the format.

**Return type** list

`pwnypack.packing.pack_size` (*fmt, endian=None, target=None*)



`pwnypack.packing.P` (*value*, *bits=None*, *endian=None*, *target=None*)

Pack an unsigned pointer for a given target.

#### Parameters

- **value** (*int*) – The value to pack.
- **bits** (*Bits*) – Override the default word size. If *None* it will look at the word size of *target*.
- **endian** (*Endian*) – Override the default byte order. If *None*, it will look at the byte order of the *target* argument.
- **target** (*Target*) – Override the default byte order. If *None*, it will look at the byte order of the global *target*.

`pwnypack.packing.p` (*value*, *bits=None*, *endian=None*, *target=None*)

Pack a signed pointer for a given target.

#### Parameters

- **value** (*int*) – The value to pack.
- **bits** (`pwnypack.target.Target.Bits`) – Override the default word size. If *None* it will look at the word size of *target*.
- **endian** (*Endian*) – Override the default byte order. If *None*, it will look at the byte order of the *target* argument.
- **target** (*Target*) – Override the default byte order. If *None*, it will look at the byte order of the global *target*.

`pwnypack.packing.U` (*data*, *bits=None*, *endian=None*, *target=None*)

Unpack an unsigned pointer for a given target.

#### Parameters

- **data** (*bytes*) – The data to unpack.
- **bits** (`pwnypack.target.Target.Bits`) – Override the default word size. If *None* it will look at the word size of *target*.
- **endian** (*Endian*) – Override the default byte order. If *None*, it will look at the byte order of the *target* argument.
- **target** (*Target*) – Override the default byte order. If *None*, it will look at the byte order of the global *target*.

**Returns** The pointer value.

**Return type** `int`

`pwnypack.packing.u` (*data*, *bits=None*, *endian=None*, *target=None*)

Unpack a signed pointer for a given target.

#### Parameters

- **data** (*bytes*) – The data to unpack.
- **bits** (`pwnypack.target.Target.Bits`) – Override the default word size. If *None* it will look at the word size of *target*.
- **endian** (*Endian*) – Override the default byte order. If *None*, it will look at the byte order of the *target* argument.

- **target** (*Target*) – Override the default byte order. If *None*, it will look at the byte order of the global *target*.

**Returns** The pointer value.

**Return type** *int*

```
pwnypack.packing.p8 (value, endian=None, target=None)
    Pack signed 8 bit integer. Alias for pack ('b', ...).

pwnypack.packing.P8 (value, endian=None, target=None)
    Pack unsigned 8 bit integer. Alias for pack ('B', ...).

pwnypack.packing.u8 (data, endian=None, target=None)
    Unpack signed 8 bit integer. Alias for unpack ('b', ...).

pwnypack.packing.U8 (data, endian=None, target=None)
    Unpack unsigned 8 bit integer. Alias for unpack ('B', ...).

pwnypack.packing.p16 (value, endian=None, target=None)
    Pack signed 16 bit integer. Alias for pack ('h', ...).

pwnypack.packing.P16 (value, endian=None, target=None)
    Pack unsigned 16 bit integer. Alias for pack ('H', ...).

pwnypack.packing.u16 (data, endian=None, target=None)
    Unpack signed 16 bit integer. Alias for unpack ('h', ...).

pwnypack.packing.U16 (data, endian=None, target=None)
    Unpack unsigned 16 bit integer. Alias for unpack ('H', ...).

pwnypack.packing.p32 (value, endian=None, target=None)
    Pack signed 32 bit integer. Alias for pack ('l', ...).

pwnypack.packing.P32 (value, endian=None, target=None)
    Pack unsigned 32 bit integer. Alias for pack ('L', ...).

pwnypack.packing.u32 (data, endian=None, target=None)
    Unpack signed 32 bit integer. Alias for unpack ('l', ...).

pwnypack.packing.U32 (data, endian=None, target=None)
    Unpack unsigned 32 bit integer. Alias for unpack ('L', ...).

pwnypack.packing.p64 (value, endian=None, target=None)
    Pack signed 64 bit integer. Alias for pack ('q', ...).

pwnypack.packing.P64 (value, endian=None, target=None)
    Pack unsigned 64 bit integer. Alias for pack ('Q', ...).

pwnypack.packing.u64 (data, endian=None, target=None)
    Unpack signed 64 bit integer. Alias for unpack ('q', ...).

pwnypack.packing.U64 (data, endian=None, target=None)
    Unpack unsigned 64 bit integer. Alias for unpack ('Q', ...).
```

