Referer:

- https://xael.org/pages/python-nmap-en.html
- scantastic-tool https://github.com/maK-/scantastic-tool
- PublicMonitors 0.1 https://github.com/grayddq/PublicMonitors.git
- https://github.com/MyKings/python-masscan
- 目标端口扫描+系统服务指纹识别 https://github.com/ring04h/wyportmap

python-nmap

```
>>> import nmap
>>> nm = nmap.PortScanner()
>>> nm.scan('127.0.0.1', '22-443')
>>> nm.command_line()
```

结果解析:

```
def markScanport(self):
    nm = nmap.PortScanner()
    info = nm.scan(hosts=self.Mask,arguments='-p %s'%str(self.single_port))
    nmap_info = info['nmap']
    command_line = nmap_info['command_line']
    scaninfo = nmap_info['scaninfo']
    scan = info['scan']
    output.dataOut('[*] commond: %s'%command_line)
    output.dataOut('[*] scaninfo:'+str(scaninfo)+'\n')
    for ip in scan:
        hoststate = scan[ip]['status']['state']
        portstate = scan[ip]['tcp'][self.single_port]['state']
        portname = scan[ip]['tcp'][self.single_port]['name']
        mes = '{0} is {1},{2}/tcp {3}
{4}'.format(ip,hoststate,self.single_port,portstate,portname)
```

```
        output.maskOpenPort(mes)
        portscan_result.append(mes)
```

libnmap

Referer:

- libnmap文档:https://libnmap.readthedocs.io/en/latest/objects/nmapservice.html

1. 安装

   apt install python-libnmap

2. 一些功能:

- 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

3. 和python-nmap的对比:
4. python-nmap它的代码统一写在一个单文件中，主要定义了三种扫描类、一个目标字典类和一个扫描错误类。基础类PortScanner，该类主要提供扫描方法、扫描结果xml输出、扫描CSV结果的文本输出功能。从源码可以看出，它主要就是通过python调用nmap进行扫描，并没有太多其他的实现，所有的nmap需求通过nmap自身的参数来实现。
5. 相比之下，libnmap的函数实现就要复杂一些，它定义了包括对象、插件和基础功能在内的多个函数。基础功能包process，主体类NmapProcess，在该类中，定义类包括扫描行为、输出结果、后台执行、sudo执行、执行时间状态等多种方法。

process 模块

- is_running() 判断nmap是否在运行
- run_background() 作为线程在后台运行nmap扫描。 对于特权扫描，请考虑NmapProcess.sudo_run_background（）
- rc nmap执行的返回码的访问器
- state: 扫描状态
- etc: 扫描时间戳
- process: 扫描进度
- stdout: 扫描结果
- command: 扫描命令

```
self.READY
self.RUNNING
self.FAILED
self.CANCELLED
self.DONE
```

- summary 访问者返回扫描结果的简短摘要，eg：`Nmap done at Mon Apr 1 21:05:25 2019; 1 IP address (1 host up) scanned in 18.39 seconds`

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

from libnmap.process import NmapProcess
from time import sleep
import time


def do_nmap_scan(target):
    nmap_proc = NmapProcess(targets=target, options="-p1-65535")
    nmap_proc.run_background()
    print nmap_proc.command
    while nmap_proc.is_running():
        print("Nmap Scan running:: {0} DONE: {1}%".format(time.strftime("%H-%M-
%S",time.localtime(float(nmap_proc.etc))),
                                                              nmap_proc.progress))
        sleep(3) # nmap not stop, just slow down the stdout

    print("rc: {0} output: {1}".format(nmap_proc.rc, nmap_proc.summary))
    return nmap_proc.stdout


if __name__ == '__main__':
    start = time.time()
    print do_nmap_scan('127.0.0.1/30')
    print 'time:',time.time()-start

