### Contents

1 About libnmap 1

2 libnmap’s modules 3  
   2.1 libnmap.process  ................................................................. 3  
   2.2 libnmap.parser  ....................................................................... 9  
   2.3 libnmap.objects  ...................................................................... 12  
   2.4 libnmap.objects.cpe ................................................................. 12  
   2.5 libnmap.objects.host ............................................................... 13  
   2.6 libnmap.objects.report ............................................................. 16  
   2.7 libnmap.objects.service ............................................................ 18  
   2.8 libnmap.objects.os ................................................................. 20  
   2.9 libnmap.diff ............................................................ 22  
   2.10 libnmap.plugins.s3.NmapS3Plugin ........................................... 25

3 Indices and tables 27

Python Module Index 29
About libnmap

libnmap is a python toolkit for manipulating nmap. It currently offers the following modules:

- process: enables you to launch nmap scans
- parse: enables you to parse nmap reports or scan results (only XML so far) from a file, a string,...
- report: enables you to manipulate a parsed scan result and de/serialize scan results in a json format
- diff: enables you to see what changed between two scans
- objects: contains basic nmap objects like NmapHost and NmapService. It is to note that each object can be “diff()ed” with another similar object.
  - report: contains NmapReport class definition
  - host: contains NmapHost class definition
  - service: contains NmapService class definition
  - os: contains NmapOSFingerprint class definition and some other classes like NmapOSMatch, NmapOSClass,...
  - cpe: contains CPE class definition
- plugins: enables you to support datastores for your scan results directly in the “NmapReport” object from report module
  - mongodb: only plugin implemented so far, ultra basic, for POC purpose only
  - sqlalchemy: Allow to store/retrieve NmapReport to sqlite/mysql/... all engine supported by sqlalchemy
  - rabbitMQ: todo
  - couchdb: todo
  - elastic search: todo
  - csv: todo
libnmap’s modules

The full source code is available on GitHub. Please, do not hesitate to fork it and issue pull requests.

The different modules are documented below:

## 2.1 libnmap.process

### 2.1.1 Purpose of libnmap.process

The purpose of this module is to enable the lib users to launch and control nmap scans. This module will consequently fire the nmap command following the specified parameters provided in the constructor.

It is to note that this module will not perform a full inline parsing of the data. Only specific events are parsed and exploitable via either a callback function defined by the user and provided in the constructor; either by running the process in the background and accessing the NmapProcess attributes will the scan is running.

To run an nmap scan, you need to:

- instanciate NmapProcess
- call the run*() methods

Raw results of the scans will be available in the following properties:

- NmapProcess.stdout: string, XML output
- NmapProcess.stderr: string, text error message from nmap process

To instanciate a NmapProcess instance, call the constructor with appropriate parameters

### 2.1.2 Processing of events

While Nmap is running, some events are process and parsed. This would enable you to:

- evaluate estimated time to completion and progress in percentage
- find out which task is running and how many nmap task have been executed
- know the start time and nmap version

As you may know, depending on the nmap options you specified, nmap will execute several tasks like “DNS Resolve”, “Ping Scan”, “Connect Scan”, “NSE scripts”,... This is of course independent from libnmap but the lib is able to parse these tasks and will instanciate a NmapTask object for any task executed. The list of executed task is available via the following properties:
2.1.3 Using libnmap.process

This module enables you to launch nmap scans with simple python commands:

```python
def do_scan(targets, options):
    nm = NmapProcess(targets, options)
    result = nm.run()

    if result['rc'] == 0:
        print(nm.stdout)
    else:
        print(nm.stderr)
```

This module is also able to trigger a callback function provided by the user. This callback will be triggered each time nmap returns data to the lib. It is to note that the lib forces nmap to return its status (progress and etc) every two seconds. The event callback could then play around with those values while running.

To go a bit further, you can always use the threading capabilities of the NmapProcess class and run the class in the background:

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from libnmap.process import NmapProcess
from time import sleep

nm = NmapProcess(targets="scanme.nmap.org", options="-sV")
result = nm.run()

if result['rc'] == 0:
    print(nm.stdout)
else:
    print(nm.stderr)
```

```python
nmap_proc = NmapProcess(targets="scanme.nmap.org", options="-sT")
nmap_proc.run_background()

while nmap_proc.is_running():
    print("Nmap Scan running: ETC: {0} DONE: {1}%").format(nmap_proc.etc, nmap_proc.progress))
    sleep(2)

print("rc: {0} output: {1}".format(nmap_proc.rc, nmap_proc.summary))
```
The above code will print out the following on standard output:

```
(pydev)[dev@bouteille python-nmap-lib]$ python examples/proc_async.py
Nmap Scan running: ETC: 0 DONE: 0%
Nmap Scan running: ETC: 1369433951 DONE: 2.45%
Nmap Scan running: ETC: 1369433932 DONE: 13.55%
Nmap Scan running: ETC: 1369433930 DONE: 25.35%
Nmap Scan running: ETC: 1369433931 DONE: 33.40%
Nmap Scan running: ETC: 1369433932 DONE: 41.50%
Nmap Scan running: ETC: 1369433931 DONE: 52.90%
Nmap Scan running: ETC: 1369433931 DONE: 62.55%
Nmap Scan running: ETC: 1369433930 DONE: 75.55%
Nmap Scan running: ETC: 1369433931 DONE: 81.35%
Nmap Scan running: ETC: 1369433931 DONE: 99.99%
rc: 0 output: Nmap done at Sat May 25 00:18:51 2013; 1 IP address (1 host up) scanned in 22.02 seconds
```

Another and last example of a simple use of the NmapProcess class. The code below prints out the scan results a la nmap

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from libnmap.process import NmapProcess
from libnmap.parser import NmapParser, NmapParserException

# start a new nmap scan on localhost with some specific options
def do_scan(targets, options):
    parsed = None
    nmproc = NmapProcess(targets, options)
    rc = nmproc.run()
    if rc != 0:
        print("nmap scan failed: {0}".format(nmproc.stderr))
        print(type(nmproc.stdout))
    try:
        parsed = NmapParser.parse(nmproc.stdout)
    except NmapParserException as e:
        print("Exception raised while parsing scan: {0}".format(e.msg))
    return parsed

# print scan results from a nmap report
def print_scan(nmap_report):
    print("Starting Nmap {0} ( http://nmap.org ) at {1}".format(
        nmap_report.version,
        nmap_report.started))
    for host in nmap_report.hosts:
        if len(host.hostnames):
            tmp_host = host.hostnames.pop()
        else:
            tmp_host = host.address
        print("Nmap scan report for {0} ({1})".format(
            tmp_host,
            host.address))
```
The above code will print out the following on standard output:

```
Starting Nmap 5.51 ( http://nmap.org ) at Sat May 25 00:14:54 2013
Nmap scan report for localhost (127.0.0.1)
Host is up.
PORT STATE SERVICE
22/tcp open ssh (product: OpenSSH extrainfo: protocol 2.0 version: 5.3)
25/tcp open smtp (product: Postfix smtpd hostname: bouteille.localdomain)
80/tcp open http (product: nginx version: 1.0.15)
111/tcp open rpcbind (version: 2-4 extrainfo: rpc #10000)
631/tcp open ipp (product: CUPS version: 1.4)
Nmap done at Sat May 25 00:15:00 2013; 1 IP address (1 host up) scanned in 6.25 seconds
```

The full source code is available on GitHub. Please, do not hesitate to fork it and issue pull requests.

### 2.1.4 NmapProcess methods

```python
class libnmap.process.NmapProcess(targets='127.0.0.1', options='-sT', event_callback=None, safe_mode=True, fqp=None)
```

NmapProcess is a class which wraps around the nmap executable.

Consequently, in order to run an NmapProcess, nmap should be installed on the host running the script. By default NmapProcess will produce the output of the nmap scan in the nmap XML format. This could be then parsed out via the NmapParser class from libnmap.parser module.

```python
__init__(targets='127.0.0.1', options='-sT', event_callback=None, safe_mode=True, fqp=None)
```

Constructor of NmapProcess class.

**Parameters**

- `targets (string or list)` – hosts to be scanned. Could be a string of hosts separated with a coma or a python list of hosts/ip.
- `options` – list of nmap options to be applied to scan. These options are all documented in nmap’s man pages.
• **event_callback** – callable function which will be ran each time nmap process outputs data. This function will receive two parameters:
  1. the nmap process object
  2. the data produced by nmap process. See readme for examples.
• **safe_mode** – parameter to protect unsafe options like -oN, -oG, -iL, -oA,....
• **fqp** – full qualified path, if None, nmap will be searched in the PATH

**Returns** NmapProcess object

**command**
return the constructed nmap command or empty string if not constructed yet.

**current_task**
Accessor for the current NmapTask beeing run

**elapsed**
Accessor returning for how long the scan ran (in seconds)

**endtime**
Accessor for time when scan ended

**etc**
Accessor for estimated time to completion

**get_command_line**()
Public method returning the reconstructed command line ran via the lib

**has_failed**()
Checks if nmap has failed.

**has_terminated**()
Checks if nmap has terminated. Could have failed or succeeded

**is_running**()
Checks if nmap is still running.

**is_successful**()
Checks if nmap terminated successfully.

**options**
Provides the list of options for that scan

---

2.1. *libnmap.process*
**Returns** list of string (nmap options)

**progress**
Accessor for progress status in percentage

**Returns** percentage of job processed.

**rc**
Accessor for nmap execution’s return code

**Returns** nmap execution’s return code

**run()**
Public method which is usually called right after the constructor of NmapProcess. This method starts the nmap executable’s subprocess. It will also bind a Process that will read from subprocess’ stdout and stderr and push the lines read in a python queue for further processing. This processing is waken-up each time data is pushed from the nmap binary into the stdout reading routine. Processing could be performed by a user-provided callback. The whole NmapProcess object could be accessible asynchronously.

**return:** return code from nmap execution

**run_background()**
run nmap scan in background as a thread. For privileged scans, consider NmapProcess.sudo_run_background()

**starttime**
Accessor for time when scan started

**Returns** string. Unix timestamp

**state**
Accessor for nmap execution state. Possible states are:

* self.READY
* self.RUNNING
* self.FAILED
* self.CANCELLED
* self.DONE

**Returns** integer (from above documented enum)

**stderr**
Accessor for nmap standard error

**Returns** output from nmap when errors occurred.

**Return type** string

**stdout**
Accessor for nmap standard output

**Returns** output from nmap scan in XML

**Return type** string

**stop()**
Send KILL -15 to the nmap subprocess and gently ask the threads to stop.

**sudo_run (run_as='root')**
Public method enabling the library’s user to run the scan with privileges via sudo. The sudo configuration
should be set manually on the local system otherwise sudo will prompt for a password. This method alters
the command line by prefixing the sudo command to nmap and will then call self.run()

Parameters run_as – user name to which the lib needs to sudo to run the scan

Returns return code from nmap execution

**sudo_run_background** *(run_as='root')*
Public method enabling the library’s user to run in background a nmap scan with priviledges via sudo. The
sudo configuration should be set manually on the local system otherwise sudo will prompt for a password.
This method alters the command line by prefixing the sudo command to nmap and will then call self.run()

Parameters run_as – user name to which the lib needs to sudo to run the scan

Returns return code from nmap execution

**summary**
Accessor returning a short summary of the scan’s results

Returns string

**targets**
Provides the list of targets to scan

Returns list of string

**tasks**
Accessor returning for the list of tasks ran during nmap scan

Returns dict of NmapTask object

**version**
Accessor for nmap binary version number

Returns version number of nmap binary

Return type string

### 2.1.5 NmapTask methods

**class** libnmap.process.NmapTask *(name, starttime=0, extrainfo='')*

NmapTask is a internal class used by process. Each time nmap starts a new task during the scan, a new class
will be instanciated. Classes examples are: “Ping Scan”, “NSE script”, “DNS Resolve”... To each class an
estimated time to complete is assigned and updated at least every second within the NmapProcess. A property
NmapProcess.current_task points to the running task at time T and a dictionnary NmapProcess.tasks with “task
name” as key is built during scan execution