## 2.7 pwnypack.php – PHP related functions

```
pwnypack.php.php_serialize (value)
    Serialize a value for use with PHP's unserialize() function. This function can serialize bytes, strings, integers, floats, booleans, None, lists, dicts and custom objects implementing __php__().
```

**Parameters** *value* – The value to serialize.

**Returns** The serialized form of *value* ready to be unserialized by PHP.

**Return type** bytes

### Example

```
>>> from pwny import *
>>> php_serialize([b'foo', u'bar', 42, 2.5, True, None, {'a': 'b'}])
b'a:7:{i:0;s:3:"foo";i:1;s:3:"bar";i:2;i:42;i:3;d:2.5;i:4;b:1;i:5;N;i:6;a:1:{s:1:"a";s:1:"b";}}'
```

**class** pwnypack.php.PhpObject (*class\_name*, *properties=None*)

Bases: object

Helper class to represent PHP objects for serialization using `php_serialize()`.

Instances of this class act like a dictionary of properties that should be set on the deserialized PHP instance. You can prefix the property names with 'public ', 'protected ' or 'private ' to ensure the correct instance variables are set.

### Parameters

- **class\_name** (*str*) – The name of the PHP class to use when deserializing.
- **properties** (*dict*) – The properties to deserialize in this instance.

### Example

```
>>> from pwny import *
>>> o = PhpObject('Foo\Bar', {'protected fg': '#000000'})
>>> php_serialize(o)
b'O:7:"Foo\Bar":1:{s:5:"\x00*\x00fg";s:7:"#000000";}'
```

## 2.8 pwnypack.rop – ROP gadgets

The ROP module contains a function to find gadgets in ELF binaries that can be used to create ROP chains.

**pwnypack.rop.find\_gadget** (*elf*, *gadget*, *align=1*, *unique=True*)

Find a ROP gadget in a the executable sections of an ELF executable or library. The ROP gadget can be either a set of bytes for an exact match or a (bytes) regular expression. Once it finds gadgets, it uses the capstone engine to verify if the gadget consists of valid instructions and doesn't contain any call or jump instructions.

### Parameters

- **elf** (*ELF*) – The ELF instance to find a gadget in.
- **gadget** (*bytes or regexp*) – The gadget to find.
- **align** (*int*) – Make sure the gadget starts at a multiple of this number
- **unique** (*bool*) – If true, only unique gadgets are returned.

### Returns

A dictionary containing a description of the found gadget. Contains the following fields:

- **section**: The section the gadget was found in.
- **offset**: The offset inside the segment the gadget was found at.

- `addr`: The virtual memory address the gadget will be located at.
- `gadget`: The machine code of the found gadget.
- `asm`: A list of disassembled instructions.

**Return type** dict

## 2.9 pwnypack.target – Target definition

The *Target* class describes the architecture of a targeted machine, executable or environment. It encodes the generic architecture, the word size, the byte order and an architecture dependant mode.

It is used throughout *pwnypack* to determine how data should be interpreted or emitted.

**class** `pwnypack.target.Target` (*arch=None, bits=None, endian=None, mode=0*)

Bases: object

**class** `Arch`

Bases: `enum.Enum`

Describes the general architecture of a target.

**class** `Target.Bits`

Bases: `enum.IntEnum`

The target architecture's word size.

**class** `Target.Endian`

Bases: `enum.IntEnum`

The target architecture's byte order.

**class** `Target.Mode`

Bases: `enum.IntEnum`

Architecture dependant mode flags.

`Target.arch`

The target's architecture. One of *Target.Arch*.

`Target.assume` (*other*)

Assume the identity of another target. This can be useful to make the global target assume the identity of an ELF executable.

**Parameters** *other* (*Target*) – The target whose identity to assume.

### Example

```
>>> from pwny import *
>>> target.assume(ELF('my-executable'))
```

`Target.bits`

The target architecture word size. One of *Target.Bits*.

`Target.endian`

The target architecture byte order. One of *Target.Endian*.

`Target.mode`

The target architecture dependant flags. OR'ed values of *Target.Mode*.

## 2.10 pwnypack.util – Utility functions

The util module contains various utility functions.