>>>
root@lj /m/h/F/s/p/t/libnmap# python do_nmap.py
 /usr/bin/nmap -oX - -vvv --stats-every 1s -sS -T4 -p1-65535 127.0.0.1/30
Nmap Scan running:: 08-00-00 DONE: 0%
Nmap Scan running:: 09-43-30 DONE: 66.67%
Nmap Scan running:: 09-43-35 DONE: 45.90%
Nmap Scan running:: 09-43-41 DONE: 52.30%
Nmap Scan running:: 09-43-45 DONE: 58.68%
Nmap Scan running:: 09-43-48 DONE: 65.03%
Nmap Scan running:: 09-43-51 DONE: 71.44%
Nmap Scan running:: 09-43-53 DONE: 77.82%
Nmap Scan running:: 09-43-55 DONE: 83.94%
Nmap Scan running:: 09-43-58 DONE: 86.07%
Nmap Scan running:: 09-44-01 DONE: 86.27%
Nmap Scan running:: 09-44-05 DONE: 86.35%
```

parse模块

NmapParser解析整个数据并返回可通过其记录的API使用的nmap对象

函数:

```
//parse()直接解析NmapProcess的输出
nmap_report = NmapParser.parse(nm.stdout)
//从xml文件中解析
nmap_report = NmapParse.parse_fromfile(xmlfile)
```

基本使用:

```
def do_nmap_parse():
    nm = NmapProcess("127.0.0.1/30")
    nm.run()

    nmap_report = NmapParser.parse(nm.stdout)

    for host in nmap_report.hosts:
        print host,host.status,host.address,host.endtime,host.services #输出host的一些属性
        for service in host.services: #遍历host的services列表
            print service

>>>
NmapHost: [127.0.0.0 () - down] down 127.0.0.0  []
NmapHost: [127.0.0.1 (localhost) - up] up 127.0.0.1 1554867043 [NmapService: [open 25/tcp
smtp ()], NmapService: [open 80/tcp http ()], NmapService: [open 587/tcp submission ()],
NmapService: [open 631/tcp ipp ()], NmapService: [open 1080/tcp socks ()], NmapService:
[open 3306/tcp mysql ()], NmapService: [open 9200/tcp wap-wsp ()]]
NmapService: [open 25/tcp smtp ()]
NmapService: [open 80/tcp http ()]
NmapService: [open 587/tcp submission ()]
NmapService: [open 631/tcp ipp ()]
NmapService: [open 1080/tcp socks ()]
NmapService: [open 3306/tcp mysql ()]
NmapService: [open 9200/tcp wap-wsp ()]
```

解析出来的每个scanned_hosts都是一个libnmap.objects.host对象

host对象

属性如下:

- address: IP address of the scanned host 返回值: string
- endtime:结束时间 返回值: string
- status: host’ s status (up, down, unknown...) 返回值: string
- services: 返回一组NmapService对象列表，每个属性都是一个service对象.
- os: 只有当存在 `-O` 参数时才有:

```
print host.os #输出host的一些属性
print host.os.osmatch()

>>>
Linux 3.8 - 4.14: 100
  |__ os class: general purpose: Linux, Linux(3.X)
    |__ cpe:/o:linux:linux_kernel:3
  |__ os class: general purpose: Linux, Linux(4.X)
    |__ cpe:/o:linux:linux_kernel:4Fingerprints:
['Linux 3.8 - 4.14']
```

- scripts_results: 输出脚本扫描主机的结果

- get_open_ports()[source] Same as get_ports() but only for open ports

Returns: list: of tuples (port,’ proto’) ie:[(22,’ tcp’),(25, ‘tcp’)]

service对象:

- banner: 仅在使用nmap选项-sV或类似选项时可用
- changed: 检查NmapService是否与另一个不同

```
Parameters: other – NmapService
Returns:    boolean
```

- get_dict(): 返回NmapService对象的python dict表示。 这用于通过NmapDiff对nmapService对象进行diff ()

- open(): 判断端口是否打开

- port: Returns:integer or -1

- state: service’ s state (open, filtered, closed,...)

- service:具体的服务名称，比如ssh,ftp一类。 return:str

- scripts_results 输出脚本扫描端口的结果

插件

libnmap库的插件如下目录如下:

```
plugins
    backendplugin.py #插件基类，扩展的插件需要继承该基类
    backendpluginFactory.py #实现插件的动态调用
    es.py #es插件
    mongodb.py #mongo插件
    s3.py #s3插件
    sql.py #sqlalchemy插件
```

插件基类NmapBackendPlugin主要定义了实现的一些抽象方法: `insert(),delete(),get(),getall()`

新编写的插件都要实现这几个抽象方法。

主要看一下`backendpluginFactory.py`如何来实现的插件动态调用:

```
class BackendPluginFactory(object):
    """
        This is a backend plugin factory a backend instance MUST be
        created via the static method create()
        ie : mybackend = BackendPluginFactory.create()
    """
    @classmethod
    def create(cls, plugin_name="mongodb", **kwargs):
        """Import the needed lib and return an object NmapBackendPlugin
           representing the backend of your desire.
           NmapBackendPlugin is an abstract class, to know what argument
           need to be given, review the code of the subclass you need
           :param plugin_name: str : name of the py file without .py
           :return: NmapBackend (abstract class on top of all plugin)
        """
        backendplugin = None
        plugin_path = "libnmap.plugins.{0}".format(plugin_name)
        __import__(plugin_path)
        pluginobj = sys.modules[plugin_path]
        pluginclasses = inspect.getmembers(pluginobj, inspect.isclass)
        for classname, classobj in pluginclasses:
            if inspect.getmodule(classobj).__name__.find(plugin_path) == 0:
                try:
                    backendplugin = classobj(**kwargs)
                except Exception as error:
                    raise Exception("Cannot create Backend: {0}".format(error))
        return backendplugin
```

这儿的插件自动调用的方法值得学习，使用`__import__`导入插件路径后，通过inspect.getmembers()来获取插件里面的所有成员。

sqlaichemy插件

python-masscan

```
pip install python-masscan
```

usage:

```
import masscan


mas = masscan.PortScanner()
mas.scan('172.0.8.78/24', ports='22,80,8080')
print mas.scan_result
```

nmap 指纹

- nmap-service-probes 文件格式: https://nmap.org/book/vscan-fileformat.html
- nmap服务识别和操作系统探测 https://segmentfault.com/a/1190000011871145

nmap(一般在/usr/share/nmap目录下) 把主流服务的特征信息存储在 nmap-services-probes 和 nmap-os-db 这两个文件中。其中前者包含应用程序的特征信息，后者放置操作系统的特征信息。

片段

```
Probe TCP GetRequest q|GET / HTTP/1.0\r\n\r\n|
rarity 1
ports 1,70,79,80-
85,88,113,139,143,280,497,505,514,515,540,554,591,620,631,783,888,898,900,901,1026,1080,1
042,1214,1220,1234,1314,1344,1503,1610,1611,1830,1900,2001,2002,2030,2064,2160,2306,2396,
2525,2715,2869,3000,3002,3052,3128,3280,3372,3531,3689,3872,4000,4444,4567,4660,4711,5000
,5427,5060,5222,5269,5280,5432,5800-
5803,5900,5985,6103,6346,6544,6600,6699,6969,7002,7007,7070,7100,7402,7776,8000-
8010,8080-8085,8088,8118,8181,8880-
8888,9000,9001,9030,9050,9080,9090,9999,10000,10001,10005,11371,13013,13666,13722,14534,1
5000,17988,18264,31337,40193,50000,55555
sslports 443,993,995,1311,1443,3443,4443,5061,7443,8443,9443,10443,14443,44443,60443

...
match http m|^HTTP/1\.[01] \d\d\d.*?\r\nServer: nginx\r\n|s p/nginx/
cpe:/a:igor_sysoev:nginx/
softmatch http m|^HTTP/1\.[01] .*\r\nX-Powered-By: PHP/(\d[\w._-]+)|s i/PHP $1/
```

字段含义:

Probe(探测)

语法: `Probe <protocol> <probename> <probestring>` , 表示新起一个探测项。后面跟着的 `TCP GetRequest` 表示该探测项的种类和名字。 `q|GET / HTTP/1.0\r\n\r\n|` 表示探测时采取的动作，这里就是发送一个最小的 GET 请求。

- protocol:tcp 或者 udp
- probename: 特征名
- probestring: 特征字符串,告诉Nmap发送什么。它必须以q开头，然后是一个以定界开头和结尾的字符串的。分隔符之间是实际发送的字符串。 它的格式类似于C或Perl字符串，因为它允许以下标准转义字符 `\\ \0, \a, \b, \f, \n, \r, \t, \v` ,

eg:

```
Probe TCP GetRequest q|GET / HTTP/1.0\r\n\r\n|
Probe UDP DNSStatusRequest q|\0\0\x10\0\0\0\0\0\0\0\0\0|
Probe TCP NULL q||
```

第三个Probe TCP NULL q|| 值得介绍 这个 Probe 是为了检测出某些会主动给连上来的客户端发送数据的服务设置的。跟其他 Probe 不同，它不发送请求，只是干等 6 秒。当你给 nmap 抓包时，端口探测和服务识别间会相差 6 秒，就是这个探测导致的。

rarity(稀有度)
表示该探测的罕见性。nmap 默认只会执行 rarity 7 以下的探测，不过可以通过 --version-intensity N 选项更改。

ports
需要执行该探测端口列表。在发起识别服务的扫描之前，nmap 已经扫描出当前可用的开放端口列表了。对于开放列表中的端口，如果它在某个探测端口列表里，nmap 就会对该端口执行对应的探测。不用担心每个端口都执行许多探测会明显减慢扫描速度 —— nmap 会并行扫描多个端口。

sslports
类似于 ports，不同之处在于生效的时间。当 nmap 完成 tls 握手后，会根据 sslports 登记的配置去运行对应的探测。

match
语法: `match <service> <pattern> [<versioninfo>]`, 顾名思义，它们都是响应的匹配规则。其中这里列出的匹配规则表示，如果 GET 请求对应的响应匹配到 `^HTTP/1\.[01] \d\d\d.*?\r\nServer: nginx\r\n`，说明对应的服务是 Nginx。单个产品可能会有多个与之相关的规则，比如 nmap 会有以下方法去匹配 Nginx：

```
响应报头里包含 Server: nginx
错误页面中包含 <center>nginx</center>
构造一个畸形的 HTTP 请求，返回 <head><title>400 Bad Request</title></head>\r\n<h1>400 Bad
Request</h1>
向 HTTPS 端口发送 HTTP 请求，返回 HTTP/1.1 400 Bad Request\r\n.*<title>400 The plain HTTP
request was sent to HTTPS port</title>
```

- service: 探测的应用指纹名
- pattern: m 开头的是正则表达式，p 开头的是产品名字，v 开头的是版本号，cpe 开头的是 cpe 编号， i 开头表示附加的信息。

eg:

```
match ftp m/^220.*Welcome to .*Pure-?FTPd (\d\S+\s*)/ p/Pure-FTPd/ v/$1/
cpe:/a:pureftpd:pure-ftpd:$1/
match ssh m/^SSH-([\d.]+)-OpenSSH[_-]([\w.]+)\r?\n/i p/OpenSSH/ v/$2/ i/protocol $1/
cpe:/a:openbsd:openssh:$2/
```

```
match mysql m|^\x10\0\0\x01\xff\x13\x04Bad handshake$| p/MySQL/ cpe:/a:mysql:mysql/
match chargen m|@ABCDEFGHIJKLMNOPQRSTUVWXYZ|
match uucp m|^login: login: login: $| p/NetBSD uucpd/ o/NetBSD/ cpe:/o:netbsd:netbsd/a
match printer m|^([\w-_.]+): lpd: Illegal service request\n$| p/lpd/ h/$1/
match afs m|^[\d\D]{28}\s*(OpenAFS)([\d\.]{3}[^\s\0]*)\0| p/$1/ v/$2/
```

softmatch

语句最多只会匹配到一个服务。如果想匹配多个服务，比如 Nginx 代理的 PHP 应用，可以结合 softmmatch 使用。`最后输出结果里会多一个括号：8080/tcp open http nginx (PHP 5.6.25)`