### 2.2 libnmap.parser

#### 2.2.1 Purpose of libnmap.parser

This modules enables you to parse nmap scans’ output. For now on, only XML parsing is supported. NmapParser is a
factory which will return a NmapReport, NmapHost or NmapService object. All these objects’ API are documented.
The module is capable of parsing:

- a complete nmap XML scan report
- an incomplete/interrupted nmap XML scan report
libnmap Documentation, Release 0.2

- partial nmap xml tags: <host>, <ports> and <port>

Input the above capabilities could be either a string or a file path.

Based on the provided data, NmapParse.parse() could return the following:

- NmapReport object: in case a full nmap xml/dict report was provided
- NmapHost object: in case a nmap xml <host> section was provided
- NmapService object: in case a nmap xml <port> section was provided
- Python dict with following keys: ports and extraports; python lists.

### 2.2.2 Using libnmap.parser module

NmapParser parse the whole data and returns nmap objects usable via their documented API.

The NmapParser should never be instanciated and only the following methods should be called:

- NmapParser.parse(string)
- NmapParser.parse_fromfile(file_path)
- NmapParser.parse_fromstring(string)

All of the above methods can receive as input:

- a full XML nmap scan result and returns a NmapReport object
- a scanned host in XML (<host>...</host> tag) and will return a NmapHost object
- a list of scanned services in XML (<ports>...</ports> tag) and will return a python array of NmapService objects
- a scanned service in XML (<port>...</port> tag) and will return a NmapService object

Small example:

```python
from libnmap.parser import NmapParser

nmap_report = NmapParser.parse_fromfile('libnmap/test/files/1_os_banner_scripts.xml')
print "Nmap scan summary: {0}".format(nmap_report.summary)
```

Basic usage from a processed scan:

```python
from libnmap.process import NmapProcess
from libnmap.parser import NmapParser

nm = NmapProcess("127.0.0.1, scanme.nmap.org")
nm.run()

nmap_report = NmapParser.parse(nm.stdout)

for scanned_hosts in nmap_report.hosts:
    print scanned_hosts
```

For more details on using the results from NmapParser, refer to the API of class: NmapReport, NmapHost, NmapService.

### 2.2.3 NmapParser methods

```python
class libnmap.parser.NmapParser
```
classmethod parse(nmap_data=None, data_type='XML', incomplete=False)

Generic class method of NmapParser class.

The data to be parsed does not need to be a complete nmap scan report. You can possibly give
<hosts>...</hosts> or <port> XML tags.

Parameters

- nmap_data (string) – any portion of nmap scan result. nmap_data should always be
  a string representing a part or a complete nmap scan report.
- data_type (string ("XML"/"JSON"/"YAML")) – specifies the type of data to
  be parsed.
- incomplete (boolean) – enable you to parse interrupted nmap scans and/or incom-
  plete nmap xml blocks by adding a </nmaprun> at the end of the scan.

As of today, only XML parsing is supported.

Returns NmapObject (NmapHost, NmapService or NmapReport)

classmethod parse_fromdict(rdict)

Strange method which transforms a python dict representation of a NmapReport and turns it into an
NmapReport object. Needs to be reviewed and possibly removed.

Parameters rdict (dict) – python dict representation of an NmapReport

Returns NmapReport

classmethod parse_fromfile(nmap_report_path, data_type='XML', incomplete=False)

Call generic cls.parse() method and ensure that a correct file path is given as argument. If not, an exception
is raised.

Parameters

- nmap_data – Same as for parse(). Any portion of nmap scan reports could be passed as
  argument. Data type _must_ be a valid path to a file containing nmap scan results.
- data_type – Specifies the type of serialization in the file.
- incomplete (boolean) – enable you to parse interrupted nmap scans and/or incom-
  plete nmap xml blocks by adding a </nmaprun> at the end of the scan.

Returns NmapObject

classmethod parse_fromstring(nmap_data, data_type='XML', incomplete=False)

Call generic cls.parse() method and ensure that a string is passed on as argument. If not, an exception is
raised.

Parameters

- nmap_data (string) – Same as for parse(), any portion of nmap scan. Reports could
  be passed as argument. Data type _must_ be a string.
- data_type – Specifies the type of data passed on as argument.
- incomplete (boolean) – enable you to parse interrupted nmap scans and/or incom-
  plete nmap xml blocks by adding a </nmaprun> at the end of the scan.

Returns NmapObject
2.3 libnmap.objects

2.3.1 Using libnmap.objects module

This module contains the definition and API of all “NmapObjects” which enables user to manipulate nmap data:

1. NmapReport
2. NmapHost
3. NmapService

The three objects above are the most common one that one would manipulate. For more advanced usage, the following objects might be useful

1. NmapOSFingerprint (contains: NmapOSMatch, NmapOSClass, OSFPPortUsed)
2. CPE (Common platform enumeration contained in NmapService or NmapOSClass)

The following structure applies by default:

**NmapReport contains:**

- Scan “header” data (start time, nmap command, nmap version, ...)
- List of NmapHosts (0 to X scanned hosts could be nested in a nmap report)
- Scan “footer” data (end time, summary, ...)

**NmapHost contains:**

- Host “header” data (state, hostnames, ip, ...)
- List of NmapService (0 to X scanned services could be nested in a scanned host)
- Host “footer” data (os version, fingerprint, uptime, ...)