`pwnypack.util.cycle` (*length*, *width*=4)

Generate a de Bruijn sequence of a given length (and width). A de Bruijn sequence is a set of varying repetitions where each sequence of *n* characters is unique within the sequence. This type of sequence can be used to easily find the offset to the return pointer when exploiting a buffer overflow.

### Parameters

- **length** (*int*) – The length of the sequence to generate.
- **width** (*int*) – The width of each element in the sequence.

**Returns** The sequence.

**Return type** str

### Example

```
>>> from pwny import *
>>> cycle(80)
AAAABAAACAAADAAAEAAAFAAAGAAAHAAIAAAJAAAKAAALAAAMAAANAAAOAAAPAAAQAAARAAASAAATAAA
```

`pwnypack.util.cycle_find` (*key*, *width*=4)

Given an element of a de Bruijn sequence, find its index in that sequence.

### Parameters

- **key** (*str*) – The piece of the de Bruijn sequence to find.
- **width** (*int*) – The width of each element in the sequence.

**Returns** The index of *key* in the de Bruijn sequence.

**Return type** int

`pwnypack.util.reghex` (*pattern*)

Compile a regular hexpression (a short form regular expression subset specifically designed for searching for binary strings).

A regular hexpression consists of hex tuples interspaced with control characters. The available control characters are:

- `?:` Any byte (optional).
- `..` Any byte (required).
- `?{n}`: A set of 0 up to *n* bytes.
- `.{n}`: A set of exactly *n* bytes.
- `*:` Any number of bytes (or no bytes at all).
- `+:` Any number of bytes (at least one byte).

**Parameters** **pattern** (*str*) – The reghex pattern.

**Returns** A regular expression as returned by `re.compile()`.

**Return type** regexp

`pwnypack.util.pickle_call` (*func*, \**args*)

Create a byte sequence which when unpickled calls a callable with given arguments.

**Parameters**

- **func** (*callable*) – The function to call or class to instantiate.
- **args** (*tuple*) – The arguments to call the callable with.

**Returns** The data that when unpickled calls `func(*args)`.

**Return type** bytes

**Example**

```
>>> from pwny import *
>>> import pickle
>>> def hello(arg):
...     print('Hello, %s!' % arg)
...
>>> pickle.loads(pickle_call(hello, 'world'))
Hello, world!
```

---

## Indices and tables

---

- `genindex`
- `modindex`





## p

- `pwnypack.asm`, 5
- `pwnypack.codec`, 6
- `pwnypack.elf`, 10
- `pwnypack.flow`, 14
- `pwnypack.fmtstring`, 19
- `pwnypack.packing`, 20
- `pwnypack.php`, 22
- `pwnypack.rop`, 23
- `pwnypack.target`, 24
- `pwnypack.util`, 25



## A

abi\_version (pwnypack.elf.ELF attribute), 13  
addr (pwnypack.elf.ELF.SectionHeader attribute), 12  
addralign (pwnypack.elf.ELF.SectionHeader attribute), 12  
align (pwnypack.elf.ELF.ProgramHeader attribute), 11  
arch (pwnypack.target.Target attribute), 24  
asm() (in module pwnypack.asm), 5  
AsmSyntax (class in pwnypack.asm), 5  
assume() (pwnypack.target.Target method), 24

## B

bits (pwnypack.target.Target attribute), 24

## C

caesar() (in module pwnypack.codec), 7  
close() (pwnypack.flow.Flow method), 16  
close() (pwnypack.flow.ProcessChannel method), 15  
close() (pwnypack.flow.SocketChannel method), 15  
connect\_ssh() (pwnypack.flow.Flow static method), 16  
connect\_tcp() (pwnypack.flow.Flow class method), 16  
content (pwnypack.elf.ELF.SectionHeader attribute), 12  
content (pwnypack.elf.ELF.Symbol attribute), 12  
cycle() (in module pwnypack.util), 25  
cycle\_find() (in module pwnypack.util), 25

## D

deb64() (in module pwnypack.codec), 8  
dehex() (in module pwnypack.codec), 8  
deurlform() (in module pwnypack.codec), 9  
deurlquote() (in module pwnypack.codec), 10  
disasm() (in module pwnypack.asm), 6

## E

ELF (class in pwnypack.elf), 10  
elf (pwnypack.elf.ELF.SectionHeader attribute), 12  
elf (pwnypack.elf.ELF.Symbol attribute), 13  
ELF.Machine (class in pwnypack.elf), 11  
ELF.OSABI (class in pwnypack.elf), 11  
ELF.ProgramHeader (class in pwnypack.elf), 11