**NmapService contains:**

- scan results for this service:
  - service state, service name
  - optional: service banner
  - optional: NSE scripts data

Each of the above-mentioned objects have a diff() method which enables the user of the lib the compare two different objects of the same type. If you read the code you’ll see the dirty trick with id() which ensures that proper objects are being compared. The logic of diff will certainly change overtime but the API (i/o) will be kept as is.

For more info on diff, please check the module’s documentation <diff>.

2.4 libnmap.objects.cpe

2.4.1 Using libnmap.objects.cpe module

TODO
2.4.2 CPE methods

```python
class libnmap.objects.cpe.CPE(cestring)
CPE class offers an API for basic CPE objects. These objects could be found in NmapService or in <os> tag
within NmapHost.

    Todo interpret CPE string and provide appropriate API

    cpestring
        Accessor for the full CPE string.

    get_edition()
        Returns the cpe edition

    get_language()
        Returns the cpe language

    get_part()
        Returns the cpe part (/o, /h, /a)

    get_product()
        Returns the product name

    get_update()
        Returns the update version

    get_vendor()
        Returns the vendor name

    get_version()
        Returns the version of the cpe

    is_application()
        Returns True if cpe describes an application

    is_hardware()
        Returns True if cpe describes a hardware

    is_operating_system()
        Returns True if cpe describes an operating system
```

2.5 libnmap.objects.host

2.5.1 Using libnmap.objects.host module

TODO

2.5.2 NmapHost methods

```python
class libnmap.objects.NmapHost(starttime='', endtime='', address=None, status=None, host-
    names=None, services=None, extras=None)
NmapHost is a class representing a host object of NmapReport

    address
        Accessor for the IP address of the scanned host

        Returns IP address as a string
```
changed (other)
    return the number of attribute who have changed :param other: NmapHost object to compare :return int
diff (other)
    Calls NmapDiff to check the difference between self and another NmapHost object.
    Will return a NmapDiff object.
    This objects return python set() of keys describing the elements which have changed, were added, removed or kept unchanged.

    Parameters other – NmapHost to diff with
    Returns NmapDiff object
distance
    Number of hops to host
    Returns int
distance
    Number of hops to host
    Returns int
eextraports_reasons
    dictionary containing reasons why extra ports scanned for which a common state, usually, closed was discovered.
    Returns array of dict containing keys ‘state’ and ‘count’ or None
eextraports_state
    dictionary containing state and amount of extra ports scanned for which a common state, usually, closed was discovered.
    Returns dict with keys ‘state’ and ‘count’ or None
get_dict()
    Return a dict representation of the object.
    This is needed by NmapDiff to allow comparaison
    :return dict
get_open_ports()
    Same as get_ports() but only for open ports
    Returns list: of tuples (port,’proto’) ie: [(22,’tcp’),(25,’tcp’)]
get_ports()
    Retrieve a list of the port used by each service of the NmapHost
    Returns list: of tuples (port,’proto’) ie: [(22,’tcp’),(25,’tcp’)]
get_service(portno, protocol=’tcp’)
    Parameters
        • portno – int the portnumber
        • protocol=’tcp’ – string (’tcp’,’udp’)
    Returns NmapService or None
get_service_byid(service_id)
    Returns a NmapService by providing its id.
The id of a nmap service is a python tuple made of (protocol, port)

**hostnames**
Accessor returning the list of hostnames (array of strings).

  **Returns**  array of string

**id**
id of the host. Used for diff(ing) NmapObjects

  **Returns**  string

**ipsequence**
Return the class of ip sequence of the remote hosts.

  **Returns**  string

**ipv4**
Accessor for the IPv4 address of the scanned host

  **Returns**  IPv4 address as a string

**ipv6**
Accessor for the IPv6 address of the scanned host

  **Returns**  IPv6 address as a string

**is_up()**
method to determine if host is up or not

  **Returns**  bool

**lastboot**
Since when the host was booted.

  **Returns**  string

**mac**
Accessor for the MAC address of the scanned host

  **Returns**  MAC address as a string

**os_class_probabilities()**
Returns an array of possible OS class detected during the OS fingerprinting.

  **Returns**  Array of NmapOSClass objects

**os_fingerprint**
Returns the fingerprint of the scanned system.

  **Returns**  string

**os_fingerprinted**
Specify if the host has OS fingerprint data available

  **Returns**  Boolean

**os_match_probabilities()**
Returns an array of possible OS match detected during the OS fingerprinting

  **Returns**  array of NmapOSMatches objects

**os_ports_used()**
Returns an array of the ports used for OS fingerprinting

  **Returns**  array of ports used: [dict(portid: '22', proto: 'tcp', state: 'open')],
### 2.6 libnmap.objects.report

#### 2.6.1 Using libnmap.objects.report module

TODO

#### 2.6.2 NmapReport methods

```python
class libnmap.objects.NmapReport (raw_data=None)
```

NmapReport is the usual interface for the end user to read scans output.

A NmapReport as the following structure:

- Scan headers data
- A list of scanned hosts (NmapReport.hosts)
- Scan footer data

It is to note that each NmapHost comprised in NmapReport.hosts array contains also a list of scanned services (NmapService object).

This means that if NmapParser.parse*() is the input interface for the end user of the lib. NmapReport is certainly the output interface for the end user of the lib.
commandline
Accessor returning the full nmap command line fired.

    Returns  string

diff (other)
Calls NmapDiff to check the difference between self and another NmapReport object.
Will return a NmapDiff object.

    Returns  NmapDiff object
    Todo    remove is_consistent approach, diff() should be generic.

elapsed
Accessor returning the number of seconds the scan took

    Returns  float (0 >= or -1)

detime
Accessor returning a unix timestamp of when the scan ended.

    Returns  integer

destimestr
Accessor returning a human readable time string of when the scan ended.

    Returns  string

get_dict()
Return a python dict representation of the NmapReport object. This is used to diff() NmapReport objects
via NmapDiff.

    Returns  dict

get_host_byid(host_id)
Gets a NmapHost object directly from the host array by looking it up by id.

    Parameters  ip_addr(string) – ip address of the host to lookup
    Returns  NmapHost

get_raw_data()
Returns a dict representing the NmapReport object.

    Returns  dict
    Todo     deprecate. get rid of this ugliness.

hosts
Accessor returning an array of scanned hosts.
Scanned hosts are NmapHost objects.

    Returns  array of NmapHost

hosts_down
Accessor returning the number of host detected as ‘down’ during the scan.

    Returns  integer (0 >= or -1)

hosts_total
Accessor returning the number of hosts scanned in total.

    Returns  integer (0 >= or -1)
hosts_up
Accessor returning the number of host detected as ‘up’ during the scan.

    Returns  integer (0 >= or -1)

id
Dummy id() defined for reports.

is_consistent()
Checks if the report is consistent and can be diffed().
This needs to be rewritten and removed: diff() should be generic.

    Returns  boolean

save(backend)
This method gets a NmapBackendPlugin representing the backend.

    Parameters backend – libnmap.plugins.PluginBackend object.

Object created by BackendPluginFactory and enabling nmap reports to be saved/stored in any type of
backend implemented in plugins.
The primary key of the stored object is returned.

    Returns  str

scan_type
Accessor returning a string which identifies what type of scan was launched (syn, ack, tcp,...).

    Returns  string

started
Accessor returning a unix timestamp of when the scan was started.

    Returns  integer

summary
Accessor returning a string describing and summarizing the scan.

    Returns  string

version
Accessor returning the version of the nmap binary used to perform the scan.

    Returns  string

2.7 libnmap.objects.service

2.7.1 Using libnmap.objects.service module
TODO

2.7.2 NmapService methods

class libnmap.objects.NmapService(portid, protocol='tcp', state=None, service=None,
  owner=None, service_extras=None)

NmapService represents a nmap scanned service. Its id() is comprised of the protocol and the port.
Depending on the scanning options, some additional details might be available or not. Like banner or extra data
from NSE (nmap scripts).
**banner**
Accessor for the service’s banner. Only available if the nmap option `-sV` or similar was used.

    Returns string

**changed**(other)
Checks if a NmapService is different from another.

    Parameters other – NmapService
    Returns boolean

**cpelist**
Accessor for list of CPE for this particular service

**diff**(other)
Calls NmapDiff to check the difference between self and another NmapService object.

    Will return a NmapDiff object.
    This objects return python set() of keys describing the elements which have changed, were added, removed or kept unchanged.

    Returns NmapDiff object

**get_dict**()
Return a python dict representation of the NmapService object.

    This is used to diff() NmapService objects via NmapDiff.

    Returns dict

**id**
Accessor for the id() of the NmapService.

    This is used for diff()ing NmapService object via NmapDiff.

    Returns tuple

**open**()
Tells if the port was open or not

    Returns boolean

**owner**
Accessor for service owner if available

**port**
Accessor for port.

    Returns integer or -1

**protocol**
Accessor for protocol

    Returns string

**reason**
Accessor for service’s state reason (syn-ack, filtered,...)

    Returns string or empty if not applicable

**reason_ip**
Accessor for service’s state reason ip

    Returns string or empty if not applicable
reason_ttl
Accessor for service’s state reason ttl

Returns string or empty if not applicable

scripts_results
Gives a python list of the nse scripts results.
The dict key is the name (id) of the nse script and the value is the output of the script.

Returns dict

service
Accessory for service name.

Returns string or empty

service_dict
Accessory for service dictionary.

Returns dict or None

servicefp
Accessory for the service’s fingerprint if the nmap option -sV or -A is used

Returns string if available

state
Accessory for service’s state (open, filtered, closed,...)

Returns string

tunnel
Accessory for the service’s tunnel type if applicable and available from scan results

Returns string if available

## 2.8 libnmap.objects.os

### 2.8.1 Using libnmap.objects.os module

TODO

### 2.8.2 NmapOSFingerprint methods

class libnmap.objects.os.NmapOSFingerprint

NmapOSFingerprint is a easier API for using os fingerprinting. Data for OS fingerprint (<os> tag) is instanciated from a NmapOSFingerprint which is accessible in NmapHost via NmapHost.os

get_osmatch(osclass_obj)

This function enables NmapOSFingerprint to determine if an NmapOSClass object could be attached to an existing NmapOSMatch object in order to respect the common interface for the nmap xml version < 1.04 and >= 1.04

This method will return an NmapOSMatch object matching with the NmapOSClass provided in parameter (match is performed based on accuracy)

Returns NmapOSMatch object
ports_used
Return an array of OSFPPortUsed object with the ports used to perform the os fingerprint. This dict might contain another dict embedded containing the ports_reason values.

2.8.3 NmapOSMatch methods

class libnmap.objects.os.NmapOSMatch (osmatch_dict)
NmapOSMatch is an internal class used for offering results from an nmap os fingerprint. This common interfaces makes a compatibility between old nmap xml (<1.04) and new nmap xml versions (used in nmapv6 for instance).

In previous xml version, osclass tags from nmap fingerprints were not directly mapped to a osmatch. In new xml version, osclass could be embedded in osmatch tag.

The approach to solve this is to create a common class which will, for older xml version, match based on the accuracy osclass to an osmatch. If no match, an osmatch will be made up from a concat of os class attributes: vendor and osfamily. Unmatched osclass will have a line attribute of -1.

More info, see issue #26 or http://seclists.org/nmap-dev/2012/q2/252

accuracy
Accessor for accuracy

    Returns  int

add_osclass (osclass_obj)
Add a NmapOSClass object to the OSMatch object. This method is useful to implement compatibility with older versions of NMAP by providing a common interface to access os fingerprint data.

get_cpe()
This method return a list of cpe stings and not CPE objects as the NmapOSClass.cpelist property. This method is a helper to simplify data management.