ELF.ProgramHeader.Flags (class in pwnypack.elf), 11  
ELF.ProgramHeader.Type (class in pwnypack.elf), 11  
ELF.SectionHeader (class in pwnypack.elf), 11  
ELF.SectionHeader.Flags (class in pwnypack.elf), 11  
ELF.SectionHeader.Type (class in pwnypack.elf), 12  
ELF.Symbol (class in pwnypack.elf), 12  
ELF.Symbol.Binding (class in pwnypack.elf), 12  
ELF.Symbol.SpecialSection (class in pwnypack.elf), 12  
ELF.Symbol.Type (class in pwnypack.elf), 12  
ELF.Symbol.Visibility (class in pwnypack.elf), 12  
ELF.Type (class in pwnypack.elf), 13  
enb64() (in module pwnypack.codec), 8  
endian (pwnypack.target.Target attribute), 24  
enhex() (in module pwnypack.codec), 8  
entry (pwnypack.elf.ELF attribute), 13  
entsize (pwnypack.elf.ELF.SectionHeader attribute), 12  
enurlform() (in module pwnypack.codec), 9  
enurlquote() (in module pwnypack.codec), 9  
execute() (pwnypack.flow.Flow class method), 16  
execute\_ssh() (pwnypack.flow.Flow class method), 17

## F

f (pwnypack.elf.ELF attribute), 13  
fileno() (pwnypack.flow.ProcessChannel method), 15  
fileno() (pwnypack.flow.SocketChannel method), 15  
filesz (pwnypack.elf.ELF.ProgramHeader attribute), 11  
find\_gadget() (in module pwnypack.rop), 23  
flags (pwnypack.elf.ELF attribute), 13  
flags (pwnypack.elf.ELF.ProgramHeader attribute), 11  
flags (pwnypack.elf.ELF.SectionHeader attribute), 12  
Flow (class in pwnypack.flow), 16  
fmtstring() (in module pwnypack.fmtstring), 19  
frequency() (in module pwnypack.codec), 10

## G

get\_program\_header() (pwnypack.elf.ELF method), 13  
get\_section\_header() (pwnypack.elf.ELF method), 13  
get\_symbol() (pwnypack.elf.ELF method), 13

## H

hsize (pwnypack.elf.ELF attribute), 14

## I

info (pwnypack.elf.ELF.SectionHeader attribute), 12  
 info (pwnypack.elf.ELF.Symbol attribute), 13  
 interact() (pwnypack.flow.Flow method), 17  
 invoke\_ssh\_shell() (pwnypack.flow.Flow class method), 17

## K

kill() (pwnypack.flow.Flow method), 17  
 kill() (pwnypack.flow.ProcessChannel method), 15  
 kill() (pwnypack.flow.SocketChannel method), 15

## L

link (pwnypack.elf.ELF.SectionHeader attribute), 12  
 listen\_tcp() (pwnypack.flow.Flow class method), 17

## M

machine (pwnypack.elf.ELF attribute), 14  
 memsz (pwnypack.elf.ELF.ProgramHeader attribute), 11  
 mode (pwnypack.target.Target attribute), 24

## N

name (pwnypack.elf.ELF.SectionHeader attribute), 12  
 name (pwnypack.elf.ELF.Symbol attribute), 13  
 name\_index (pwnypack.elf.ELF.SectionHeader attribute), 12  
 name\_index (pwnypack.elf.ELF.Symbol attribute), 13

## O

offset (pwnypack.elf.ELF.ProgramHeader attribute), 11  
 offset (pwnypack.elf.ELF.SectionHeader attribute), 12  
 osabi (pwnypack.elf.ELF attribute), 14  
 other (pwnypack.elf.ELF.Symbol attribute), 13

## P

P() (in module pwnypack.packing), 20  
 p() (in module pwnypack.packing), 21  
 P16() (in module pwnypack.packing), 22  
 p16() (in module pwnypack.packing), 22  
 P32() (in module pwnypack.packing), 22  
 p32() (in module pwnypack.packing), 22  
 P64() (in module pwnypack.packing), 22  
 p64() (in module pwnypack.packing), 22  
 P8() (in module pwnypack.packing), 22  
 p8() (in module pwnypack.packing), 22  
 pack() (in module pwnypack.packing), 20  
 pack\_size() (in module pwnypack.packing), 20  
 paddr (pwnypack.elf.ELF.ProgramHeader attribute), 11  
 parse\_file() (pwnypack.elf.ELF method), 14  
 phentsize (pwnypack.elf.ELF attribute), 14  
 phnum (pwnypack.elf.ELF attribute), 14  
 phoff (pwnypack.elf.ELF attribute), 14  
 php\_serialize() (in module pwnypack.php), 22