For more advanced handling of CPE data, use NmapOSClass.cpelist and use the methods from CPE class

line
Accessor for line attribute as integer. value equals -1 if this osmatch holds orphans NmapOSClass objects. This could happen with older version of nmap xml engine (<1.04 (e.g: nmapv6)).

    Returns  int

name
Accessor for name attribute (e.g.: Linux 2.4.26 (Slackware 10.0.0))

osclasses
Accessor for all NmapOSClass objects matching with this OS Match

2.8.4 NmapOSClass methods

class libnmap.objects.os.NmapOSClass (osclass_dict)
NmapOSClass offers an unified API to access data from analysed osclass tag. As implemented in libnmap and newer version of nmap, osclass objects will always be embedded in a NmapOSMatch. Unmatched NmapOSClass will be stored in “dummy” NmapOSMatch objects which will have the particularity of have a line attribute of -1. On top of this, NmapOSClass will have optional CPE objects embedded.

accuracy
Accessor for OS class detection accuracy (int)

    Returns  int
**cpelist**
Returns a list of CPE Objects matching with this os class

**Returns** list of CPE objects

**Return type** Array

**description**
Accessor helper which returns a concataned string of the valuable attributes from NmapOSClass object

**Returns** string

**osfamily**
Accessor for OS family information (Windows, Linux,...)

**Returns** string

**osgen**
Accessor for OS class generation (7, 8, 2.4.X,...).

**Returns** string

**type**
Accessor for OS class type (general purpose,...)

**Returns** string

**vendor**
Accessor for vendor information (Microsoft, Linux,...)

**Returns** string

### 2.8.5 OSFPPortUsed methods

**class** `libnmap.objects.os.OSFPPortUsed(port_used_dict)`
Port used class: this enables the user of NmapOSFingerprint class to have a common and clear interface to access portused data which were collected and used during os fingerprint scan

**portid**
Accessor for the referenced port number used

**proto**
Accessor for the portused protocol (tcp, udp,...)

**state**
Accessor for the portused state (closed, open,...)

### 2.9 libnmap.diff

#### 2.9.1 Using libnmap.diff module

This modules enables the user to diff two NmapObjects: NmapService, NmapHost, NmapReport.

The constructor returns a NmapDiff object which he can then use to call its inherited methods:

- `added()`
- `removed()`
- `changed()`
unchanged()

Those methods return a python set() of keys which have been changed/added/removed/unchanged from one object to another. The keys of each objects could be found in the implementation of the get_dict() methods of the compared objects.

The example below is a heavy version of going through all nested objects to see what has changed after a diff:

```python
#!/usr/bin/env python
from libnmap.parser import NmapParser

rep1 = NmapParser.parse_fromfile('libnmap/test/files/1_hosts.xml')
rep2 = NmapParser.parse_fromfile('libnmap/test/files/1_hosts_diff.xml')

rep1_items_changed = rep1.diff(rep2).changed()
changed_host_id = rep1_items_changed.pop().split('::')[1]
changed_host1 = rep1.get_host_byid(changed_host_id)
changed_host2 = rep2.get_host_byid(changed_host_id)
host1_items_changed = changed_host1.diff(changed_host2).changed()
changed_service_id = host1_items_changed.pop().split('::')[1]
changed_service1 = changed_host1.get_service_byid(changed_service_id)
changed_service2 = changed_host2.get_service_byid(changed_service_id)
servicel_items_changed = changed_service1.diff(changed_service2).changed()

for diff_attr in servicel_items_changed:
    print "diff({0}, {1}) [{2}:{3}] [{4}:{5}]".format(changed_service1.id,
                                                   changed_service2.id,
                                                   diff_attr,
                                                   getattr(changed_service1, diff_attr),
                                                   diff_attr,
                                                   getattr(changed_service2, diff_attr))
```

This outputs the following line:

```bash
(pydev)$ python /tmp/z.py
diff(tcp.3306, tcp.3306) [state:open] [state:filtered]
```

Of course, the above code is quite ugly and heavy but the idea behind diff was to be as generic as possible in order to let the user of the lib defines its own algorithms to extract the data.

A less manual and more clever approach would be to recursively retrieve the changed attributes and values of nested objects. Below, you will find a small code example doing it

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from libnmap.parser import NmapParser

def nested_obj(objname):
    rval = None
    splitted = objname.split("::")
    if len(splitted) == 2:
        rval = splitted
    return rval
```

2.9. libnmap.diff
def print_diff_added(obj1, obj2, added):
    for akey in added:
        nested = nested_obj(akey)
        if nested is not None:
            if nested[0] == 'NmapHost':
                subobj1 = obj1.get_host_byid(nested[1])
            elif nested[0] == 'NmapService':
                subobj1 = obj1.get_service_byid(nested[1])
            print("+ {0}".format(subobj1))
        else:
            print("+ {0} {1}: {2}".format(obj1, akey, getattr(obj1, akey)))

def print_diff_removed(obj1, obj2, removed):
    for rkey in removed:
        nested = nested_obj(rkey)
        if nested is not None:
            if nested[0] == 'NmapHost':
                subobj2 = obj2.get_host_byid(nested[1])
            elif nested[0] == 'NmapService':
                subobj2 = obj2.get_service_byid(nested[1])
            print("- {0}".format(subobj2))
        else:
            print("- {0} {1}: {2}".format(obj2, rkey, getattr(obj2, rkey)))

def print_diff_changed(obj1, obj2, changes):
    for mkey in changes:
        nested = nested_obj(mkey)
        if nested is not None:
            if nested[0] == 'NmapHost':
                subobj1 = obj1.get_host_byid(nested[1])
                subobj2 = obj2.get_host_byid(nested[1])
            elif nested[0] == 'NmapService':
                subobj1 = obj1.get_service_byid(nested[1])
                subobj2 = obj2.get_service_byid(nested[1])
            print_diff(subobj1, subobj2)
        else:
            print("~ {0} {1}: {2} => {3}".format(obj1, mkey,
                                                  getattr(obj2, mkey),
                                                  getattr(obj1, mkey)))

def print_diff(obj1, obj2):
    ndiff = obj1.diff(obj2)
    print_diff_changed(obj1, obj2, ndiff.changed())
    print_diff_added(obj1, obj2, ndiff.added())
    print_diff_removed(obj1, obj2, ndiff.removed())

def main():
    newrep = NmapParser.parse_fromfile('libnmap/test/files/2_hosts_achange.xml')
    oldrep = NmapParser.parse_fromfile('libnmap/test/files/1_hosts.xml')
    print_diff(newrep, oldrep)
if __name__ == "__main__":
    main()

This code will output the following:

~ NmapReport: started at 1361737906 hosts up 2/2 hosts_total: 1 => 2
~ NmapReport: started at 1361737906 hosts up 2/2 commandline: nmap -sT -vv -oX 1_hosts.xml localhost => nmap -sS -vv -oX 2_hosts.xml localhost scanme.nmap.org
~ NmapReport: started at 1361737906 hosts up 2/2 hosts_up: 1 => 2
~ NmapService: [closed 25/tcp smtp ()] state: open => closed
+ NmapService: [open 23/tcp telnet ()]
- NmapService: [open 111/tcp rpcbind ()]
~ NmapReport: started at 1361737906 hosts up 2/2 scan_type: connect => syn
~ NmapReport: started at 1361737906 hosts up 2/2 elapsed: 0.14 => 134.36
+ NmapHost: [74.207.244.221 (scanme.nmap.org scanme.nmap.org) - up]

Note that, in the above example, lines prefixed with:

1. '~' means values changed
2. '+' means values were added
3. '-' means values were removed

### 2.9.2 NmapDiff methods

```python
class libnmap.diff.NmapDiff(nmap_obj1, nmap_obj2)
    NmapDiff compares two objects of same type to enable the user to check:
    • what has changed
    • what has been added
    • what has been removed
    • what was kept unchanged

    NmapDiff inherit from DictDiffer which makes the actual comparaison. The different methods from DictDiffer
    used by NmapDiff are the following:
    • NmapDiff.changed()
    • NmapDiff.added()
    • NmapDiff.removed()
    • NmapDiff.unchanged()