PhpObject (class in pwnypack.php), 23  
 pickle\_call() (in module pwnypack.util), 25  
 ProcessChannel (class in pwnypack.flow), 15  
 program\_headers (pwnypack.elf.ELF attribute), 14  
 pwnypack.asm (module), 5  
 pwnypack.codec (module), 6  
 pwnypack.elf (module), 10  
 pwnypack.flow (module), 14  
 pwnypack.fmtstring (module), 19  
 pwnypack.packing (module), 20  
 pwnypack.php (module), 22  
 pwnypack.rop (module), 23  
 pwnypack.target (module), 24  
 pwnypack.util (module), 25

## R

read() (pwnypack.flow.Flow method), 17  
 read() (pwnypack.flow.ProcessChannel method), 15  
 read() (pwnypack.flow.SocketChannel method), 15  
 read\_eof() (pwnypack.flow.Flow method), 18  
 read\_until() (pwnypack.flow.Flow method), 18  
 readline() (pwnypack.flow.Flow method), 18  
 readlines() (pwnypack.flow.Flow method), 18  
 reghex() (in module pwnypack.util), 25  
 rot13() (in module pwnypack.codec), 7

## S

section\_headers (pwnypack.elf.ELF attribute), 14  
 shentsize (pwnypack.elf.ELF attribute), 14  
 shndx (pwnypack.elf.ELF.Symbol attribute), 13  
 shnum (pwnypack.elf.ELF attribute), 14  
 shoff (pwnypack.elf.ELF attribute), 14  
 shstrndx (pwnypack.elf.ELF attribute), 14  
 size (pwnypack.elf.ELF.SectionHeader attribute), 12  
 size (pwnypack.elf.ELF.Symbol attribute), 13  
 SocketChannel (class in pwnypack.flow), 15  
 symbols (pwnypack.elf.ELF attribute), 14

## T

Target (class in pwnypack.target), 24  
 Target.Arch (class in pwnypack.target), 24  
 Target.Bits (class in pwnypack.target), 24  
 Target.Endian (class in pwnypack.target), 24  
 Target.Mode (class in pwnypack.target), 24  
 TCPClientSocketChannel (class in pwnypack.flow), 16  
 type (pwnypack.elf.ELF attribute), 14  
 type (pwnypack.elf.ELF.ProgramHeader attribute), 11  
 type (pwnypack.elf.ELF.SectionHeader attribute), 12  
 type (pwnypack.elf.ELF.Symbol attribute), 13  
 type\_id (pwnypack.elf.ELF.ProgramHeader attribute), 11  
 type\_id (pwnypack.elf.ELF.SectionHeader attribute), 12  
 type\_id (pwnypack.elf.ELF.Symbol attribute), 13

## U

`U()` (in module `pwnypack.packing`), 21  
`u()` (in module `pwnypack.packing`), 21  
`U16()` (in module `pwnypack.packing`), 22  
`u16()` (in module `pwnypack.packing`), 22  
`U32()` (in module `pwnypack.packing`), 22  
`u32()` (in module `pwnypack.packing`), 22  
`U64()` (in module `pwnypack.packing`), 22  
`u64()` (in module `pwnypack.packing`), 22  
`U8()` (in module `pwnypack.packing`), 22  
`u8()` (in module `pwnypack.packing`), 22  
`unpack()` (in module `pwnypack.packing`), 20  
`until()` (`pwnypack.flow.Flow` method), 18

## V

`vaddr` (`pwnypack.elf.ELF.ProgramHeader` attribute), 11  
`value` (`pwnypack.elf.ELF.Symbol` attribute), 13  
`visibility` (`pwnypack.elf.ELF.Symbol` attribute), 13

## W

`write()` (`pwnypack.flow.Flow` method), 19  
`write()` (`pwnypack.flow.ProcessChannel` method), 15  
`write()` (`pwnypack.flow.SocketChannel` method), 16  
`writeline()` (`pwnypack.flow.Flow` method), 19  
`writelines()` (`pwnypack.flow.Flow` method), 19

## X

`xor()` (in module `pwnypack.codec`), 6