    Each of the returns a python set() of key which have changed in the compared objects. To check the dif-
    ferent keys that could be returned, refer to the get_dict() method of the objects you which to compare (i.e: libnmap.objects.NmapHost, NmapService,...).
```

### 2.10 libnmap.plugins.s3.NmapS3Plugin

#### 2.10.1 Using libnmap.plugins.s3

This modules enables the user to directly use S3 buckets to store and retrieve NmapReports.
2.10.2 NmapS3Plugin methods
Indices and tables

- genindex
- modindex
- search
libnmap.diff, 25
libnmap.objects, 18
libnmap.objects.cpe, 13
libnmap.objects.os, 20
libnmap.parser, 10
libnmap.process, 6
Symbols

__init__() (libnmap.process.NmapProcess method), 6

A
accuracy (libnmap.objects.os.NmapOSClass attribute), 21
accuracy (libnmap.objects.os.NmapOSMatch attribute), 21
add_osclass() (libnmap.objects.os.NmapOSMatch method), 21
address (libnmap.objects.NmapHost attribute), 13

B
banner (libnmap.objects.NmapService attribute), 18

C
changed() (libnmap.objects.NmapHost method), 13
changed() (libnmap.objects.NmapService method), 19
command (libnmap.process.NmapProcess attribute), 7
commandline (libnmap.objects.NmapReport attribute), 16
CPE (class in libnmap.objects.cpe), 13
cpelist (libnmap.objects.NmapOSClass attribute), 21
cpestring (libnmap.objects.cpe.CPE attribute), 13
current_task (libnmap.process.NmapProcess attribute), 7

D
description (libnmap.objects.os.NmapOSClass attribute), 22
diff() (libnmap.objects.NmapHost method), 14
diff() (libnmap.objects.NmapReport method), 17
diff() (libnmap.objects.NmapService method), 19
distance (libnmap.objects.NmapHost attribute), 14

e
elapsed (libnmap.objects.NmapReport attribute), 17
elapsed (libnmap.process.NmapProcess attribute), 7
endtime (libnmap.objects.NmapHost attribute), 14
endtime (libnmap.objects.NmapProcess attribute), 7
endtime() (libnmap.objects.NmapProcess attribute), 7
extraports_reasons (libnmap.process.NmapProcess attribute), 7
extraports_state (libnmap.objects.NmapHost attribute), 14
e etc (libnmap.process.NmapProcess attribute), 7
e

G
get_command_line() (libnmap.process.NmapProcess method), 7
get_cpe() (libnmap.objects.os.NmapOSMatch method), 21
get_dict() (libnmap.objects.NmapHost attribute), 14
get_dict() (libnmap.objects.NmapService method), 19
get_dict() (libnmap.objects.NmapService method), 19
get_dict() (libnmap.objects.NmapReport method), 17
get_dict() (libnmap.objects.NmapHost method), 14
get_host_byid() (libnmap.objects.NmapReport method), 17
get_language() (libnmap.objects.cpe.CPE attribute), 13
get_open_ports() (libnmap.objects.NmapHost attribute), 14
get_osmatch() (libnmap.objects.os.NmapOSFingerprint method), 20
get_part() (libnmap.objects.cpe.CPE method), 13
get_ports() (libnmap.objects.NmapHost method), 14
get_product() (libnmap.objects.cpe.CPE method), 13
get_raw_data() (libnmap.objects.NmapReport method), 17
get_service() (libnmap.objects.NmapHost method), 14
get_service_byid() (libnmap.objects.NmapHost method), 14
get_update() (libnmap.objects.cpe.CPE method), 13
get_vendor() (libnmap.objects.cpe.CPE method), 13
get_version() (libnmap.objects.cpe.CPE method), 13

H
has_failed() (libnmap.process.NmapProcess method), 7
has_terminated() (libnmap.process.NmapProcess method), 7
hostnames (libnmap.objects.NmapHost attribute), 15
hosts (libnmap.objects.NmapReport attribute), 17
hosts_down (libnmap.objects.NmapReport attribute), 17
hosts_total (libnmap.objects.NmapReport attribute), 17
hosts_up (libnmap.objects.NmapReport attribute), 17
id (libnmap.objects.NmapHost attribute), 15
id (libnmap.objects.NmapReport attribute), 18
id (libnmap.objects.NmapService attribute), 19
ipsequence (libnmap.objects.NmapHost attribute), 15
ipv4 (libnmap.objects.NmapHost attribute), 15
ipv6 (libnmap.objects.NmapHost attribute), 15
is_application() (libnmap.objects.cpe.CPE method), 13
is_consistent() (libnmap.objects.NmapReport method), 18
is_hardware() (libnmap.objects.cpe.CPE method), 13
is_operating_system() (libnmap.objects.cpe.CPE method), 13
is_running() (libnmap.process.NmapProcess method), 7
is_successful() (libnmap.process.NmapProcess method), 7
is_up() (libnmap.objects.NmapHost method), 15
lastboot (libnmap.objects.NmapHost attribute), 15
libnmap.diff (module), 25
libnmap.objects (module), 13, 16, 18
libnmap.objects.cpe (module), 13
libnmap.objects.os (module), 20
libnmap.parser (module), 10
libnmap.process (module), 6
line (libnmap.objects.os.NmapOSMatch attribute), 21
mac (libnmap.objects.NmapHost attribute), 15
name (libnmap.objects.os.NmapOSMatch attribute), 21
NmapDiff (class in libnmap.diff), 25
NmapHost (class in libnmap.objects), 13
NmapOSClass (class in libnmap.objects.os), 21
NmapOSFingerprint (class in libnmap.objects.os), 20
NmapOSMatch (class in libnmap.objects.os), 21
NmapParser (class in libnmap.parser), 10
NmapProcess (class in libnmap.process), 6
NmapReport (class in libnmap.objects), 16
NmapService (class in libnmap.objects), 18
NmapTask (class in libnmap.process), 9
open() (libnmap.objects.NmapService method), 19
options (libnmap.process.NmapProcess attribute), 7
os_class_probabilities() (libnmap.objects.NmapHost method), 15
os_fingerprint (libnmap.objects.NmapHost attribute), 15
os_fingerprinted (libnmap.objects.NmapHost attribute), 15
os_match_probabilities() (libnmap.objects.NmapHost method), 15
os_ports_used() (libnmap.objects.NmapHost method), 15
osclasses (libnmap.objects.os.NmapOSMatch attribute), 21
osfamily (libnmap.objects.os.NmapOSClass attribute), 22
OSFPPortUsed (class in libnmap.objects.os), 22
owner (libnmap.objects.NmapService attribute), 19
parse() (libnmap.parser.NmapParser class method), 10
parse_fromdict() (libnmap.parser.NmapParser class method), 11
parse_fromfile() (libnmap.parser.NmapParser class method), 11
parse_fromstring() (libnmap.parser.NmapParser class method), 11
port (libnmap.objects.NmapService attribute), 19
portid (libnmap.objects.os.OSFPPortUsed attribute), 22
ports_used (libnmap.objects.os.NmapOSFingerprint attribute), 20
progress (libnmap.process.NmapProcess attribute), 8
proto (libnmap.objects.os.OSFPPortUsed attribute), 22
protocol (libnmap.objects.NmapService attribute), 19
rc (libnmap.process.NmapProcess attribute), 8
reason (libnmap.objects.NmapService attribute), 19
reason_ip (libnmap.objects.NmapService attribute), 19
reason_ttl (libnmap.objects.NmapService attribute), 19
run() (libnmap.process.NmapProcess method), 8
run_background() (libnmap.process.NmapProcess method), 8
save() (libnmap.objects.NmapReport method), 18
scan_type (libnmap.objects.NmapReport attribute), 18
scripts_results (libnmap.objects.NmapHost attribute), 15
scripts_results (libnmap.objects.NmapService attribute), 20
service (libnmap.objects.NmapService attribute), 20
service_dict (libnmap.objects.NmapService attribute), 20
service_fp (libnmap.objects.NmapService attribute), 20
services (libnmap.objects.NmapHost attribute), 20
started (libnmap.objects.NmapReport attribute), 16
starttime (libnmap.objects.NmapHost attribute), 16
starttime (libnmap.process.NmapProcess attribute), 8
state (libnmap.objects.NmapService attribute), 20
state (libnmap.objects.os.OSFPPortUsed attribute), 22
state (libnmap.process.NmapProcess attribute), 8
status (libnmap.objects.NmapHost attribute), 16
stderr (libnmap.process.NmapProcess attribute), 8
stdout (libnmap.process.NmapProcess attribute), 8
stop() (libnmap.process.NmapProcess method), 8
sudo_run() (libnmap.process.NmapProcess method), 8
sudo_run_background() (libnmap.process.NmapProcess method), 9
summary (libnmap.objects.NmapReport attribute), 18
summary (libnmap.process.NmapProcess attribute), 9
T
targets (libnmap.process.NmapProcess attribute), 9
tasks (libnmap.process.NmapProcess attribute), 9
tcpsequence (libnmap.objects.NmapHost attribute), 16
tunnel (libnmap.objects.NmapService attribute), 20
type (libnmap.objects.os.NmapOSClass attribute), 22
U
uptime (libnmap.objects.NmapHost attribute), 16
V
vendor (libnmap.objects.NmapHost attribute), 16
vendor (libnmap.objects.os.NmapOSClass attribute), 22
version (libnmap.objects.NmapReport attribute), 18
version (libnmap.process.NmapProcess attribute), 9